

Solutions Sample

[See Also](#)


[List of Components](#)

The Visual FoxPro Solutions sample is a collection of independent samples designed to illustrate particular features of Visual FoxPro. As you go through these samples, you can see how to accomplish specific tasks in Visual FoxPro.

► To run the Solutions Sample application

- Enter the following in the Command window:

```
DO (HOME() + 'samples\solution\solution')
```

The samples are organized into the following categories: ActiveX, Controls, Databases, Forms, Menus, Reports, Toolbars, and WinAPI. To see all of the components in this treeview, choose the **Expand All**  button and scroll through the components.

When you select a component, you can:

- Read a brief description of the sample in the description area at the bottom of the main Solutions form,
- Run the sample by choosing **Run Sample**,
- See the code by choosing **See Code**.

After you close the form or designer that is opened, you are returned to the main Solutions form.

See Also

[How the Solutions Sample Works](#)

How the Solutions Sample Works

Each component of the Solutions sample is listed in SOLUTION.DBF. This table stores the following:

- The key and parent key ids required to add nodes to a treeview control.
- The image id in the ImageList control for the picture to be displayed beside the item in the treeview control.
- The file name and path of the components.
- The method to open when you choose **See Code**.
- The description to be displayed in the Visual FoxPro Solutions form.

There are two additional tables used to provide filtered list access to the components: XREF.DBF and REFTEXT.DBF.

The FillTree method of the SOLUTION.SCX reads the information from SOLUTION.DBF to fill the treeview control.

```
* FillTree
o = THIS.pgfl.pagTree.oleTree

SCAN
  IF ALLTRIM(parent) = '0'
    oNode = o.nodes.add(,1,ALLTRIM(key),ALLTRIM(text),,,)
  ELSE
    oNode = o.nodes.add(ALLTRIM(parent),4,ALLTRIM(key), ALLTRIM(text),,,)
  ENDIF
  * add images to the treeview
  IF !empty(image)
    oNode.Image = ALLTRIM(image)
  ENDIF
ENDSCAN
```

Forms

Each form in the Solutions suite contains an object based on the c_solutions class in SOLUTION.VCX. This class provides a single place for environment and localization settings. When the c_solutions object is destroyed, old settings are restored and, if the Solutions form exists, it is redisplayed.

The custom class is used in the Solutions sample because it provides greater flexibility: you can add it to existing forms, and, if you want, you can delete it from the forms in the Solutions suite and customize the forms for your own use. Instead of using a custom class to manage the environment settings, you can provide this functionality in one form class from which all forms in the Solutions sample are derived.

Form records in SOLUTION.DBF have a value of "F" stored to the type field. In the code associated with cmdRun and cmdSee, the form is run or modified.

```
* extract from cmdRun.Click
CASE solutions.type = "F" && form
  DO FORM (ALLTRIM(solutions.path) + "\" + ALLTRIM(solutions.file))
```

Note In the Data Environment of SOLUTION.SCX, the cursor for SOLUTION.DBF is given an alias of "Solutions."

Reports and Queries

Report records in SOLUTION.DBF have a value of "R" stored to the type field and queries have a

type of "Q". You can run the reports and queries in the code associated with cmdRun, and you can modify them in the code associated with cmdSee.

```
* extract from cmdRun
  CASE solutions.type = "R" && report
    REPORT FORM (ALLTRIM(solutions.path) + "\" + ALLTRIM(solutions.file))
PREVIEW NOCONSOLE
    THISFORM.Visible = .T.
```

Running a query opens the result set in a Browse window. By default, a Browse is displayed in the currently active window, which, in this sample would be the Solutions form. To display the Browse in a separate window, this code defines and activates another window, runs the query, then releases the separate window:

```
  CASE solutions.type = "Q" && query
    #DEFINE TITLE_LOC "Results of query "
    DEFINE WINDOW brow_wind FROM 1,1 TO 30, 100 TITLE TITLE_LOC +
UPPER(ALLTRIM(file)) + ".QPR " ;
    FLOAT GROW MINIMIZE ZOOM CLOSE FONT "Arial",10
    ACTIVATE WINDOW brow_wind NOSHOW
    DO (ALLTRIM(solutions.path) + "\" + ALLTRIM(solutions.file) + ".QPR")
    RELEASE WINDOW brow_wind
    THISFORM.Visible = .T.
ENDCASE
```

Components in the Solutions Sample Application



A

[Add and Remove Items in a Treeview Control](#)

[Add Items Interactively to a List Box](#)

[Add Menu Items at Run Time](#)

[Add New Items to a Combo Box](#)

[Allow Users to Choose List Values](#)

[Allow Users to Drag and Drop Controls](#)

[Automate a Microsoft Excel Spreadsheet](#)

[Automate a Microsoft Word Document in a Form](#)

[Automate Microsoft Word and Excel](#)

B

C

[Change Pages When a User Chooses a Button](#)

[Change the Number of Tabs at Run Time](#)

[Coordinate Menu Items and Toolbar Buttons](#)

[Create a 1-To-Many Data Entry Form](#)

[Create a Default Unique Id Value for a Field](#)

[Create a Query-By-Example Form](#)

[Create a Single Table Data Entry Form](#)

[Create an SDI Form](#)

[Create Small Indexes Using BINTOC\(\)](#)

[Create Dynamic Shortcut Menus](#)

[Customize the Open Dialog Box](#)

D

[Disable or Display a Check Beside a Menu Item](#)

[Display a Stop Watch](#)

[Display a System Clock](#)

[Display Calculated Values in a Column](#)

[Display Child Records from a Relationship](#)

[Display Controls in a Grid](#)

[Display Different Pages Without Tabs](#)

[Display Line Animation on a Form](#)

[Display Multiple Columns in a List Box](#)

[Display Pictures in a List](#)

[Display Pictures in an Image Control](#)

[Display Shortcut Menus](#)

[Display System Information](#)

[Draw Lines and Shapes on a Form](#)

[Dynamically Format Grid Columns](#)

E

[Edit a Memo Field or Text File](#)

[Execute Commands at Specified Intervals](#)

F

[Fill a List with Values from Different Sources](#)

[Format Input and Validate Data in a Text Box](#)

G

[Get Application Information from the Windows Registry](#)

[Get Version Information](#)

[Graph Equations on a Form](#)

H

I

[Index on Expressions](#)

[Interactively Build a SELECT Statement](#)

J

K

L

M

[Manipulate Display Characteristics of a Graph](#)

[Move Items Between List Boxes](#)

[Multiselect Items in a List Box](#)

N

[Nest Transactions](#)

O

[Open Dialog Box Flag Values](#)

[Open Multiple Files Interactively](#)

P

[Pass Parameters between Forms](#)

[Print a Percent of Total in a Report](#)

[Print Customer Mailing Labels](#)

[Print Data Dictionary Information](#)

[Print in Catalog Format](#)

[Print Product Orders By Quarter](#)

[Play an AVI File in an ActiveX Control](#)

[Play Multimedia Files Using MCI Commands](#)

[Present a User with Multiple Choices](#)

[Print an Invoice](#)

[Programmatically Check Table Properties](#)

[Programmatically Manipulate Objects](#)

[Programmatically Manipulate Text](#)

[Provide a Hierarchical Display of Items](#)

[Provide What's This Help on a Form](#)

Q

R

[Read and Write Visual FoxPro Registry Values](#)

[Read ODBC Registry Values](#)

[Refresh a Graph in a Form](#)

[Resize and Reposition Controls at Run Time](#)

[Return a Value from a Form](#)

[Run Multiple Instances of a Form](#)

S

[See Check Box Design Options](#)

[See Command Button Design Options](#)

[Select Customers in a Specific Country](#)

[Select Records from a Full Outer Join](#)

[Select Records from a Left Outer Join](#)

[Select Records from a Nested Join](#)

[Select Records from a Right Outer Join](#)

[Select Records from an Inner Join](#)

[Select Records from Both Inner and Outer Joins](#)

[Select the Ten Worst Selling Products](#)

[Select the Top Ten Best Selling Products](#)

[Send Mail](#)

[Sort List Box Items](#)

[Sort or Order A Table At Run Time](#)

T

U

[Use API Functions that Require a STRUCT](#)

[Use API Functions that Require Pointers to Arrays](#)

[Use Conditional Formatting in a Report](#)

[Use Slider and Status Bar Controls](#)

[Use the RichText Control](#)

V

[View Type Library Information](#)

W

X

Y

Z

Add and Remove Items in a Treeview Control

File: SAMPLES\SOLUTION\OLE\BLDTREE.SCX

A treeview control displays a collection of Node objects, each of which consists of a label and an optional bitmap. After creating a treeview control, you can add, remove, arrange, and otherwise manipulate Node objects by setting properties and invoking methods.

This sample illustrates adding nodes to a treeview control, deleting nodes, selecting nodes programmatically, writing a treeview hierarchy to a table, and reading a treeview hierarchy from a table.

Adding Nodes

Each node in a treeview control needs to have a unique key that is a character string, generated by the NewKey method in this sample form.

```
*NewKey method
cKey = THIS.cNextKey
THIS.cNextKey = ALLTRIM(STR(VAL(THIS.cNextKey) + 1) + "_")
RETURN cKey
```

The Add method of the treeview control is used to add new nodes. The code associated with the Click event of cmdNewNode adds a root level node using the Add method:

```
o = THISFORM.oleTree
o.Nodes.Add(,1,THISFORM.NewKey(),"Click to edit text",0)
```

The code associated with the Click event of cmdNewChild adds a node as a child of the currently selected node:

```
o = THISFORM.oleTree
IF !ISNULL(o.SelectedItem) THEN
    o.Nodes.Add(o.SelectedItem.Key, 4, THISFORM.NewKey(), "Click to edit
text",0)
ENDIF
```

Deleting Nodes

You can delete all of the nodes by calling the Clear method:

```
THISFORM.oleTree.Nodes.Clear
```

Or you can use the Remove method to delete selected nodes. All children of the deleted node are also deleted.

```
THISFORM.oleTree.Nodes.Remove(THISFORM.oleTree.SelectedItem.Key)
```

Writing And Reading Hierarchies In Tables

To save your treeview hierarchy in a table so you can reload it and edit it, loop through all the nodes in the treeview control and write the Key, the parent node's Key, and the Text to the appropriate fields of a table.

```
FOR i = 1 TO loNodes.Count
    IF ISNULL(loNodes.Item(i).Parent)
        lcParent = "0_" && Root
    ELSE
        lcParent = loNodes.Item(i).Parent.Key
    ENDIF
    INSERT INTO (lcDBFName) VALUES ;
        (loNodes.Item(i).Key, ;
```

```
lcParent, ;  
loNodes.Item(i).Text)  
ENDFOR
```

To reconstruct the treeview hierarchy, scan through the records in the table, using the parent key, key, and text values that you previously stored.

* Fill the TreeView control with values in the table.

```
o = THISFORM.oleTree.Nodes  
SCAN  
  IF ALLTRIM(parent) = '0_'  
    o.add(,1,ALLTRIM(key),ALLTRIM(text),0)  
  ELSE  
    o.add(ALLTRIM(parent),4,ALLTRIM(key), ALLTRIM(text),0)  
  ENDIF  
  THISFORM.cNextKey = ALLTRIM(STR(VAL(key) + 1) + "_")  
ENDSCAN
```

Add Items Interactively to a List Box

File: SAMPLES\SOLUTION\CONTROLS\LISTS\LADD.SCX

This sample demonstrates adding and removing list box items. When a user types text in the text box and presses ENTER, the text in the text box is added to the list and the cursor is returned to the text box so that the user can enter another value.

To allow a user to interactively add items to a list, use the `AddItem` method. In the sample, the following code in the text box `KeyPress` event adds the text in the text box to the list box and clears the text box when the user presses ENTER:

```
PARAMETERS nKeyCode, nShiftCtrlAlt
IF nKeyCode = 13      && Enter Key
    THISFORM.lstAdd.AddItem (This.Value)
    THIS.Value = ""
ENDIF
```

The following code in the `DbClick` event of the list box removes the item that was double-clicked, sending the value to the text box:

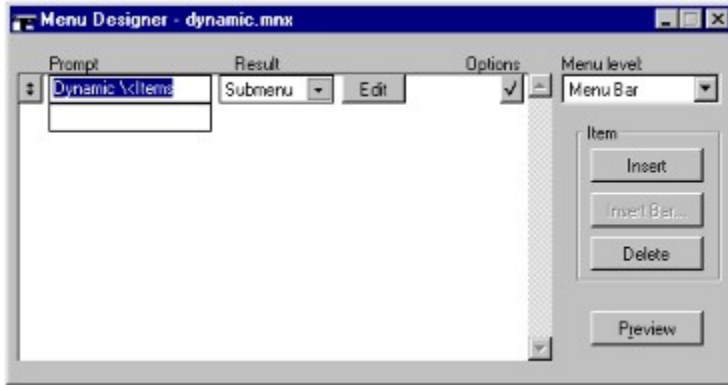
```
THISFORM.txtAddText.Value = This.List(This.ListIndex)
THIS.RemoveItem (This.ListIndex)
```

Add Menu Items at Run Time

File: SAMPLES\SOLUTION\MENUS\FILLMENU.SCX

This sample illustrates adding menu items to a menu at run time.

The menu definition in this sample is defined in the **Menu Designer**, with a single prompt and an empty submenu named `empty_pop`.



Code is included to be executed when a user chooses any item in the menu:

```
PROCEDURE takeaction(cPrompt)
#DEFINE MSG_LOC "You chose " + cPrompt + "."

IF cPrompt = "Release this menu"
    RELEASE PAD dynmenu of _MSYSMENU
ELSE
    WAIT WINDOW MSG_LOC TIMEOUT 1
ENDIF
```

The code associated with the Click event of `cmdRefresh` on the form runs the menu.

```
DO dynamic.mpr
```

Then, for each item in the list, the code defines a menu item with the prompt and message text.

```
FOR i = 1 TO THISFORM.lstMenu.ListCount
    DEFINE BAR i OF empty_pop PROMPT (ALLTRIM(THISFORM.lstMenu.List(i,1))) ;
        MESSAGE (THISFORM.lstMenu.List(i,2))
ENDFOR
```

There is also code to provide the prompt to allow a user to release the menu.

```
DEFINE BAR i + 1 OF empty_pop PROMPT "\-"
DEFINE BAR i + 2 OF empty_pop PROMPT "Release this menu" ;
    MESSAGE "Remove the Dynamic Items menu from the menu bar."
```

Add New Items to a Combo Box

File: SAMPLES\SOLUTION\CONTROLS\COMBOBOX\LOOKUP.SCX

This sample illustrates adding user-entered text to a combo box drop-down and providing lookup capability.

Adding User Text to a Combo Box Drop-Down

A combo box allows a user to enter text or select an item from a drop-down list. You can also add the text a user types in the combo box as a new item in the drop-down list so the user won't have to type the same text multiple times.

There are multiple ways to add user text to the drop-down list of a combo box, depending on the RowSourceType property setting of the combo box. In this sample, the RowSourceType of the combo boxes is 1 - Value. If the value the user enters isn't already in the drop-down list for cboCombo, the following code adds a comma and the new value to the RowSource:

```
cCountryName = ALLTRIM(Custs.Country)
IF ATC(m.cCountryName,THIS.RowSource)=0 AND !EMPTY(m.cCountryName)
    THIS.RowSource=THIS.RowSource+", "+m.cCountryName
ENDIF
```

The combo-box in the [Programmatically Manipulate Text](#) example has a RowSourceType of 0 - None. The following code in the Valid event of cboSearchString in SAMPLES\SOLUTION\CONTROLS\TXT_EDT\TEXT.SCX adds a user-entered value to the drop-down list of the combo box:

```
IF !EMPTY(THIS.Text)
    FOR i = 1 TO THIS.ListCount
        IF THIS.List(i) = THIS.Text
            RETURN
        ENDIF
    ENDFOR
    THIS.AddItem(THIS.Text)
ENDIF
```

If the RowSource of the combo box is an array, you need to add the user text to the array and call the Requery method of the combo box.

If the RowSource is a table or a cursor, you need to add a record, REPLACE the blank field with the user-entered value and Requery the combo box.

Provide Lookup Capability

In addition to illustrating how to add items to a combo box, this sample demonstrates three ways of allowing a user to filter a table for particular values. A user can:

- Enter values in a combo box.
- Select an item from a drop-down list.
- Enter partial values in a combo box for incremental search.

Enter Values in a Combo Box

The key code is associated with the LostFocus event of cboCombo. After making sure that the user has entered a value, the LostFocus code selects the records that match the user's text and sets the RecordSource property of the grid to the result set.

```
cDisplayValue = ALLTRIM(THIS.DisplayValue)

IF THIS.Value = "(All)"
```

```

SELECT country AS location,* FROM CUSTOMER;
    INTO CURSOR Custs
thisform.grdcust.recordsource = "Custs"
ELSE
SELECT country AS location,* FROM CUSTOMER ;
    WHERE UPPER(ALLTRIM(Customer.Country)) = UPPER(m.cDisplayValue);
    INTO CURSOR Custs
    THISFORM.grdCust.RecordSource = "Custs"
ENDIF

```

Select an Item from a Drop-down List

Basically the same lookup code is associated with the InteractiveChange event of cboDrop – if the Value of the combo box is “(All)”, Select all records into the cursor, otherwise, select the records that match the user’s choice.

Enter Partial Values for Incremental Search

Code in the InteractiveChange event of cboIntSearch performs the lookup every time the user types a character in the combo box.

```

#DEFINE DELKEY 127
LPARAMETERS nKeyCode, nShiftAltCtrl
LOCAL cDisplayValue
IF nKeyCode = DELKEY
    cDisplayValue = ALLTRIM(THIS.DisplayValue)
    IF LEN(m.cDisplayValue)=1
        cDisplayValue = ""
    ELSE
        cDisplayValue = LEFT(cDisplayValue,LEN(cDisplayValue)-1)
    ENDIF
ELSE
    cDisplayValue = ALLTRIM(THIS.DisplayValue)+CHR(nKeyCode)
ENDIF

THISFORM.LockScreen = .T.
DO CASE
CASE EMPTY(m.cDisplayValue)
    THISFORM.grdCust.RecordSource = " "
CASE THIS.Value = "(All)"
    SELECT country AS location,* FROM CUSTOMER;
        INTO CURSOR Custs
    THISFORM.grdCust.RecordSource = "Custs"
OTHERWISE
    SELECT country AS location,* FROM CUSTOMER ;
        WHERE UPPER(ALLTRIM(Customer.Country)) = UPPER(m.cDisplayValue);
        INTO CURSOR Custs
    THISFORM.grdCust.RecordSource = "Custs"
ENDCASE
THISFORM.ResetCombos(THIS)
THISFORM.LockScreen = .F.

```

Allow Users to Choose List Values

File: SAMPLES\SOLUTION\FORMS\DATALOOK.SCX

This sample illustrates providing users with a set of values, selected from another table, in a list box and in a drop-down list box. The value the user chooses is stored in the current table.

It is often more convenient for a user to choose from a list of predetermined values, and, of course, you minimize the risk that the user will mistype a value. Setting a few list box properties is all that is required to provide this capability. For example, the following properties were set for cboEmp_id on the **Using Combo Box** page:

Property	Setting
BoundColumn	2
ColumnCount	2
ControlSource	orders.emp_id
RowSource	SELECT DISTINCT ALLTRIM(employee.first_name) + " " + ALLTRIM(employee.last_name) , EMP_ID FROM employee INTO CURSOR cEmpCombo ORDER BY first_name
RowSourceType	3 - SQL Statement

Rather than having the user choose an employee id number from the drop-down list, the SELECT statement allows you to show the user the employees' first and last names.

Because the SELECT statement creates a cursor, code in the Destroy event closes the cursor:

```
IF USED("cEmpCombo") THEN
    USE IN cEmpCombo
ENDIF
```

Allow Users to Drag and Drop Controls

File: SAMPLES\SOLUTION\FORMS\DDROP.SCX

This sample explains how to implement drag and drop operations. One page on the form illustrates manual dragging of controls by calling the Drag method. Explicitly calling the Drag method allows you to process other code in the Click event of the control. The other page illustrates using the Automatic DragMode setting so that the Drag and Drop operation is started automatically when the user presses the mouse button over the controls.

Repositioning the Command Button

In the MouseMove event of cmdDrop, if the left mouse button is down, find out how far the mouse pointer is from the top and the left of the command button and begin the drag operation.

```
LPARAMETERS nButton, nShift, nXCoord, nYCoord
IF nButton = 1 && Left button
    THISFORM.XOffset = nXCoord - THIS.Left
    THISFORM.YOffset = nYCoord - THIS.Top
    THIS.Drag
ENDIF
```

In the DragDrop event of the page that the command button is on, reposition the command button.

```
oSource.Left = nXCoord - THISFORM.XOffset
oSource.Top = nYCoord - THISFORM.Yoffset
```

Changing the Color of the Shape

The DragMode property of the colored squares on the Change Colors page of the form is set to 1 - Automatic. As soon as the user presses the mouse button on the squares, the drag operation begins. Any code added to the Click event of the squares would never be executed.

Code in the DragDrop event of the circle changes its BackColor property to match that of the drag source.

```
LPARAMETERS oSource, nXCoord, nYCoord
THIS.BackColor = oSource.BackColor
```


Automate Microsoft Word and Excel

File: SAMPLES\SOLUTION\OLE\OLEAUT1.SCX

This is a simple sample showing Visual FoxPro as an OLE Automation Controller. It demonstrates how to select data from a Visual FoxPro database, place the data in a Microsoft Excel worksheet, graph the data using Chart, and then display the graph in Word. To use this sample, you need at least Excel 5.0, Word 6.0, and TESTDATA.DBC!customer.dbf, located in the \SAMPLES\DATA directory.

The following line of code creates a reference to a Microsoft Excel spreadsheet.

```
objXLsheet=createobject("Excel.Sheet")
```

The following line of code adds a chart to the spreadsheet.

```
objChart1 = objXLsheet.ChartObjects.Add(100, 100, 200, 200)
```

The following line of code creates a reference to Microsoft Word

```
objWDdoc=createobject("word.basic")
```

```
oWordRef = GetObject('', 'word.basic')
```

Automate a Microsoft Excel Spreadsheet

File: SAMPLES\SOLUTION\OLE\OLEXL.SCX

This form shows two samples of Visual FoxPro calling an Excel worksheet through automation.

The **Trend** button performs a query followed by a cross-tab to create a dataset to pass to Excel through automation using the Excel.Sheet object. Once in Excel, Visual FoxPro can call a Trend function to perform a simple regression analysis.

The **Chart** button automates the Excel.Chart object to create a chart. Excel supports passing data through automation, but MS Graph does not. However, you can distribute MS Graph with your applications.

Automate a Microsoft Word Document in a Form

File: SAMPLES\SOLUTION\OLE\OLEWORD.SCX

This sample shows how you can automate an embedded Word object on a form, insert some rich text, and then format the text. Typical automation with Word is usually done via the “Word.Basic” object used interactively with a CREATEOBJECT() function.

```
oForm = THISFORM
oForm.AddObject('oWordDoc','OleControl','WordDocument')
oForm.oWordDoc.Visible = .t.
oForm.oWordDoc.DoVerb(0)
```

Change Font Attributes

File: SAMPLES\SOLUTION\TOOLBARS\FORMAT.SCX

This example illustrates using a toolbar to set `FontName`, `FontSize`, `FontBold`, `FontItalic`, `ForeColor`, and `BackColor` properties of controls on a form.

The toolbar is the `tbrEditing` class in `SAMPLES\CLASSES\SAMPLES.VCX`. The `nAppliesTo` property of the class specifies whether to set the properties of the currently selected control, all text boxes and edit boxes on the form, or all controls on the form. The code in the `InteractiveChange` or `Click` event of the toolbar controls sets the properties. For example, the following code is associated with the `Click` event of `cmdBold`:

```
IF TYPE("_SCREEN.ActiveForm") = 'O'
    oForm = _SCREEN.ActiveForm
ELSE
    RETURN
ENDIF

DO CASE
    CASE THIS.Parent.nAppliesTo = 1 && Current Control
        oForm.ActiveControl.FontBold = THIS.Value

    CASE THIS.Parent.nAppliesTo = 2 && Text and edit boxes
        oForm.SetAll('FontBold', THIS.Value, 'TEXTBOX')
        oForm.SetAll('FontBold', THIS.Value, 'EDITBOX')

    CASE THIS.Parent.nAppliesTo = 3 && All Controls
        oForm.SetAll('FontBold', THIS.Value)
ENDCASE
```

The only code in the form is associated with the `GotFocus` event of the controls. Each `GotFocus` event contains the following line of code:

```
THISFORMSET.tbrEditing.Refresh(THIS)
```

The `Refresh` method of `tbrEditing` sets the values of the editing controls of the toolbar to reflect the current settings of the object whose reference is passed in as a parameter.

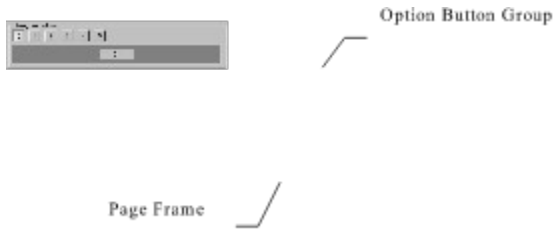
Change Pages When a User Chooses a Button

File: SAMPLES\SOLUTION\CONTROLS\PAGEFRAME\MSGBOX

This sample allows you to interactively construct a message box.

There are six command button combinations possible in a message box. Each of these combinations is displayed on a separate page in a page frame on the MessageBox Builder form.

The option buttons, labeled 0 through 5, allow you to specify which combination of command buttons you want to be displayed in your message box. The selected button in the page indicates the default MessageBox button selection. The ActivePage property of the page frame is set to the Value of the Option group.



Constructing the MESSAGEBOX Code

In the InteractiveChange event of the controls on the form, the WriteCode method of the form is called to dynamically construct the MESSAGEBOX code.

```
* WriteCode Method
#DEFINE QM '''

cString = '=MESSAGEBOX('
cString = cString + QM + ALLTRIM(STRTRAN(THIS.edtMessage.Value,
CHR(13)+CHR(10), QM + '+CHR(13)+' + QM)) + QM + ', '
cString = cString + THIS.cboIcon.Value + '+'
cString = cString + THIS.DefaultButton + '+'
cString = cString + ALLTRIM(STR(THIS.opgButtons.Value - 1)) + ', '
cString = cString + QM + ALLTRIM(THIS.txtCaption.Value) + QM + ') '

THIS.edtCode.Value = cString
```

Change the Number of Tabs at Run Time

File: SAMPLES\SOLUTION\PGFRAME\PFSSAM1.SCX

This sample illustrates using a page frame with tabs. The number of tabs changes dynamically when a user chooses a new value in a spinner. When the user selects a tab, a list box displays values specific to the tab.

- ▶ **To change the number of tabs at run time**
 - Set the PageCount property to the desired number of pages.

The list box in this sample is positioned on the form, not on any of the pages in the page frame. This placement allows you to have a single control visible in all of your pages.

Two user-defined form methods set the captions on the tabs and make sure the values in the list box match: UpdateList and SetCaption.

UpdateList Method

UpdateList is called when the number of tabs changes and when the user selects a new tab.

```
LOCAL lnPage, lcHigh, lcLow
#define NO_MATCH_LOC 'No Matching Names for '
DIMENSION THISFORM.aCustomers[1,2]
lnPage = THISFORM.pgfRolodex.activepage
THISFORM.aCustomers[1,1] = NO_MATCH_LOC + ;
    THISFORM.pgfRolodex.Pages(lnPage).Caption
THISFORM.aCustomers[1,2] = ""

lcHigh = substr(THISFORM.pgfrolodex.Pages(lnPage).caption, 3, 1)
lcLow = substr(THISFORM.pgfrolodex.Pages(lnPage).caption, 1, 1)
SELECT company, phone FROM customer;
    WHERE company <= lcHigh and company => lcLow;
    ORDER BY company;
    INTO ARRAY THISFORM.aCustomers

THISFORM.lstCustomers.Requery
THISFORM.lstCustomers.Value = 1
```

SetCaption Method

SetCaption is called in the spinner's UpClick and DownClick events to calculate what captions should appear on the tabs.

```
THISFORM.LockScreen = .T.
FOR n = 1 to THISFORM.pgfRolodex.PageCount
    FirstLetter = SUBSTR(THISFORM.Alphabet, ((n -
1)*ROUND(LEN(THISFORM.Alphabet)/THISFORM.pgfRolodex.PageCount, 0))+1, 1)
    IF n = THISFORM.pgfRolodex.PageCount &&last page
        LastLetter = right(THISFORM.Alphabet, 1)
    ELSE
        LastLetter = SUBSTR(THISFORM.Alphabet,
((n)*ROUND(LEN(THISFORM.Alphabet)/THISFORM.pgfRolodex.PageCount, 0)), 1)
    ENDIF
    THISFORM.pgfRolodex.Pages(n).Caption = FirstLetter + "-" + LastLetter
ENDFOR
THISFORM.LockScreen = .F.
```

Alphabet is a user-defined form property that is used to determine the captions for each tab To get the same functionality in different languages, set the Alphabet property to a string containing all the letters

in the target language alphabet.

Coordinate Menu Items and Toolbar Buttons

File: SAMPLES\SOLUTION\MENUS\TOOLMENU.SCX

This sample illustrates coordinating menu items and toolbar buttons that provide the same functionality.

Sometimes you'll want to make particular functionality accessible from both menu items and toolbar buttons. For sample, in Visual FoxPro you can save a file by choosing the **Save** toolbar button or by choosing **Save** from the **File** menu.

There are three components to this sample:

Component	Description
TOOLMENU.SCX	the form
tbrBackColor in SOLUTION.VCX	the toolbar
TOOLMENU.MNX	the menu

In the Init event of the form, a toolbar object is created whose object reference is a property on the form, oToolbar.

```
SET CLASSLIB TO ..\solution
This.oToolbar = CREATEOBJECT('tbrbackcolor')
```

```
* Position the toolbar and show it
THIS.oToolbar.Left = THIS.Left + 10
THIS.oToolbar.Top = THIS.Top - 50
THIS.oToolbar.Visible = .T.
```

```
* Push the current menu on the stack so it can be
* restored in the Destroy event of the form.
PUSH MENU _MSYSMENU
```

```
* Run the menu
DO toolmenu.mpr
```

Rather than coding duplicate functionality in the toolbar and the menu, updating it both places if changes are required, the code associated with the menu items calls the code associated with the toolbar buttons. For example, the following command is associated with the first menu item:

```
_VFP.ActiveForm.oToolbar.cmdRed.click
```

The SKIP FOR clause of the menu items causes them to be disabled when the associated toolbar button is disabled. For example, the following expression is associated with the SKIP FOR clause of the first menu item:

```
!_VFP.ActiveForm.oToolbar.cmdRed.Enabled
```


Create a 1-To-Many Data Entry Form

File: SAMPLES\SOLUTION\FORMS\MANY.SCX

This sample illustrates the basic tasks associated with a one-to-many data entry form.

When you choose the **New** button beside the parent table text boxes, the following code is executed:

```
SELECT CUSTOMER  
APPEND BLANK  
THISFORM.Refresh
```

When you choose the **New** button beside the grid that displays the child records, more code is executed.

After selecting the orders table, the code calculates a new id value for the order, adds the blank record, then stores the parent id and order id to the appropriate fields.

```
CALCULATE MAX(order_id) ALL TO lMaxID  
lMaxID = ALLTRIM(STR(VAL(lMaxID) + 1))
```

```
APPEND BLANK
```

```
REPLACE cust_id WITH Customer.Cust_id IN orders  
REPLACE order_id with lMaxID in orders
```

Note This method of creating a new id value would not be reliable if multiple users were adding new orders at the same time. Instead, you can create a new id as illustrated in the [Create a Default Unique ID Value for a Field](#) sample.

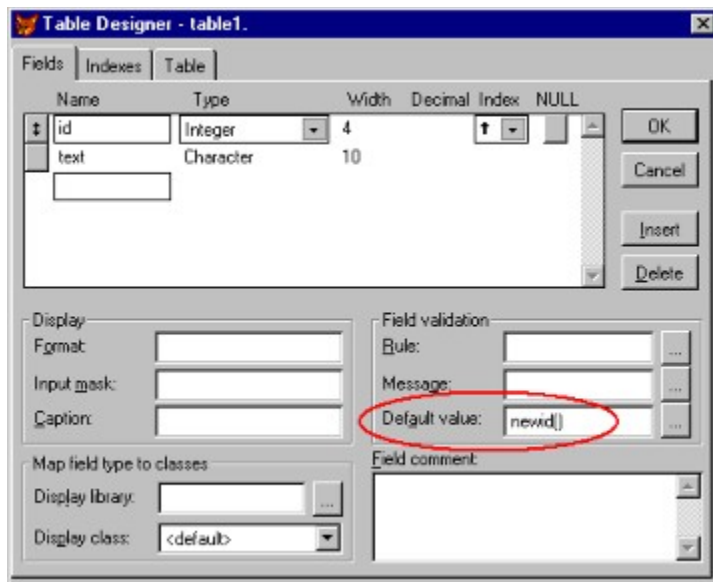
Create a Default Unique ID Value for a Field

File: SAMPLES\SOLUTION\ADB\NEWID.SCX

This sample uses a stored procedure in a database to provide a default primary key value.

► To create a default unique ID

- 1 Add a separate table to the database that stores the next ID for each table in the database.
The name of this table in the sample is IDS.DBF. It contains two fields: Table C(10), NextID I.
- 2 Create a function in the stored procedures of the database to return the next id value from the id table.
The name of this function in the sample is NewID. It is listed below.
- 3 Set the default value for the field to the function.



NewID Stored Procedure

```
FUNCTION NewID(tcAlias)
LOCAL lcAlias, lnID, lcOldReprocess, lnOldArea

lnOldArea = SELECT()

IF PARAMETERS() < 1
    lcAlias = UPPER(ALIAS())
ELSE
    lcAlias = UPPER(tcAlias)
ENDIF

lcOldReprocess = SET('REPROCESS')

* Lock until user presses Esc
SET REPROCESS TO AUTOMATIC

IF !USED("IDS")
    USE newid!ids IN 0
ENDIF
```

```
SELECT ids

IF SEEK(lcAlias, "Ids", "table")
  IF RLOCK()
    lnID = ids.nextid
    REPLACE ids.nextid WITH ids.nextid + 1
    UNLOCK
  ENDIF
ENDIF

SELECT (lnOldArea)
SET REPROCESS TO lcOldReprocess

RETURN lnID
ENDFUNC
```

Create a Query-By-Example Form

File: SAMPLES\SOLUTION\FORMS\QBF.SCX

This sample enables users to search and filter records in the same interface that they view the records.

When a user chooses **Enter QBF**, the code associated with the Click event of cmdQBFForm begins a transaction and appends a blank record to the table to store the user's query text.

```
* extract from cmdQBFForm.Click
BEGIN TRANSACTION
APPEND BLANK
THISFORM.Refresh
```

When the user chooses **Query**, code associated with the Click event of cmdExecuteQBF rolls back the transaction, throwing away the new record.

You can use transactions only with tables contained in a database. If you want to include QBF capabilities for a table that is not in a database, you can loop through all the controls on the form, save the old ControlSource property setting and set the ControlSource property of the controls to an empty string. After getting the user query string, you can loop back through the controls and reset the ControlSource properties.

Code associated with the Click event of cmdExecuteQBF also loops through the controls on the form, checks for user-entered values, and calls the ParseCondition method to assemble the filter string.

A user can enter a single value to match or an expression. For example, a user can enter either of the values below in the Title text box to set a filter:

Sales Manager

!= "Sales Manager"

If the user doesn't enter an expression, quotation marks are not required.

The ParseCondition Method

```
LPARAMETERS cCondition, cControlSource
LOCAL lcRetCondition, lcFieldName
IF TYPE('cCondition') = 'C'
    cCondition = ALLTRIM(cCondition)
ENDIF

lcFieldName = SUBSTRC(cControlSource, (RATC(".", cControlSource)+1))

IF !EMPTY(cCondition) THEN
    IF TYPE('cCondition')$ "CM"
        IF ("<" $ cCondition OR ;
            "==" $ cCondition OR ;
            "LIKE" $ cCondition OR ;
            "<>" $ cCondition OR ;
            "!=" $ cCondition OR ;
            "#" $ cCondition OR ;
            "=" $ cCondition OR ;
            ">" $ cCondition)
            lcRetCondition = lcFieldName + cCondition
        ENDIF
    ENDIF
ENDIF
```

```
IF EMPTY(lcRetCondition)
  DO CASE
    * put quotes around character expressions
    CASE TYPE(cControlSource) $ "CM"
      lcRetCondition = lcFieldName + " = " + CHR(34) + cCondition +
CHR(34)

    * put braces around date expressions
    CASE TYPE(cControlSource) $ "DT"
      lcRetCondition = lcFieldName + " = {" + DTOC(cCondition) + "}"

    OTHERWISE
      lcRetCondition = lcFieldName + " = " + STR(cCondition)
    ENDCASE
  ENDIF
ELSE
  lcRetCondition = ""
ENDIF

RETURN lcRetCondition
```

Create a Single Table Data Entry Form

File: SAMPLES\SOLUTION\FORMS\SINGLE.SCX

This sample illustrates the simplest scenario for a single-user, single-table data entry form.

The ControlSource property of each of the text boxes and combo boxes on the form is set to a field in the Customer table.

An APPEND BLANK command is issued in the Click event of cmdNew.

```
APPEND BLANK  
THISFORM.Refresh
```

The DELETE command is issued in the Click event of cmdDelete.

```
* cmdDelete.Click  
#DEFINE MSGBOX_YES 6  
#DEFINE C_MSGBOX1 36  
#DEFINE C_DELETE_LOC "Are you sure you want to delete this record?"  
  
IF MESSAGEBOX(C_DELETE_LOC,C_MSGBOX1) = MSGBOX_YES  
    DELETE  
    IF !EOF()  
        SKIP 1  
    ENDIF  
    IF EOF() AND !BOF()  
        SKIP -1  
    ENDIF  
    THISFORM.Refresh  
ENDIF
```

Extending the Data Entry Form

To create a more robust data entry form, one that allows a user to cancel changes or allows multiple users to access the same data, you need to use transactions and table or row buffering. For more information on these features, see Chapter 17, "Programming for Shared Access," in the Visual FoxPro *Developer's Guide*.

Create an SDI Form

File: SAMPLES\SOLUTION\FORMS\SDIFORM.SCX

This sample illustrates Visual FoxPro support of SDI forms. In order to create an SDI form, all you have to do is set the form's ShowWindow property to 2 - As Top-Level Form. SDI forms do not require the Visual FoxPro window to be visible.

Adding Forms

There are two ways to add forms to a Top-Level (SDI) form. The first and easiest is to create a new form and set its ShowWindow property to 1 (In Top-Level form). While this may seem the most appropriate, it is not always the best solution because the ShowWindow property can only be set at design time.

Another method is to use the ACTIVATE WINDOW command. The code below from the SDIForm sample shows just this. Since this command is not object-based, you must reference a form by its window name (Name property).

```
ACTIVATE WINDOW (thisform.oWindows[m.nGetWin].NAME) ;  
IN WINDOW (thisform.name)
```

Adding Menus

Menus can also be added to SDI forms. There is now a new Top-Level Form check box in the **Menu Designer's View** menu **General Options** dialog box. A **Top-Level Form** menu will be generated if you check this option and can be called as shown here.

```
DO sdiform.mpr WITH THISFORM, .T.
```

Note If system menu items rely on the main Visual FoxPro window being visible, they won't give the expected results when the main Visual FoxPro window is hidden.

Adding Toolbars

In addition to menus and windows, you can also add toolbars to your SDI forms. Toolbars are objects like forms, so you can take advantage of the Visual FoxPro object model and scope that toolbar to the form. The SDI form sample uses a custom property to create and keep the toolbar in scope.

```
SET CLASSLIB TO sditbar ADDITIVE  
thisform.oToolbar=create("sditb1")  
thisform.oToolbar.show
```

Create Dynamic Shortcut Menus

File: SAMPLES\SOLUTION\MENUS\DYNSHORT.SCX

This sample illustrates an alternative way to create shortcut menus.

The shortcut menus in the [Display Shortcut Menus](#) sample were created in the Menu Designer. The advantages of using the Menu Designer are ease of design, the ability to create cascading menus, and simple integration on a form. With the Menu Designer, however, you need a separate .MNX, .MNT, and .MPR file for each shortcut menu. If you make a change, you need to generate and compile the menu code again.

The dynamic shortcut menu engine is a custom class that you can add to any form: menuLib in SAMPLES\CLASSES\UTILITY.VCX. The ShowMenu method of this class defines a menu and displays it at the MousePointer location as determined by the MROW() and MCOL() functions.

In the RightClick event of the objects you want to create a shortcut menu for:

1. Create an array with the shortcut menu items.
2. Pass the array to the ShowMenu method of the menuLib class.
3. Process the user choice by checking the BAR() value.

For example, the following code is associated with the RightClick event of the form:

```
LOCAL laMenu[5]

laMenu=""
laMenu[1]="\<Center"
laMenu[2]="\<Font..."
laMenu[3]="\<Minimize"
laMenu[4]="\-"
laMenu[5]="E\<xit"
THISFORM.oMenuShortcut.ShowMenu(@laMenu)
DO CASE
    CASE BAR()=1
        THISFORM.AutoCenter=.T.
    CASE BAR()=2
        THISFORM.SetFont
    CASE BAR()=3
        THISFORM.WindowState=1
    CASE BAR()=5
        THISFORM.Release
ENDCASE
```


Customize the Open Dialog Box

File: SAMPLES\SOLUTION\OLE\COMMDLOG.SCX

This sample illustrates customizing the appearance and behavior of the **Open** dialog box as exposed through the Common Dialog control.

You can set the Flags property of the Common Dialog control to specify the behavior of the **Open** dialog box. For a list of the various Flags values, see [Open Dialog Box Flag Values](#).

The following code in the Click event of cmdFiles checks the values of the various check boxes on the form and sets the Flags property of the Common Dialog control.

Set the Read-only checkbox flag

```
IF !thisform.chkRead.Value
    m.nFlags = m.nFlags + 4
ENDIF
```

Set the Multiple files flag

```
IF thisform.chkMulti.Value
    m.nFlags = m.nFlags + 512
ENDIF
```

Set the Help button flag

```
IF thisform.chkHelp.Value
    m.nFlags = m.nFlags + 16
ENDIF
```

Set the enforce file existence flag

```
IF thisform.chkMulti.Value
    m.nFlags = m.nFlags + 4096
ENDIF
```

Set the Flags property of the Common Dialog control

```
THISFORM.oleCommDlog.Flags = m.nFlags
```

Set the Filter property of the Common Dialog control

You can use the Filter property of the Common Dialogs control to specify what files a user is allowed to choose.

```
THISFORM.oleCommDlog.Filter = "All files" + ;
    " (*.*)|*.*|Text (*.txt)|*.txt" + ;
    "|Pictures (*.bmp;*.*ico)|*.bmp;*.*ico"
```

Open Dialog Box Flag Values

You can customize the **Open** dialog box by setting the Flag property to the sum of a selection of these values:

Value	Description
1	Causes the Read Only check box to be initially checked when the dialog box is created. This flag also indicates the state of the Read Only check box when the dialog box is closed.
2	Causes the Save As dialog box to generate a message box if the selected file already exists. The user must confirm whether to overwrite the file.
4	Hides the Read Only check box.
8	Forces the dialog box to set the current directory to what it was when the dialog box was opened.
16	Causes the dialog box to display the Help button.
256	Specifies that the common dialog box allows invalid characters in the returned filename.
512	Specifies that the File Name list box allows multiple selections. The user can select more than one file at run time by pressing the SHIFT key and using the UP ARROW and DOWN ARROW keys to select the desired files. When this is done, the FileName property returns a string containing the names of all selected files. The names in the string are delimited by spaces.
1024	Indicates that the extension of the returned filename is different from the extension specified by the DefaultExt property. This flag isn't set if the DefaultExt property is Null, if the extensions match, or if the file has no extension. This flag value can be checked upon closing the dialog box.
2048	Specifies that the user can enter only valid paths. If this flag is set and the user enters an invalid path, a warning message is displayed.
4096	Specifies that the user can enter only names of existing files in the File Name text box. If this flag is set and the user enters an invalid filename, a warning is displayed. This flag automatically sets the 2048 flag.
8192	Specifies that the dialog box prompts the user to create a file that doesn't currently exist. This flag automatically sets the 4096 and 2048 flags.
16384	Specifies that sharing violation errors will be ignored.
32768	Specifies using the Windows 95 explorer-like Open A File dialog box template. (Windows 95 only.)
524288	Use the Explorer-like Open A File dialog box template. Common dialogs that use this flag do not work under Windows NT using the Windows 95 shell.
1048576	Do not dereference shell links (also known as shortcuts). By default, choosing a shell link causes it to be dereferenced by the shell.

- 1048576 Do not dereference shortcuts (shell links). By default, choosing a shortcut causes it to be dereferenced by the shell. (Windows 95 only.)
- 2097152 Allows the user to use long filenames. (Windows 95 only.)

Disable or Display a Check Beside a Menu Item

File: SAMPLES\SOLUTION\MENUS\CHKMENU.SCX

This sample illustrates dynamically disabling menu items and displaying check marks beside specific menu items.

The menu in this sample was created in the **Menu Designer** and is run when the form is initialized.

```
DO chkmenu.mpr
```

To disable a menu item, include an expression that evaluates to false (.F.) in the SET SKIP OF command. For example, the following line of code in the InteractiveChange event of a check box issues SET SKIP OF with the inverse of the value of the check box, disabling the menu item when the check box is not selected.

```
SET SKIP OF BAR 1 OF checkitems !THIS.Value
```

To display a check mark beside a menu item, include an expression that evaluates to true (.T.) after the TO keyword of the SET MARK OF command. For example, the following line of code in the InteractiveChange event of a check box issues the SET MARK OF command with the Value property setting of the check box.

```
SET MARK OF BAR 1 OF checkitems TO THIS.Value
```

Display a Stop Watch

File: SAMPLES\SOLUTION\CONTROLS\TIMER\SWATCH.SCX

This sample includes the Stopwatch class (described in Chapter 3, “Object-Oriented Programming,” in the *Developer’s Guide*).

The Stopwatch Class

The Stopwatch class includes a timer and several display labels. The Timer increments numeric custom properties of the class and sets the Caption property of the labels accordingly.

The Start method of the class sets the Enabled property of the Timer to true (.T.). The Stop method sets the Enabled property of the Timer to false (.F.). And the Reset method sets the numeric properties to 0.

The Swatch Form

The Swatch form contains an object derived from the Stopwatch class, along with some command buttons. The code written for the Click event of the first command button on the form calls the Start and Stop methods of the Stopwatch class. The second button calls the Reset method of the Stopwatch class.

Display a System Clock

File: SAMPLES\SOLUTION\CONTROLS\TIMER\CLOCK.SCX

This sample contains the Clock custom control in SAMPLES.VCX on a form. The class includes a text box for displaying the date and time and a timer for updating the display.

The TimeFormat property of the clock class can be set to 0 for 24-hour time format or 1 for 12-hour format. The code in the Timer event of the timer updates the time:

```
#DEFINE LONGDATE_LOC CDATE() + " " + CMONTH(CDATE()) + " " + ;
    ALLTRIM(STR(DAY(CDATE()))) + ", " + ALLTRIM(STR(YEAR(CDATE())))

IF This.Parent.TimeFormat = 0
    This.Parent.txtTime.Value = IIF(VAL(SUBSTR(TIME(), 1, 2)) > 12, ;
        ALLTRIM(STR((VAL(SUBSTR(TIME(), 1, 2)) - 12))) + SUBSTR(TIME(), 3, 6), TIME())
ELSE
    This.Parent.txtTime.Value = TIME()
ENDIF

THIS.Parent.txtDate.Value = LONGDATE_LOC
```

This class contains a separate text box for date and time values. If you want to hide one or the other, you can easily set its Visible property to false (.F.). An easier way to display the time and date is to use a single text box and set the value of this text box in a timer control to DATETIME(). This will allow you to take advantage of the Hours, Seconds, and DateFormat properties of a text box.

Display Calculated Values in a Column

File: SAMPLES\SOLUTION\CONTROLS\GRID\CALC.SCX

This sample illustrates how to display a calculated value in a column.

Set the ControlSource property of the column to an expression with a calculation. For example, the following expression for the ControlSource of the Profit column displays the difference between the price and cost.

```
Products.Unit_Price - Products.Unit_Cost
```

When you change one or more values in the expression, the value in the column is automatically updated.

Note Columns that display an expression containing a calculation are read-only.

Display Child Records from a Relationship

File: SAMPLES\SOLUTION\CONTROLS\GRID\1_MANY.SCX

This sample demonstrates coordinating a one-to-many-to-many form with two grids displaying the many sides of the relationships. Very little code is needed in this sample. Setting a few properties is all that is involved.

Property settings

Text boxes for the "one" side of the relationship:

```
ControlSource = Customer.cust_id  
ControlSource = Customer.company
```

Grid for the first "many" side of the relationship:

```
ChildOrder = cust_id  
LinkMaster = customer  
RecordSource = orders  
RecordSourceType = 1 - Alias  
RelationalExpr = customer.cust_id  
ColumnCount = 4  
    Column1.ControlSource = "orders.order_no"  
    Column1.Header1.Caption = "Order"  
    Column2.ControlSource = "orders.order_date"  
    Column2.Header1.Caption = "Date"  
    Column3.ControlSource = "orders.to_name"  
    Column3.Header1.Caption = "Ship To"  
    Column4.ControlSource = "orders.order_amt"  
    Column4.Sparse = .F.  
    Column4.Header1.Caption = "Total"  
    Column4.Text1.InputMask = "$999,999.99"
```

Grid for the 2nd 'many' side of the relationship

```
ChildOrder = order_id  
LinkMaster = Orders  
RecordSource = Orditems  
RecordSourceType = 1 - Alias  
RelationalExpr =Orders.order_id  
ColumnCount = 4  
    Column1.ControlSource = "orditems.line_no"  
    Column1.Header1.Caption = "Item"  
    Column2.ControlSource = "products.prod_name"  
    Column2.Header1.Caption = "Product"  
    Column3.ControlSource = "orditems.quantity"  
    Column3.Header1.Caption = "Qty."  
    Column4.ControlSource = "orditems.unit_price"  
    Column4.Sparse = .F.  
    Column4.Header1.Caption = "Price"  
    Column4.Text1.InputMask = "$9,999.99"
```

In addition to the text boxes and the grids, the form contains an object based on the VCR class in BUTTONS.VCX. The SkipTable property of this class is set to the table you want the record pointer to be moved in when the user chooses a table navigation button. Because the user can easily manually move through the records in the grids (Orders and Orditems), SkipTable is set to Customer.

Display Controls in a Grid

File: SAMPLES\SOLUTION\CONTROLS\GRID\CONTROLS.SCX

This sample demonstrated controls displayed in grid columns.

You can add controls to a column in the **Form Designer** by selecting a column and adding a control or programmatically with the `AddObject` method.

There are a few properties that are important for displaying controls in a column:

CurrentControl Property

After you add a control to a grid column, you need to set the column's `CurrentControl` property to the new control for the new control to be displayed.

Sparse Property

The check boxes on the form allow you to toggle the `Sparse` property of the grid columns that contain controls. When `Sparse` is set to `.T.`, the control is only displayed when the focus is on a cell in the column. When `Sparse` is set to `.F.`, the control is always displayed in each cell in the column.

Controls and Events

Notice that the control in the column processes the events when you are set the focus to a cell in the column. For example, if you select a cell in a column with a spinner, when you press the up and down arrows, you increment or decrement the value in the spinner. This is the default behavior for a spinner, not the default behavior for a text box in a grid cell.

For a more detailed description of adding controls to grid columns, see Chapter 10, "Using Controls," in the *Developer's Guide*.

Display Different Pages Without Tabs

File: SAMPLES\SOLUTION\CONTROLS\PGFRAME\PFSSAM2.SCX

This sample demonstrates manipulating pages without tabs in a form page. The frame in the middle of the form contains three pages that are brought to the front as the user chooses command buttons. Each page can have its own controls and appearance.

The method that brings a particular page to the front is `ZOrder`. For example, the following line of code in the click event of one of the command buttons brings `pagCustomers` to the front:

```
THISFORM.pgfPeople.pagCustomers.ZOrder
```

You can also use the `ActivePage` property of the page frame to determine which pages are displayed:

```
THISFORM.pgfPeople.ActivePage = 1
```

Display Line Animation on a Form

File: SAMPLES\SOLUTION\GRAPHICS\ANIM.SCX

This sample illustrates drawing lines on a form. More specifically, it demonstrates saving coordinates of sets of lines drawn on a form and redrawing them, along with additional lines at intermediate positions, giving the illusion of motion.

Adding lines to the table.

Each time a user draws a line on the form, its coordinates are stored in a table with the following structure:

<u>Name</u>	<u>Type</u>	<u>Description</u>
FrameNo	I	Incremented each time the user chooses New Frame .
Objno	I	Incremented each time a line is added to a frame.
X1	I	The starting X coordinate for a line.
X2	I	The ending X coordinate for a line.
Y1	I	The starting Y coordinate for a line
Y2	I	The ending Y coordinate for a line.

Playing the frames.

The code that plays the frames, uses the table again in another work area, selects the second work area, and goes to the next frame:

```
USE (lcTable) AGAIN IN 0 ALIAS shadow
SELECT shadow
LOCATE FOR frameno # &lcTable..frameno
```

The variable *nBetween* determines how many intermediate lines are drawn on the form between a line in one frame and the corresponding line in the next frame.

```
FOR nb = 1 TO nBetween
```

Inside the FOR loop, the code scans for all of the lines associated with a frame and calculates coordinates for the intermediate lines, for example:

```
nx1 = frames.x1 + nb * (shadow.x1 - frames.x1) / nBetween
ny1 = frames.y1 + nb * (shadow.y1 - frames.y1) / nBetween
```

The code then prints each intermediate line, and after a WAIT of .05 seconds clears the form and continues the loop.

```
THISFORMSET.frmAnimation.line(nx1,ny1,nx2,ny2)
```

Display Multiple Columns in a List Box

File: SAMPLES\SOLUTION\CONTROLS\LISTS\LMULCOL.SCX

This sample demonstrates displaying multiple columns in a list box. The form displayed contains a list box and a spinner. The number of columns in the list box is set to the Value property of the spinner: 1, 2, 3, or 4.

To set the width of the columns in a multi-column list box, use the ColumnWidths property. For example, if there are 3 columns in the list box, the following command will set the column widths to 10, 15, and 20, respectively:

```
Form.List.ColumnWidths = "10, 15, 20"
```

To set the fields to be displayed in the columns, set the RowSource property. For example, the following command sets the sources of three columns in a 3-column list box to the contact, city, and country fields of the customer table.

```
Form.List.RowSource = "contact,city,country"
```

The ColumnLines property determines whether lines are displayed between the columns in the list.

Display Pictures in a List

File: SAMPLES\SOLUTION\LISTS\PICLIST.SCX

This sample illustrates how you can spice up your list boxes by adding picture graphics beside the individual text items. In this sample, you can pick any database (.DBC) file, and the list box will show different images beside the names of tables, local views and remote views.

The following code from cmdDatabase.Click enumerates through a .DBC and selectively adds pictures to list items using the Picture property. The Picture property relies on an index setting to determine which list item to affect.

```
FOR i = (m.nTblCount+1) TO thisform.lstDatabase.ListCount
IF DBGETPROP(ALLTRIM(thisform.lstDatabase.List[m.i]),; "view","sourcetype")
= 1
    *Local view
    thisform.lstDatabase.Picture[m.i] = m.cLViewBMP
ELSE
    * Remote view
    thisform.lstDatabase.Picture[m.i] = m.cRViewBMP
ENDIF
ENDFOR
```

Note You need to ensure that the images used are sized correctly to fit within the space of a single list box item, which differs with the FontSize of the list.

Display Pictures in an Image Control

File: SAMPLES\SOLUTION\FORMS\IMAGE.SCX

This sample illustrates displaying bitmaps or icons in an image control.

The list box on the form displays files. In the InteractiveChange event of the list box, the Picture property of the image control is set to the selected file, if the file is a .BMP or .ICO.

```
* lstFiles.InteractiveChange
cSelected = UPPER(THIS.List(THIS.ListIndex))
CD THIS.List(2)
IF ".BMP"$cSelected OR ".ICO"$cSelected
    THISFORM.imgDisplay.Picture = THIS.List(2) + cSelected
ENDIF
```

Display Shortcut Menu

File: SAMPLES\SOLUTION\MENUS\SHORTCUT.SCX

This sample illustrates displaying shortcut menus when a user “right-clicks” an object. When you have created a shortcut menu in the **Menu Designer**, the code in the form to activate the menu is very simple. The following code is associated with the RightClick event of the form:

```
DO frmshort.mpr WITH THIS
```

The following code is associated with the RightClick event of the edit boxes on the form:

```
DO edtshort.mpr WITH THIS
```

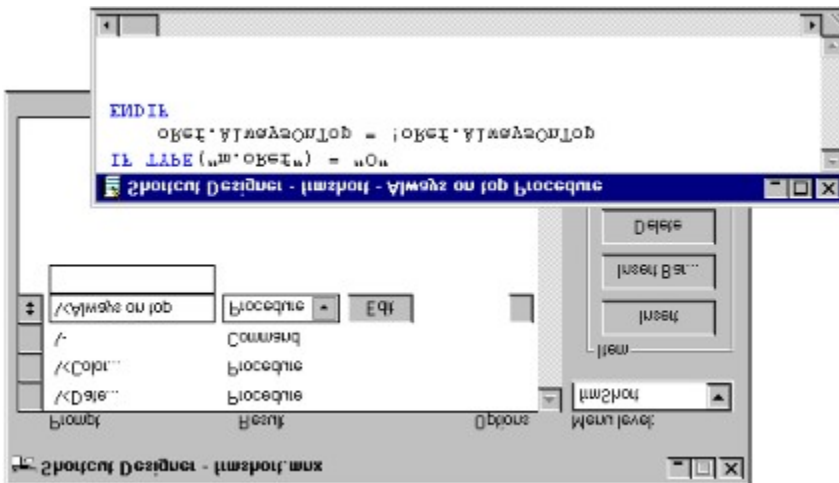
The FRMSHORT Menu

The FRMSHORT menu was designed to take an object parameter. The following code is included in the menu’s setup code:

```
PARAMETER oREF  
#PREPOP
```

The #PREPOP generative directive causes the code in the menu’s cleanup to be generated before the ACTIVATE POPUP command. This allows you to disable or enable menu items at run time, as well as displaying check marks beside menu items.

The code associated with the menu items, sets properties of the object that was passed to the .MPR as a parameter. For example, the **Always on top** item sets the AlwaysOnTop property of the form.



The following code, included in the menu’s cleanup code, displays a check mark beside the **Always on top** item when the form’s AlwaysOnTop property is set to true (.T.).

```
SET MARK OF BAR 4 OF frmshort TO oRef.AlwaysOnTop
```

The EDTSHORT Menu

The EDTSHORT menu also accepts an object parameter in its setup code. Because there is no need to display a check mark beside the items in this menu, the #PREPOP generator directive is not used.

For an alternative way to display shortcut menus, see the [Create Dynamic Shortcut Menus](#) sample.

Display System Information

File: SAMPLES\SOLUTION\OLE\SYSINFO.SCX

This sample illustrates using the SysInfoControl to display system information and to notify when a system setting changes.

The bulk of the code in this sample is in the CheckStatus method. Code in this method checks the settings of various SysInfoControl properties to see their current settings, and populates a treeview control with this information. For example, the following section of code checks the BatteryLifePercent setting:

```
IF ThisForm.SysInfo.BatteryLifePercent = 255
    * Add a node to display the information
ENDIF
```

When a system setting changes, an event of the SysInfoControl occurs. Code associated with each of these events sets the caption of a label and calls the CheckStatus method to refresh the treeview control. For example, the following code is associated with the SysColorsChanged event:

```
ThisForm.Status.Caption = SysColorsChanged_LOC
ThisForm.CheckStatus
```


Draw Lines and Shapes on a Form

File: SAMPLES\SOLUTION\FORMS\GRAPHICS\ANIM.SCX

This sample illustrates how to use form graphics methods (PSet, Line, and Circle) and properties (DrawMode, DrawStyle, DrawWidth) to allow a user to draw figures and shapes on a form in different colors and different pen widths.

The sample includes a toolbar (defined in FDPROC.PRG) and a form (based on frmFD in FD.VCX). The user can select settings in the toolbar that affect the graphics properties of the form. When the form is set to accept user drawing, the MousePointer is set to 2 (cross-hairs).

The drawing functionality is coded in the methods associated with the MouseDown and MouseMove events of the form:

```
* MouseDown
PARAMETERS nButton, nShift, nXCoord, nYCoord
IF THIS.MousePointer = 2
    THIS.PSet(nXCoord, nYCoord)
ENDIF

* MouseMove
PARAMETERS nButton, nShift, nXCoord, nYCoord
IF THIS.MousePointer = 2
    THISFORM.LINE(nXCoord, nYCoord)
ENDIF
```

Dynamically Format Grid Columns

File: SAMPLES\SOLUTION\CONTROLS\GRID\DYNGRID.SCX

This sample illustrates setting the DynamicForeColor and DynamicBackColor properties of grid columns.

The code of interest in this sample is associated with the InteractiveChange event of the drop-down list cboFormat.

First, the code clears the dynamic fore and back color settings:

```
oGrd.SetAll("dynamicbackcolor", "", "Column")  
oGrd.SetAll("dynamicforecolor", "", "Column")
```

Then, code in a CASE statement sets the new DynamicForeColor or DynamicBackColor properties. For example, the following line of code displays discontinued items with a gray ForeColor.

```
oGrd.SetAll("dynamicforecolor", ;  
    "IIF(discontinu, RGB(192,192,192), RGB(0,0,0))", "Column")
```

Discontinu is a logical field in the Products table.

Edit a Memo Field or Text File

File: SAMPLES\SOLUTION\CONTROLS\TXT_EDT\EDITBOX.SCX

This sample enables you to view and edit text from a memo field or a text file in an edit box.

Displaying text from a memo field is as easy as setting the ControlSource of the edit box to the memo field.

There are two ways you can edit a text file in an edit box: using low-level file functions and creating a cursor to hold the text. This sample creates a cursor instead of using low-level file functions.

Editing a Text File with Low-Level File Functions.

You can open a file using FOPEN(), read the contents of the file using FREAD(), and store the contents to a memory variable or to the Value property of the edit box. You can then write changes to the file using FWRITE() and close the file using FCLOSE().

Editing a Text File by Loading It into a Cursor Field

The advantage to creating a cursor for a text file, other than the fact that the code is a little simpler, is that Visual FoxPro manages writing large amounts of text in a cursor to temporary files if there isn't enough memory.

```
IF SELECT("textfile") = 0
    CREATE CURSOR textfile (filename c(60),mem m)
    APPEND BLANK
ENDIF
REPLACE textfile.FileName WITH GETFILE("TXT")
IF EMPTY(textfile.FileName)
    RETURN
ENDIF

SELECT textfile
APPEND MEMO mem FROM (textfile.FileName) OVERWRITE
THIS.Parent.edtText.ControlSource = "textfile.mem"
THIS.Parent.cmdSave.Enabled = .T.
THIS.Parent.lblFileName.Caption = ALLTRIM(textfile.FileName)
THIS.Parent.Refresh
```

Execute Commands at Specified Intervals

File: SAMPLES\SOLUTION\CONTROLS\TIMER\TIMECOMM.SCX

This sample form contains a text box and a spinner. Text you type in the text box is displayed in a WAIT WINDOW. You can set the spinner to the number of seconds you want to elapse between each WAIT WINDOW command.

In the InteractiveChange event of the spinner, the appropriate timer interval is set. The Interval property is 1/1000th of a second, so the interval is set to the spinner value * 1000:

```
THISFORM.Timer1.Interval = (THIS.Value * 1000)
```

In the Timer event, the WAIT WINDOW command is issued:

```
WAIT WINDOW ALLTRIM(THISFORM.Text1.Value) TIMEOUT 0.5
```

The code associated with the Timer event can include any command or procedure: code to update data, check for mail, display system resources, and so on.

Fill a List with Values from Different Sources

File: SAMPLES\SOLUTION\CONTROLS\LISTS\MULTDAT1.SCX

This sample demonstrates setting the RowSourceType of a list box at run time to a value, an alias, a SQL statement, a query (.QPR), an array, fields in a table, files in a directory, and the structure of a table.

The combo box on the form contains all the possible RowSourceTypes of a list (except for 9 - popup, which is included for backward compatibility). The code associated with the InteractiveChange event sets the new RowSourceType and RowSource properties for the list.

When you change both the RowSourceType and RowSource of a list box at run time, you need to:

1. Set the RowSourceType to 0
2. Set the RowSource to the new source
3. Set the RowSourceType to the new type

If you don't follow this order, the existing RowSource and RowSourceType settings could conflict after one new value is set and before the corresponding value is set.

Format Input and Validate Data in a Text Box

File: SAMPLES\SOLUTION\CONTROLS\TXT_EDIT\TEXTBOX.SCX

This sample shows how to set text box properties to make it easy for a user to enter data in the required format.

Format	Property	Setting
Allow Only Digits	InputMask	999999999
Select On Entry	SelectOnEntry	.T.
All Uppercase	Format	!
Read Only	ReadOnly	.T.
US Telephone Number	InputMask	(999) 999-9999
Password text	PasswordChar	*
Date Formatting	DateFormat	a number between 0 and 14

Validating Input

The following code in the Valid event of a text box prevents a user from leaving the text box if the letter 'a' is in the text.

```
IF "a"$ THIS.Value
    #DEFINE MESSAGE_LOC "The text box value cannot contain the letter 'a'"
    MESSAGEBOX (MESSAGE_LOC, 48+0+0)
    RETURN 0
ELSE
    RETURN .T.
ENDIF
```

Get Application Information from the Windows Registry

File: SAMPLES\SOLUTION\WINAPI\REGFILE.SCX

This sample shows how to access the Windows registry using the native Visual FoxPro DECLARE-DLL command. The Windows API provides a number of functions you can use to access, read and write to the registry. The REGISTRY.PRG class library in \SAMPLES\CLASSES contains a complete set of these functions you can use in your applications.

In this specific sample, you can use registry functions to check for the existence of a particular application. For example, you might want to distribute an application which relies on using Excel to automate the creation of a Pivot Table. The registry contains the location, version and other information about a particular application.

Get Version Information

File: SAMPLES\SOLUTION\WINAPI\GETVER.SCX

This sample allows you to display version information from a .DLL or .EXE.

A new function called `GetFileVersion()` has been added to `FOXTOOL.FLL` which allows you to obtain version information on a file. The following code illustrates how you can call the function. See `TOOLS\FOXTOOLS.HLP` for details on the specific array elements.

```
SET LIBRARY TO FoxTools ADDITIVE
DIMENSION aFileVer[12]
nRetVal = GetFileVersion(GetFile("EXE"),@aFileVer)
IF nRetVal = 0
    DISPLAY MEMO LIKE aFileVer
ENDIF
SET LIBRARY TO
```

Note Visual FoxPro now allows you to add File Version information to EXE and DLL files at build time. This version information is stored in a file resource and can be accessed via the Windows Explorer. EXE files created in Visual FoxPro 3.0 will not have a file version resource.

Graph Equations on a Form

File: SAMPLES\SOLUTION\FORMS\GRAPHICS\GRAPH.SCX

This sample shows how to graph equations with form graphics. The user can scale the equation by choosing the Zoom command buttons. The user can also move the graph's origin by clicking and dragging.

The graphing engines for this sample are in two programs: CGRAPH.PRG and PGRAPH.PRG. CGRAPH graphs equations based on Cartesian coordinates. PGRAPH graphs equations based on Polar coordinates. These programs use the PSet and Line methods of a form to draw the equation.

The following code, associated with the OneGraph method of the form set, runs the CGRAPH program when a user chooses the **Graph** button.

```
THISFORMSET.frmGraph.Draw  
  
    DO cgraph WITH ;  
        graph.equation, ;  
        graph.step, ;  
        graph.ecolor, ;  
        graph.connect, ;  
        THISFORMSET.nFormX, ;  
        THISFORMSET.nFormY, ;  
        .F., ;  
        THISFORMSET.frmgraph, ;  
        THISFORMSET.nFormScale
```

The following table lists the parameters for CGRAPH:

Parameter	Type	Description
equation	C	The equation in terms of X. Will plot the answer as Y.
step	N	Step increment.
ecolor	N	Equation color.
connect	L	Whether the previous point is connected to the current point with a line.
nFormX	N	Point on form where $x = 0$
nFormY	N	Point on form where $y = 0$
IAddCoords		Whether to draw coordinate lines.
frmgraph	C	Name of the form to write to.
nFormScale	N	Scale to graph the equation at

Index on Expressions

File: SAMPLES\SOLUTION\ADB\INDEX2.SCX

This index sample shows several options you might consider when using a character field to represent a Unique ID (one often comprised of numeric data). An index tag based on a character field will sort based on the ASCII values of the data whereas a numeric expression, using the VAL() function, will sort on numeric sequences. The following table shows the sort difference of two indexes.

CExpression	VAL(cExpression)
1	1
11	2
2	11

Note When you use the VAL() function, all non-numeric values will be converted to 0.

By incorporating the BINTOC() function, you can also reduce the size of your index tags considerably.

Create Small Indexes Using BINTOC()

File: SAMPLES\SOLUTION\ADB\INDEX1.SCX

This sample show various indexing schemes you can use with numeric and integer data types to get significant savings in index size and disk space consumed. In addition, smaller sized indexes usually result in faster searches. The new BINTOC() function allows you to convert an integer (numeric) value to a binary character representation.

Syntax

`BINTOC(nExpression [, nSize])`

Depending on the size of your expression, you can set the *nSize* parameter to accommodate the value of your expression in the least amount of characters.

The following line of code creates an index on a numeric field.

```
INDEX on line_no TAG line_no
```

The next time you are working with indexes on numeric integer data, consider using something like the following:

```
INDEX on BINTOC(line_no,1) TAG line_no
```

Interactively Build a SELECT Statement

File: SAMPLES\SOLUTION\FORMS\MAKESQL.SCX

This sample shows how to allow a user to build a custom query at run time. Combo boxes on the form allow a user to choose fields from the currently open table. The BldSQL method processes the names of the fields and the values users type into text boxes to create an executable SQL SELECT statement.

Additional methods, ValidateType and SetTextboxFormat, make sure that the appropriate values are entered into the text boxes and correctly incorporated into the SELECT statement.

After the WHERE clause has been constructed and stored to the variable lcWhere, the following command creates the SELECT statement:

```
lcSQL = "SELECT * FROM " + lcAlias + " " + lcWHERE
```

Once the SELECT statement is constructed, it can be executed with macro substitution:

```
&lcSQL
```

Manipulate Display Characteristics of a Graph

File: SAMPLES\SOLUTION\OLE\OLEGRAPH.SCX

This sample shows how to incorporate MS Graph into your applications. MS Graph ships with Visual FoxPro and can also be included in your distributed applications. MS Graph 5.0 is an OLE Automation Server that Visual FoxPro can automate using the standard CREATEOBJECT() function. With Visual FoxPro, Graph objects can be embedded in either bound or unbound OLE Controls. An OleBoundControl (bound to a General field) is the only way to programmatically insert data into a graph.

MS Graph's automation support provides access only to a graph object, not its datasheet. Data can be inserted into a Graph only with the APPEND GENERAL command on a General field. The following code uses the HasLegend property and is an example of automation.

```
cGData = ""+TAB+"Cats"+TAB+"Dogs"+CRLF+;
         "1994"+TAB+"11"+TAB+"22"+CRLF+;
         "1995"+TAB+"33"+TAB+"44"+CRLF+;
         "1996"+TAB+"55"+TAB+"55"+CRLF
APPEND GENERAL gen1 CLASS "msgraph.chart" DATA m.cGData
THIS.OleBoundControl1.ControlSource = "Gen1"
THIS.OleBoundControl1.HasLegend = .F.
```

The APPEND GENERAL command in this case creates a new chart object. If you do not include the CLASS "msgraph.chart" clause, then the chart is merely updated. The CLASS clause will create a new chart and override any existing formatting.

Move Items Between List Boxes

File: SAMPLES\SOLUTION\CONTROLS\LISTS\LMOVER.SCX

This sample demonstrates moving items from one list box to another. A user can double-click one item to move it, or select one or more items and drag them, or use the appropriate command buttons to move the items between the lists.

The two list boxes and four associated command buttons are saved as a class: MoverLists in SAMPLES.VCX. The base class of MoverLists is CONTAINER. To be able to add and remove items from lists, the RowSourceType of the lists must be set to 0 - None. If you want to fill the list box with array elements or values from a table, you can use code like the following:

```
*array
FOR i = 1 to ALEN(myarray)
    List.AddItem(myarray[i])
ENDFOR
```

```
*table
SCAN
    List.AddItem(table.field)
ENDSCAN
```

To Move Items by Double-Clicking

The following code is associated with the DblClick event of the left list box (lstSource). Similar code is associated with the DblClick event of the right list box (lstSelected).

```
THIS.Parent.lstSelected.AddItem(THIS.List(THIS.ListIndex))
This.RemoveItem(THIS.ListIndex)
THIS.Parent.Refresh
```

"THIS" refers to lstSource. "THIS.Parent" refers to the moverlists class, the container of the lists.

To Move All Items from One List to Another

The following code is associated with the Click event of cmdAddAll:

```
DO WHILE THIS.PARENT.lstSource.ListCount > 0
    THIS.PARENT.lstSelected.AddItem;
    (THIS.PARENT.lstSource.List(1))
    THIS.PARENT.lstSource.RemoveItem(1)
ENDDO
THIS.PARENT.Refresh
```

To Move Selected Items from One List to Another

If you remove an item from a list box, the ListCount property of the list box is decremented, as is the ListIndex of all the subsequent items in the list. To move multiply-selected items, you need to use a DO WHILE loop. The following code is associated with the Click event of cmdAdd:

```
nCnt = 1
DO WHILE nCnt <= THIS.PARENT.lstSource.ListCount
    IF THIS.PARENT.lstSource.Selected(nCnt)
        THIS.PARENT.lstSelected.AddItem;
        (THIS.PARENT.lstSource.List(nCnt))
        THIS.PARENT.lstSource.RemoveItem(nCnt)
    ELSE
        nCnt = nCnt + 1
    ENDIF
```

```
ENDDO
THIS.PARENT.Refresh
```

To Drag-and-Drop Items from One List to Another

Implementing drag-and-drop between the list boxes involves code associated with the MouseDown, MouseMove, DragOver, and DragDrop events. Three user-defined properties (DragThreshold, MouseX, and MouseY) extend the usability of the class.

The MouseDown code stores the X and Y coordinates of the mouse pointer to class properties.

```
*MouseDown
Parameters nButton, nShift, nXCoord, nYCoord
THIS.PARENT.MouseX = nXCoord
THIS.PARENT.MouseY = nYCoord
```

The MouseMove code makes sure the left mouse button is pressed before initiating the drag procedure. In addition, to guard against accidental dragging, this code checks to make sure that the user has moved the mouse a distance greater than a set threshold (8 pixels by default).

```
*MouseMove
Parameters nButton, nShift, nXCoord, nYCoord
IF nButton = 1 && Left Mouse
    IF ABS(nXCoord - THIS.PARENT.MouseX) > ;
        THIS.Parent.DragThreshold OR ;
        ABS(nYCoord - THIS.PARENT.MouseY) > ;
        THIS.Parent.DragThreshold
        THIS.Drag
    ENDIF
ENDIF
```

The DragOver code changes the DragIcon of the source when the mouse pointer enters and leaves the list box “air space.”

```
*DragOver
Parameters oSource, nXCoord, nYCoord, nState
DO CASE
    CASE nState = 0 && Enter
        oSource.DragIcon = THIS.Parent.CanDropIcon
    CASE nState = 1 && Leave
        oSource.DragIcon = THIS.Parent.NoDropIcon
ENDCASE
```

The DragDrop code makes sure that the source of the drag is not the same as the target of the drag and calls the method associated with the Click event of the cmdAdd command button (in the case of lstSelected) or cmdRemove (in the case of lstSource).

```
*DragDrop
Parameters oSource, nXCoord, nYCoord
IF oSource.Name != THIS.Name
    THIS.PARENT.cmdAdd.Click
ENDIF
```

Multiselect Items in a List Box

File: SAMPLES\SOLUTION\CONTROLS\LISTS\LMSEL.SCX

This sample shows how to manage multiple items selected in a list box. The form contains a list box with the MultiSelect property set to True (.T.). A combo box on the form contains all the items that are selected in the list box.

To process the multiple selected items in a list box – to copy them to an array or incorporate them elsewhere in your application – loop through the list items and process the ones for which the Selected property is set to true (.T.). The following code is included in the Click event of the list box to display the selected items in a combo box and the number of selected items in a text box:

```
nNoSelected = 0    && variable to track number of selected items
THISFORM.cboSelected.Clear    && clear the combo box

* main processing loop
FOR i = 1 TO THIS.ListCount
  IF THIS.Selected(i)
    nNoSelected = nNoSelected + 1
    THISFORM.cboSelected.AddItem (THIS.List(i))
  ENDF
ENDFOR

THISFORM.txtNoSelected.Value = nNoSelected
```

In the code associated with the Init event of this form, and in many places throughout this sample application, the character string values initially added to the list are first defined with a #DEFINE directive. Each of the defined constants ends with "_LOC". Microsoft internal localization tools extract these definitions so that the strings can be translated into whatever language the application is being localized in.

Nest Transactions

File: SAMPLES\SOLUTION\DB\TRANS.SCX

This sample illustrates beginning, ending, and rolling back transactions. You can decide to commit or discard changes to a single table when you use table buffering, but you can decide to commit or discard changes to any number of tables when you use transactions.

Before editing either table, click the **Begin** button. After editing, you can click **Begin** again to nest another transaction, or you can click **End** to commit your edits to the table or **Rollback** to discard your changes.

The key commands in this example are:

```
BEGIN TRANSACTION
```

```
END TRANSACTION
```

```
ROLLBACK
```

The function that returns the current number of nested transactions is:

```
TXNLEVEL ( )
```

Open Multiple Files Interactively

File: SAMPLES\SOLUTION\CONTROLS\LISTS\MULTIFILE.SCX

This sample shows how to populate a list with files. When you populate a list with files, built-in functionality allows a user to select new drives and directories. In this sample, a user can select one or more files in the list and open them for editing.

The following code loops through the items in the list box and opens the selected files for editing:

```
FOR nFile = 5 to THISFORM.lstFiles.ListCount
  IF THISFORM.lstFiles.Selected(nFile)
    MODIFY FILE (THISFORM.lstFiles.List(2) + ;
      THISFORM.lstFiles.List(nFile)) NOWAIT
  ENDIF
ENDFOR
```

When the RowSourceType property is set to 7 - Files:

- lstFiles.List(1) refers to the drive
- lstFiles.List(2) refers to the path
- lstFiles.List(3) is a separator line
- lstFiles.List(4) is [..]. Click to go to the parent directory

Files can start at lstFiles.List(5)

Pass Parameters between Forms

File: SAMPLES\SOLUTION\FORMS\PARAM.SCX

This solution illustrates passing parameters to a form and returning a value back.

PARAM Form

This form has the user enter a question and possible responses. Then, in the Click event of the cmdAsk button, the ParamAsk form is opened by passing the Question and Possible responses.

```
cParam1 = THISFORM.txtPassValue1.value  
nParam2 = THISFORM.opgPassValue2.value
```

```
DO FORM LOCFILE("ParamAsk.scx") WITH cParam1, nParam2 TO nRetValue
```

PARAMASK Form

The PARAMETERS are passed to the Init of the form where they are processed.

```
PARAMETERS cQuestion, nButtons  
THISFORM.txtQuestion.caption = cQuestion
```

In the Unload event of the form, the value stored in retvalue is returned back to the calling form:

```
RETURN THISFORM.retValue
```

Play an AVI File in an ActiveX Control

File: SAMPLES\SOLUTION\OLE\MMSAMPLE.SCX

The sample shows how to use the Multimedia control to play an AVI file in a control that has an hWnd property.

The control with the hWnd is a custom ActiveX control called hWin, created with the OLE control wizard in Microsoft Visual C++ 4.0. In creating the hWin .OCX, all the default options in the OLE Control wizard were chosen, all the default properties were added to the control, and the code was compiled.

In the Init of the form, the following line of code directs the Multimedia control to play the AVI in the hWin control (named Test).

```
ThisForm.VCR.hWndDisplay = ThisForm.Test.hWnd
```

Play Multimedia Files Using MCI Commands

File: SAMPLES\SOLUTION\FORMS\MCI_PLAY.SCX

This sample form uses MCI (Multimedia Command Interface) to play multimedia files and can play any visual or non-visual media that is installed on your system. However, the GETFILE() function prompts for the most common media files of .AVI, .WAV, .MOV, and .MID. To choose another file, select **All Files** and choose the desired file.

The documentation for the MCI commands is in Chapter 7 of the Windows SDK *Multimedia Programmer's Reference*.

Three Windows API functions are declared in the Init of the form:

- mciSendString
- mciGetErrorString
- SetWindowPos

The DoMCI method of the form executes an MCI command that is passed in as a parameter.

Classes

You can open the form to see all the code necessary to run multimedia files, but the functionality has also been extracted into classes that you can easily incorporate into your own applications.

VideoFrame Class

The VideoFrame class in the Visual FoxPro SAMPLES\CLASSES\SAMPLES.VCX class library can be used to play a visual multimedia file, such as a Video for Windows file. The class allows you to position and size the video to be played, and then provides built-in methods to easily play the media file.

For an example of using this class, see VIDEO.SCX in the Visual FoxPro SAMPLES\SOLUTION\FORMS directory.

Property	Description
AutoOpen	Specifies whether the video file should automatically opened and displayed when the object is instantiated. The default value is true (.T.).
AutoPlay	Specifies whether the video file should automatically play when it is opened. The default value is true (.T.).
AutoRepeat	Specifies whether the video file will loop the video. Setting this to .T. will cause the video to play continuously. The default value is false (.F.).
ControlSource	Specifies a Field that contains the video file reference. If empty, the class expects a static file name to be in the VideoFile property
MCIalias	Specifies the alias to be used by MCI. If left empty, the alias defaults to the Name property of the class. Normally this can be left empty, but if the user wants to play the same video file twice at the same time, a different alias would need to be specified for each.
VideoFile	Holds the name of a video file to play, for example "D:\VFP\SAMPLES\SOLUTION\FORMS\FOX.AVI"

Method	Description
CloseVideo	Closes the video file and releases all resources associated with it.
DoMCI	Called by the other methods to execute MCI commands. It can also be called by a user to execute a specific MCI command.
OpenVideo	Opens the video file and shows the first frame.
PauseVideo	Pauses a playing video. The video can be restarted by using the PlayVideo method
PlayVideo	Plays the video file. The video file must be opened in the OpenVideo method before it can be played.
SetPosition	Allows the user to specify the position of the media file. It can be executed at any time after the video file has been opened. Valid values are "Start", "End", or a specific millisecond into the video.

Sound Player Class

This class is also contained in the SAMPLES\CLASSES\SAMPLES.VCX class library. It can be used to play a non-visual multimedia file, such as a waveaudio file. The class allows you to specify the file to be played, and then provides built-in methods to easily play the media file.

Property	Description
AutoOpen	Specifies whether the sound file should be automatically opened and displayed when the object is instantiated. The default value is true (.T.).
AutoPlay	Specifies whether the sound file should be automatically played when it is opened. The default value is true (.T.).
AutoRepeat	Specifies whether the sound file plays continuously. The default value is false (.F.).
ControlSource	Specifies the column that contains the sound file reference. If empty, the class expects a static file name to be in the SoundFile property.
MCIAlias	Specifies the alias to be used by MCI. If left empty, the alias defaults to the Name property of the class. Normally this can be left empty, but if the user wants to play the same sound file twice at the same time, a different alias would need to be specified for each.
SoundFile	Holds the name of a sound file to play, for example "C:\WINDOWS\CHIMES.WAV"

Method	Description
OpenSound	Opens the sound file.
PlaySound	Plays the sound file. The file must be opened with the OpenSound method before it can be played.
PauseSound	Pauses the playing of a sound file. Play can be continued by calling the PlaySound method.
SetPosition	Allows the user to specify the position of the media file. It can be executed at any time after the file has been opened. Valid values are "Start", "End", or a specific

CloseSound

millisecond into the sound.

Closes the sound file and releases all resources associated with it.

Present a User with Multiple Choices

File: SAMPLES\SOLUTION\CONTROLS\BUTTONS\QUIZ.SCX

This sample illustrates using an option button group with no default selection and binding a character field to the option group.

- ▶ **To display an option group with no selected button, do one of the following**
 - Set the Value property of each of the option buttons in the group to 0 of .F.
 - Set the Value property of the option button group to 0.
- ▶ **To store the caption of option buttons to a table field**
 - Set the ControlSource of the option button group to the field.

Print a Percent of Total in a Report

File: SAMPLES\SOLUTION\REPORTS\PERCENT.FRX

The sample report, PERCENT.FRX, shows one way to calculate and display values at the beginning report before all the records from the record source have been printed. This sample report prints a percentage based on a total that is usually calculated and printed at the end of a report.

The sample report uses the tables, EMPLOYEE and ORDERS in its Data Environment. A public variable and expressions using the ORDER_AMT field are used to calculate the percentages.

To sort the data appropriately for the group defined in the report, the Order property on Cursor1 is set to the EMP_ID index.

The public variable, nTotalSales, defined in the Init event of Cursor2, stores the total for all orders. The variable is declared public to make it visible beyond the Cursor2 Init code.

A field control in the Group Footer band calculates and displays the percentage for each employee using the following expression:

```
STR(INT((emp_total / nTotalSales)*100)) + " " + "%"
```

A field control in the Summary band calculates and displays the total percentage for all employees using the following expression.

```
STR(INT((nTotalSales / nTotalSales)*100)) + " " + "%"
```

Print an Invoice

File: SAMPLES\SOLUTION\REPORTS\INVOICE.FRX

This report shows an invoice report that uses a view to combine information from multiple tables, a function to collect data from the user, and a one-to-many report layout. When the report is run, the view runs first and prompts the user for a date range using a function called DATEPICK.PRG. After the user provides the dates, the view runs and selects the records to display in the report.

The Record Source View, INVOICE, used in this report resides in the TESTDATA database provided with the Solution project. This view combines data from four tables, CUSTOMER, ORDERS, ORDITEMS, and PRODUCTS. To filter the tables for records within a date range, the view has two filters each with a parameter, dStart_Date and dEnd_Date.

The Data Environment The view is included in the data environment. To store values collected for the view and to restore the path setting, the BeforeOpenTables method of the data environment declares three public variables dStart_date, dEnd_Date, cOldPath. The first two match the parameters specified in the view. The third, cOldPath saves the current path setting so the path can be restored in the Destroy method. The path is then changed so the report is able to find the function which prompts the user for a range of dates. The final command runs the function.

The Destroy method of the data environment has public variables for the start and end dates, path setting to return the default path to what it was before the report was run, and release statements for all three of the public variables.

The DATEPICK.PRG This function defines and runs a form class called frmDATEPICK. The form class includes four comboboxes that display and collect values to use in the public variables, dStart_Date and dEnd_Date. The four variables collected are nFromMonth, nFromYear, nToMonth, and nToYear. These variables store the user input and are used to determine the value of dStart_Date and dEnd_Date. The function also checks to make sure the user entered values for each element of the date and verifies that the end date is after the start date.

The Report Layout In the report layout, a Group Header band holds the controls for the customer information, the Detail band has the controls for the order items, and the Group Footer band holds the controls that display the calculated values for the invoice.

To format the address appropriately, an expression in the control concatenates several fields with appropriate punctuation. To calculate values in the Group Footer band, two report variables, nSubtotal and nDiscount, store values for the subtotal and the percentage discount.

Print Customer Mailing Labels

File: SAMPLES\SOLUTION\REPORTS\CUST.LBX

This example illustrates printing mailing labels, one for each customer in the customer table.

The labels include 4 items:

- Company
- Contact + “, “ + Title
- Address
- City + “, “ + Region+ “ “ + Country

The logo is displayed on each label by using the OLE container control.

Print Data Dictionary Information

File: SAMPLES\SOLUTION\REPORTS\DBCTOFRX.SCX

This sample illustrates a simple way to generate a report with the contents of a Visual FoxPro database (DBC). Simply pick a DBC file when prompted and a report will be generated for you to preview. You can click on the Print button to generate a hard copy of it.

The sample creates a temporary cursor and inserts new records as the DBC is scanned. Table and view records are added along with their associated field and index information.

Print Product Orders By Quarter

File: SAMPLES\SOLUTION\REPORTS\ORDGRAPH.SCX

This sample illustrates generating and printing graphs on-the-fly. Visual FoxPro allows you add graphs to forms as either OleControls or OleBoundControls. The latter representing a graph which is bound to data via a General field.

If you intend to use the VFP report writer to print graphs, you need to use General fields to store the graphs. The easiest way to create a new graph in a General field is to use the APPEND GENERAL command, for example:

```
APPEND GENERAL graphfield CLASS "msgraph"
```

The example above will create a graph in a General field, however, it won't contain your data. The DATA clause actually passes data to the graph.

The following code loops through records already containing data and constructs a string to pass to the graph with APPEND GENERAL DATA.

```
SCAN NEXT m.totrecs
  m.cData = ""+TAB+m.f2+TAB+m.f3+TAB+m.f4+CRLF+;
    EVAL(fields(1))+ TAB + ;
    ALLTRIM(STR(EVAL(field(2))))+ TAB + ;
    ALLTRIM(STR(EVAL(field(3))))+ TAB + ;
    ALLTRIM(STR(EVAL(field(4))))
  m.cDetails = ;

f2+"-" +ALLTRIM(STR(EVAL(FIELD(2)))) ;
  + CRLF + f3+"-" +ALLTRIM(STR(EVAL(FIELD(3)))) ;
  + CRLF + f4+" - " +ALLTRIM(STR(EVAL(FIELD(4)))) +CRLF
  INSERT INTO prodsales ;
VALUES(SalesData.prod_name,tmpgrph.graph,m.cDetails)
  APPEND GENERAL prodsales.sales DATA m.cData
ENDSCAN
```

Once you have a table or cursor containing a General field with your graphs, you can then print it to a Visual FoxPro report using the REPORT FORM command. You cannot include an ActiveX control in a report.

Print in Catalog Format

File: SAMPLES\SOLUTION\REPORTS\WRAPPING.FRX

The sample report, WRAPPING.FRX, prints an employee directory and shows how you can wrap text around graphics, alternate the print position of controls, and float controls below fields that stretch. The data environment for this report contains the EMPLOYEE table from the TESTDATA.DBC in the Solution project.

The Report Layout

The Title band holds the name and description of the report. The Detail band holds the field and label controls that print for each record. For each record the report displays four categories of information.

- Employee name and title
- Biography
- Box of quick reference information such as phone and address
- Photograph

After printing the name and title, this report wraps the text of the biography around the photograph, alternates information from left to right, and floats the box down the page relative to the biography.

Wrap Text Around Graphics

To wrap the memo field around the picture, the report uses two field controls to display the contents of the memo field. The first control is above the picture to display two lines of text. The second field is also sized to show two lines of text, but is narrower and positioned next to the picture.



The first field control prints all of the words in the memo field up to the position where a break should occur to continue the words in the second control. The expression is shown below.

```
LEFTC (employee.notes, nWrapCharPos)
```

The second field control prints the remaining words from the memo field. The expression for the second field control is shown below.

```
LTRIM (RIGHTC (employee.notes, (nMemoLen- (nWrapCharPos) ) ) )
```

The expressions for the controls use report variables to determine the words the controls display from the memo field.

- nFirstMemoLen determines the number of characters that can be displayed in the first field without truncating. For this report, the variables value is always 185. If you use this in your own report, you will need to change this constant to match the maximum number of characters you want above the picture.

- `nMemoLen` determines the length of the memo field for the current record. This variable's expression is `LENC (employee.notes)`.
- `nSpace` determines the character position of the first space after the characters displayed in the first control. This variable's expression is `AT_C (CHR (32) , RIGHTC (employee.notes, (nMemoLen - nFirstMemoLen)))`.
- `nWrapCharPos` determines the character position within the memo field where the break should occur. This variable's expression is `nSpace + nFirstMemoLen`.

Alternate the Position of Printed Objects

In this report, the positions of the employee biography, the quick reference group, and the photograph alternate from left to right with each record. To alternate the positions, the report has two sets of controls for the parts of the report that alternate. Because two controls are used to wrap the biography text, the first control does not alternate and, therefore, does not need to be included in the two sets.

One set of controls specifies printing with the photograph on the left side. A second set specifies the printing for the photograph on the right side. The controls in both sets use a report variable, `nCounter`, that calculates a count which determines when the set should print. One set prints when `nCounter`'s value is 0; the other when the value is 1. The Print When expression for the first set of controls is `MOD (nCounter, 2) = 0`. For the second set, the expression is `MOD (nCounter, 2) = 1`.

Float Controls Below Controls that Stretch

This report prints the box of information prints below the biography. Since the length of the biography varies with each record, the control that displays the biography information next to the picture is set to stretch and uses only the space necessary to print the contents of the field. The box that prints below the biography floats down the page depending on the length of the biography. This box is a group of field, label, and rectangle controls. To prevent the biography from printing on top of the group, the field position for each of the controls in the group is set to float.

Print Report Details in a Ledger Style

File: SAMPLES\SOLUTION\REPORTS\LEDGER.FRX

The report , LEDGER.FRX, prints an employee phone list that alternates printing a gray background behind records. The data environment for this report contains the EMPLOYEE table from the TESTDATA.DBC in the Solution project.

In the report layout, the Page Header band has label controls that print the report title, description, and column headings. The Detail band has the field and label controls that print the information for each employee.

To print with ledger style, the controls for the employee information are on top of a rectangle. The rectangle's color is gray and it's mode is opaque. The report prints the rectangle for every other record by evaluating an expression in the Print When options of the rectangle control. The expression is $\text{MOD}(\text{nCounter}, 2) = 1$ which uses a report variable, nCounter that increments for each record printed. The report uses the value of this variable to determine when to print the rectangle.

Programmatically Check Table Properties

File: SAMPLES\SOLUTION\DB\INFO.SCX

This sample illustrates getting information about a table at run time.

Field Information

The AFIELDS() function provides most of the information about table fields displayed in this sample. In addition to the information displayed in this sample, AFIELDS() provides information about field and table validation expressions and messages, trigger expressions and messages, as well as long table name and table comment in the database.

Index Information

The TAG() and KEY() functions provide index information:

```
lo = THISFORM.edtProperties
FOR i = 1 TO 254
  IF !EMPTY(TAG(i))  && Checks for tags in the index
    lo.Value = lo.Value + TAG(i) + " " + KEY(i)
  ELSE
    EXIT
  ENDIF
ENDFOR
```

Programmatically Manipulate Objects

File: SAMPLES\SOLUTION\FORMS\OBJECTS.SCX

This sample shows how to set properties at run time for objects in a form set. The sample form set contains two forms. Code associated with the Click event of controls on both forms change the property settings of other controls.

When setting properties of controls at run time, remember to reference the control through its container hierarchy. For example, to set the Value property of a control on one form from code in a method of another form, reference the highest-level container (the form set), and all subsequent containers.

```
THISFORMSET.frmLeft.chkBold.Value = .F.
```

For a more detailed description of this sample, see Chapter 9, "Creating Forms," in the *Developer's Guide*.

Programmatically Manipulate Text

File: SAMPLES\SOLUTION\CONTROLS\TXT_EDT\TEXT.TXT

This sample illustrates using the SelStart, SelLength, and SelText properties of an edit box to manipulate the text at run time. The sample also counts the characters, words, and paragraphs in a text file and allows a user to search for a string in a text file.

Formatting Text

The following code is included in the Click event of the button that formats selected text to uppercase:

```
lo = THIS.Parent.edtText
lnOldStart = lo.SelStart
lnOldLength = lo.SelLength

lo.SelText = UPPER(lo.SelText)

lo.SelStart = lnOldStart
lo.SelLength = lnOldLength
```

If you want to specify the font attributes of selected sections of text, use a RichText control.

Searching for Text

After getting the text to search for, the following code loops through all the text in the edit box, comparing it to the target string.

```
llKeepLooking = .T.
DO WHILE llKeepLooking
  FOR i = lnStart TO LEN(loEDT.Value)
    loEDT.SelStart = i
    loEDT.SelLength = lnLen
    IF loEDT.SelText = ALLTRIM(loCBO.Text) OR ;
      (!llCaseSensitive AND ;
        (UPPER(loEDT.SelText) = UPPER(ALLTRIM(loCBO.Text))))
      llFound = .T.
      llKeepLooking = .F.
      EXIT
    ENDIF
  ENDFOR

  IF !llFound
    lnChoice=MESSAGEBOX("Search string not found.", ;
      64+0+4)
    IF lnChoice = 6 && Yes
      llKeepLooking = .T.
      lnStart = 0
    ELSE
      llKeepLooking = .F.
    ENDIF
  ENDIF
ENDDO
```

Tip Be sure to set the form's LockScreen property to true (.T.) before searching and false (.F.) after searching. Otherwise, the form will repaint every time the SelStart property of the edit box is changed.

Provide a Hierarchical Display of Items

File: SAMPLES\SOLUTION\OLE\OUTLINE.SCX

This sample shows two different types of ActiveX controls which you can use for outlining. In this sample, a directory structure is displayed. A custom method on the form named FillTree is called recursively to iterate through the entire tree structure.

The Outline control can be used for simple outlining, however, it does not offer the Windows 95 shell look available with the Treeview control. In addition, the Treeview control also allows display of images with each node.

Outline Control

You can add items to the outline control with the AddItem method. Indentation is handled through the Indent method which takes an index number representing the item to affect.

```
THIS.PageFrame1.Page1.OleOutline.AddItem(LOWER(m.path))
THIS.PageFrame1.Page1.OleOutline.Indent(m.cnt-1)= m.lvl
```

Treeview Control

The new Treeview control can give your applications a truly genuine Windows 95 look. Unlike the Outline control which uses methods and item indexes, the Treeview control uses a Nodes collection.

Note To display images in a treeview control, you must add an ImageList control to the form.

```
* Add items to treeview control
o = THIS.PageFrame1.Page2.oleTreeview
IF cnt = 1
    oNode = o.nodes.add(,1,LOWER(m.path)+"_",LOWER(m.path),,)
    oNode.Image = "world" &&name of image
ELSE
    oNode = o.nodes.add(m.pkey,4,LOWER(m.path)+"_",LOWER(m.path),,)
    oNode.Image = "fldr" &&name of image
ENDIF
```

Provide What's This Help on a Form

File: SAMPLES\SOLUTION\FORMS\WHATTHIS\WHATTHIS.SCX

When you provide What's This Help on a form, a button with a question mark is displayed in the upper right corner of the form.



When a user clicks this button or presses SHIFT+ F1, the form is set to WhatsThis mode. You can also programmatically enter this state by calling the WhatsThisMode method of the form. When the form is in WhatsThis mode, the mouse pointer changes. A user can click a control and get context-sensitive help for that control in a popup window.

Note If the WhatsThisHelpID of a control or a form is set to -1 (the default), the text in the popup window indicates that no help topic is available for the control.

To provide What's This Help

- 1 Create a Help file with a topic for each control on the form.
- 2 In the Map section of the Help project, map HelpContextID values to the topics.
- 3 Set the WhatsThisHelp property of the form to true (.T.).
- 4 Set the WhatsThisButton property of the form to true (.T.).
- 5 Set the WhatsThisHelpID of the controls to the appropriate helpcontextID values.
- 6 In the Load or Init of the form, use the SET HELP TO command to specify the help file for the form.

In addition to WHATTHIS.SCX, this sample includes the following files:

File	Description
WHATTHIS.HLP	The Help file for the form
WHATTHIS.HPJ	The Help project file
WHATTHIS.RTF	The source document for the Help file

Read and Write Visual FoxPro Registry Values

File: SAMPLES\SOLUTION\WINAPI\REGFOX.SCX

This example shows how to access the Windows Registry using the native Visual FoxPro DECLARE-DLL command. The Windows API provides a number of functions you can use to access, read and write to the registry. The REGISTRY.PRG class library in \SAMPLES\CLASSES contains a class definition that exposes these functions as methods that you can call in your applications.

The contents of the Visual FoxPro Options dialog, Field Mapping settings and Label definitions (to name a few) are stored in the Registry. Since many of these settings are not available via SET functions, you can use Registry functions to access these values. The following code populates an array of all the settings in the Options dialog.

```
regfile = HOME()+"samples\classes\registry.prg"  
SET PROCEDURE TO (m.regfile) ADDITIVE  
oReg = CreateObject("FoxReg")
```

```
DIMENSION aFoxOptions[1,2]  
m.nErrNum = oReg.EnumFoxOptions(@aFoxOptions)
```

The following code calls the SetFoxOption method of the FoxReg class (defined in REGISTRY.PRG) to set TALK OFF in the registry.

```
regfile = HOME()+"samples\classes\registry.prg"  
SET PROCEDURE TO (m.regfile) ADDITIVE  
oReg = CreateObject("FoxReg")  
  
m.nErrNum = oReg.SetFoxOption("TALK","OFF")
```

Read ODBC Registry Values

File: SAMPLES\SOLUTION\WINAPI\REGODBC.SCX

This example shows how to access the Windows Registry using the native Visual FoxPro DECLARE-DLL command. The Windows API provides a number of functions you can use to access, read and write to the registry. The REGISTRY.PRG class library in \SAMPLES\CLASSES contains a class definition that exposes these functions as methods that you can call in your applications.

The Registry contains a listing of all installed ODBC drivers and data sources. If your application relies on using ODBC, then you might want to perform a Registry check to see if a particular driver or data source is installed. This example shows how to query for ODBC information.

```
LOCAL oReg, regfile, nErrNum, lDrivers
PUBLIC aODBCData

regfile = HOME()+"samples\classes\registry.prg"
SET PROCEDURE TO (m.regfile) ADDITIVE

oReg = CreateObject("ODBCReg")

DIMENSION aODBCData[1]
m.nErrNum = oReg.GetODBCDrvrs(@aODBCData)
```

Refresh a Graph in a Form

File: SAMPLES\SOLUTION\MOLE\STOCK.SCX

This sample illustrates selecting data from a table and passing it to MS Graph to refresh the values in a chart. Most of the code to do this is associated with the InteractiveChange event of the cboMonth combo box:

Select the data into a cursor

```
SELECT date, close;  
  FROM Stock1 WHERE MONTH(date) = THIS.Value ;  
  ORDER BY date INTO CURSOR wtemp
```

Create a character string containing the selected data

```
SELECT wtemp  
  
lcData = " " + TAB + "Closing Price" + CRLF  
  
SCAN  
  lcData = lcData + DTOC(date)  
  lcData = lcData + TAB  
  lcData = lcData + ALLTRIM(STR(close)) + CRLF  
ENDSCAN
```

Send the character string to MS Graph

```
SELECT Graph  
APPEND GENERAL msgraph DATA lcData
```


Resize and Reposition Controls at Run Time

File: SAMPLES\SOLUTION\FORMS\CRESIZE.SCX

A user can resize this sample form as desired. The controls on the form are resized or repositioned relative to the new Height and Width properties of the form.

This form uses the resizable class in SAMPLES\CLASSES\SAMPLES.VCX to manage resizing and repositioning of the controls. The following line of code is added to the Resize event of the form, calling the AdjustControls method of the resizable class:

```
THIS.Resizable2.AdjustControls
```

In the Init of the resizable class, code loops through all the controls in the form, storing their positions and sizes relative to the form in an array. When the form is resized, the AdjustControls method resizes and repositions the controls to their relative sizes and positions.

You can add more controls to the form and the new controls will also be resized and repositioned.

The RepositionList and ResizeList properties contain a space-delimited list of all the classes that you want to reposition and resize. If you don't want a certain class to be resized, remove it from the ResizeList.

You can prevent a user from making the form too small for the objects contained on it by setting the form's MinHeight and MinWidth properties.

Return a Value from a Form

File: SAMPLES\SOLUTION\FORMS\LOGFORM.SCX

This sample illustrates returning a value from a login form. The launching form (LOGFORM.SCX) uses the DO FORM command to run the login form and store the return value to a variable (cUser).

```
DO FORM Login TO cUser
```

Note To return a value from a form, the WindowType property of the form must be set to 1 - Modal.

The login form (LOGIN.SCX) allows a user to enter a user name and a password. Code associated with the Click event of cmdOK checks to make sure that the correct password was entered:

```
LOCATE FOR UPPER(login.userid) = UPPER(ALLTRIM(THISFORM.txtUserName.Value))

IF FOUND() AND ALLTRIM(password) == ALLTRIM(THISFORM.txtPassword.Value)
    THISFORM.cUser = ALLTRIM(login.userid)
    THISFORM.Release
ELSE
    #DEFINE MISMATCH_LOC "The user name or password is incorrect. Please try
again."
    WAIT WINDOW MISMATCH_LOC TIMEOUT 1.5
    THISFORM.txtUserName.Value = ""
    THISFORM.txtPassword.Value = ""
    THISFORM.txtUserName.SetFocus
ENDIF
```

Code associated with the Unload event of the login form returns the name of the user, if the user entered the correct password, or an empty string:

```
RETURN THIS.cUser
```

Run Multiple Instances of a Form

File: SAMPLES\SOLUTION\FORMS\LAUNCH.SCX

This sample demonstrates running multiple instances of a form. `aForms` is an array property of the form that launches the multiple instances. The following code is associated with the Click event of the command button that launches multiple forms:

Get the number of the last element in the array

```
nInstance = ALEN(THISFORM.aForms)
```

Determine the Top and Left Properties to cascade the new forms. These settings are passed as parameters when the form instances are launched.

```
IF nInstance > 1 AND ;
    TYPE('THISFORM.aForms[nInstance -1]') = 'O'
    nFormTop = THISFORM.aForms[nInstance -1].Top + 1
    nFormLeft = THISFORM.aForms[nInstance -1].Left + 1
ELSE
    nFormTop = 1
    nFormLeft = 1
ENDIF
```

6

Set the caption to reflect the instance number

```
cFormCaption = "Instance" + ALLTRIM(STR(nInstance))
```

Run the form and assign the object variable to the array element. The `LINKED` keyword indicates that all instances will be released when the array is released. Without `LINKED`, the multiple instance forms would persist after the array is released.

```
DO FORM Multi NAME THISFORM.aForms[nInstance] WITH ;
    nFormTop, nFormLeft, cFormCaption LINKED
```

Redimension the array so that more instances of the form can be launched.

```
DIMENSION THISFORM.aForms[nInstance + 1]
```

See Check Box Design Options

File: SAMPLES\SOLUTION\CONTROLS\CHECKBOX\CHECKBOX.SCX

This sample illustrates a variety of properties that can be set to customize check box display.

Property	Description
ControlSource	A variable or table field that is numeric or logical
Picture	Bitmap displayed if Style is Graphical
Style	Graphical for pictures or standard for a box
Value	.T., .F., 0, 1, 2, or .NULL.

See Command Button Design Options

File: SAMPLES\SOLUTION\CONTROLS\BUTTONS\CMDBTN.SCX

This sample illustrates a variety of properties that can be set to customize command button display.

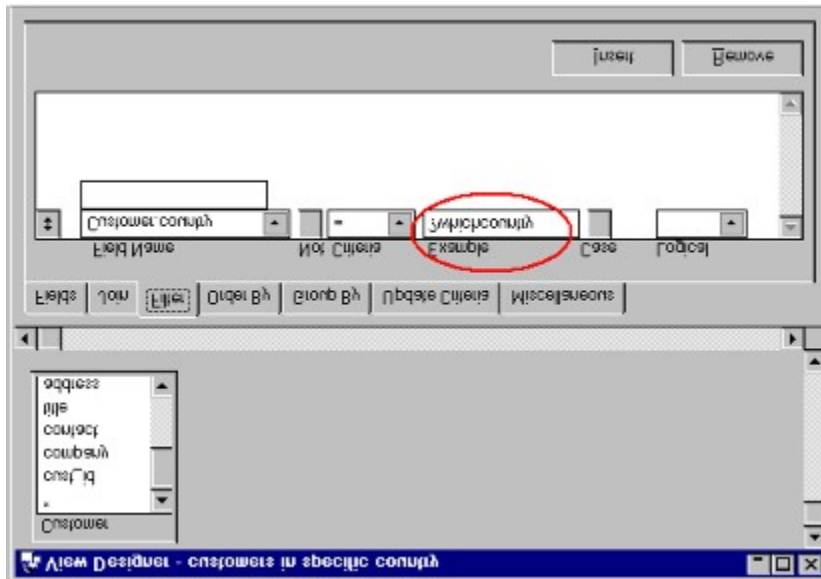
Property	Description
Cancel	The Click event code of the command button with Cancel set to true (.T.) is executed when a user presses ESC.
Default	The Click event code of the command button with Default set to true (.T.) is executed when a user presses ENTER
Picture	Bitmap displayed if Style is Graphical

Select Customers in a Specific Country

File: SAMPLES\SOLUTION\DATA\TESTDATA.DBC

This sample illustrates using a parameterized view ("customers in specific country" in the testdata database) to display a subset of records that match a variable set at run time.

To create a parameterized view in the View Designer, enter a variable prefixed with a question mark (?) in the **Example** box of the **Filter** tab.



If the variable doesn't exist at run time, the user is prompted to enter a value for the variable, `whichcountry` in this example. If the variable already exists, the view displays matching records without prompting for a new value.

Select Records from a Full Outer Join

File: SAMPLES\DATAT\FOUTERJ.QPR

The query, FOUTERJ, in the Solution project combines information from the country table and the customer table using a full outer join. Each result record has a field for each of the fields from the country table, and the `country` and `cust_id` field for the field from the customer table as specified in the SELECT clause of the SELECT-SQL statement.

```
SELECT Country.*, Customer.country, Customer.cust_id;  
FROM testdata!customer FULL JOIN country ;  
ON Customer.country = Country.country
```

Typically, a full outer join answers three questions about the records in your database. Some of the questions this query could answer are

- Which customers are in which countries?
- Which countries do not have any of our customers in them?
- Which customer records are missing country information?

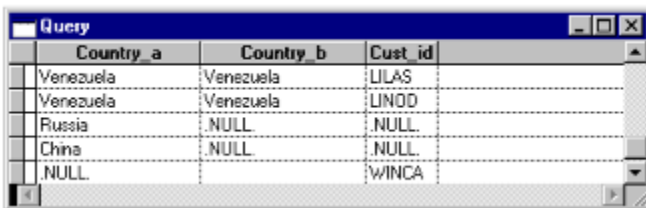
This information might help someone decide if they should expand their business to other countries, change their marketing strategy in some countries, or simply that some records in the database need the country information updated.

The full outer join returns all records from both tables and combines records that match the join condition. The result set includes three subsets of records.

- Records matching the join condition that combine information from a record in each table.
- Records from the country table that do not match the join condition.
- Records from the customers table that do not match the join condition.

Because each record in the results has the same fields, the records that did not have a match in the other table have NULL values in the fields that would otherwise hold values the other table. For example, if a record for the country, Russia, did not have any related customer records in the customer table, that record appears in the results with NULL as the value of the field, `cust_id` and `country_b`.

NULL values appear in fields for non-matching records.



Country_a	Country_b	Cust_id
Venezuela	Venezuela	LILAS
Venezuela	Venezuela	LINOD
Russia	NULL	NULL
China	NULL	NULL
NULL	NULL	WINCA

You can change the results of the query by specifying filters, a sort order, group, or other miscellaneous options for the query.

Select Records from a Left Outer Join

File: SAMPLES\DATA\LOUTERJ.QPR

The query, LOUTERJ, in the Solution project combines information from the orders table and the customer table using a left outer join. Each result record has a fields for the `order_id` from the orders table and for the `cust_id`, `company`, `country` fields from the customer table as specified in the SELECT clause of the SELECT-SQL statement.

```
SELECT Customer.cust_id, Customer.company, Customer.country, ;
Orders.order_id;
FROM testdata!orders LEFT OUTER JOIN testdata!customer ;
ON Orders.cust_id = Customer.cust_id
```

Typically, a left outer join can answer two questions about the records in your database. Some of the questions this query could answer are:

- Which orders belong to which customers?
- Which orders do not have related customer information?

This information helps the database administrator determine which order records are missing information.

The left outer join returns all records from the table on the left of the join condition combined with the records from the table on the right of the condition that match the condition. The results set includes two subsets of records.

- Records matching the join condition that combine information from a record in each table.
- Records from the orders table that do not match the join condition.

Because each record in the results has the same fields, the records with an `order_id` that did not have a match in the customer table have NULL values in the fields that would otherwise hold values from the customer table. For example, if the record for order 11079, did not have any related customer records in the customer table, that record appears in the results with the value NULL in the fields, `cust_id`, `company`, and `country`.

The order for 11079 does not have a match any record in the Customer table



Cust_id	Company	Country	Order_id
RICSU	Richter Supermarkt	Switzerland	11075
BONAP	Bon app	France	11076
RATTC	Rattlesnake Canyon Grocery	USA	11077
RATTC	Rattlesnake Canyon Grocery	USA	11078
NULL	NULL	NULL	11079

You can change the results of the query by specifying filters, a sort order, group, or other miscellaneous options for the query.

Select Records from a Nested Join

File: SAMPLES\DATA\NESTED.QPR

The query, NESTED, in the Solution project combines information from the customer, orders, and orditems tables using two inner join conditions. One join condition is between customer and orders; the other between orders and orditems. Each result record has fields for the `cust_id` and `company` from the customers table, `order_id` from the orders table, and `line_no` from the orditems table as specified in the SELECT clause of the SELECT-SQL statement.

```
SELECT Customer.cust_id, Customer.company, Orders.order_id,  
Orditems.line_no;  
FROM testdata!customer INNER JOIN testdata!orders;  
INNER JOIN testdata!orditems ;  
ON Orders.order_id = Orditems.order_id ;  
ON Customer.cust_id = Orders.cust_id
```

The order in which the tables are joined determines the order the join conditions are evaluated. In this query, the join between orders and orditems is evaluated first and produces a subset of records that meets the join condition. This subset of records is matched to the records in the customer table and produces the final results set.

You can change the results of the query by specifying filters, a sort order, group, or other miscellaneous options for the query.

Select Records from a Right Outer Join

File: SAMPLES\DATA\ROUTERJ.QPR

The query, ROUTERJ, in the Solution project uses the testdata database, and combines information from the orders table and the employee table using a right outer join. Each result record has a field for the `order_id` from the orders table and for the `lastname` field from the employee table.

```
SELECT Orders.order_id, Employee.last_name;  
FROM testdata!employee RIGHT OUTER JOIN testdata!orders ;  
ON Employee.emp_id = Orders.emp_id
```

Typically, a right outer join can answer two questions about the records in your database. Some of the questions this query could answer are

- Which orders were entered by which employees?
- Which orders do not have an employee specified?

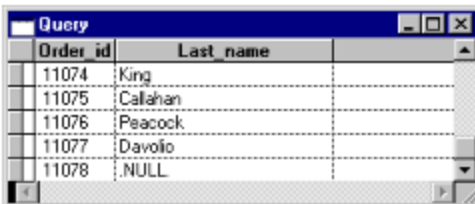
This information may help track specific orders and identify orders that do not have an employee specified as placing the order.

The right outer join retrieves all records from the table on the right of the join condition combined with the records from the table on the left that match the join condition. The results set includes two subsets of records.

- Records matching the join condition that combine information from a record in each table.
- Records from the orders table that do not match the join condition.

Because each record in the results has the same fields, the records with a `lastname` that did not have a match in the orders table have NULL values in the field that would otherwise hold values from the orders table. For example, if a record for order number 11078 did not have any related employee records in the employee table, that record appears in the results with the value NULL in the field, `lastname`.

The record for order 11078 without a matching record in the EMPLOYEE table.



Order id	Last name
11074	King
11075	Calahan
11076	Peacock
11077	Davolio
11078	NULL

You can change the results of the query by specifying filters, sort order, group, or other miscellaneous options for the query.

Select Records from an Inner Join

File: SAMPLES\DATA\INNERJ.OPR

The query, INNERJ, in the Solution project uses the database, TESTDATA, and combines information from the tables, ORDERS and CUSTOMER using an inner join condition. Each result record has fields for the CUST_ID, COMPANY, and COUNTRY from the CUSTOMERS table, and ORDER_ID from the ORDERS table as specified in the SELECT clause of the SELECT-SQL statement.

```
SELECT Customer.cust_id, Customer.company, Customer.country, ;
Orders.order_id;
FROM testdata!orders INNER JOIN testdata!customer ;
ON Orders.cust_id = Customer.cust_id
```

This query retrieves and combines the records between the two tables that match the join condition. The condition specifies that the records must have the same value in the `cust_id` field.

The inner join returns only records that match the join condition.



The screenshot shows a window titled "Query" displaying the results of an inner join. The table has four columns: Cust_id, Company, Country, and Order_id. The data is as follows:

Cust_id	Company	Country	Order_id
WHITC	White Clover Markets	USA	11066
DRACD	Drachenblut Delikatessen	Germany	11067
QUEEN	Queen Cozinha	Brazil	11068
TORTU	Tortuga Restaurante	Mexico	11069
LEHMS	Lehmanns Marktstand	Germany	11070

An inner join typically answers one basic question. In this query, the question is what orders did each customer place? If a customer has a record in the database but has not placed an order, the record does not appear in the results because it is not part of the subset that matches the join condition.

You can change the results of the query by specifying filters, a sort order, group, or other miscellaneous options for the query.

Select Records from Both Inner and Outer Joins

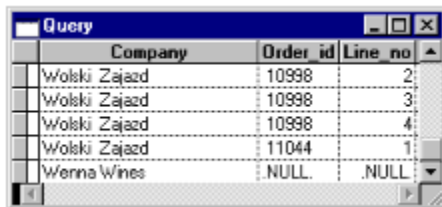
File: SAMPLES\DATA\COMBOJ.QPR

The query, COMBOJ, uses the testdata database, and combines information from the tables, customer, orders, and orditems. Each result record has fields for the `company` from the customer table, the `order_date` from the orders table, and the `line_no` from the orditems table as specified in the SELECT clause of the SELECT-SQL statement.

```
SELECT Customer.company, Orders.order_id, Orditems.line_no;  
FROM testdata!customer LEFT OUTER JOIN testdata!orders;  
    INNER JOIN testdata!orditems ;  
ON Orders.order_id = Orditems.order_id ;  
ON Customer.cust_id = Orders.cust_id
```

The order the tables are joined determines the order the join conditions are evaluated. In this query, the join between orders and orditems is evaluated first and produces a subset of records that meets the join condition. Because of the inner join, the subset of records has only records from both table that match the condition. This subset of records is matched to the records in the customer table. Because of the outer join between customer and orders, any customers not having orders also appear in the results and have the value NULL.

Customers without orders are included in the results.



Company	Order id	Line no
Wolski Zajazd	10998	2
Wolski Zajazd	10998	3
Wolski Zajazd	10998	4
Wolski Zajazd	11044	1
Wenna Wines	NULL	NULL

You can change the results of the query by specifying filters, a sort order, group, or other miscellaneous options for the query.

Select the Ten Worst Selling Products

File: SAMPLES\DATA\WORSTTEN.QPR

This query displays the worst ten selling products based on sales recorded in the Order Items table. The query uses the TOP *n* feature of the SELECT statement, and, because the default order is from smallest to largest, ordering by the sum of price and quantity returns the 10 lowest values.

```
SELECT TOP 10 Products.prod_name, ;
    SUM(Orditems.unit_price*Orditems.quantity);
FROM testdata!products INNER JOIN testdata!orditems ;
    ON Products.product_id = Orditems.product_id;
GROUP BY Products.prod_name;
ORDER BY 2
```

Select the Top Ten Best Selling Products

File: SAMPLES\DATA\TOPTEN.QPR

This query displays the top ten selling products based on sales recorded in the Order Items table. The query uses the TOP *n* feature of the SELECT statement. Because the order is set to DESCENDING, ordering by the sum of price and quantity returns the 10 highest values.

```
SELECT TOP 10 Products.prod_name, ;
    SUM( Orditems.unit_price* Orditems.quantity);
FROM testdata!products INNER JOIN testdata!orditems ;
    ON Products.product_id = Orditems.product_id;
GROUP BY Products.prod_name;
ORDER BY 2 DESC
```

Send Mail

File: SAMPLES\SOLUTION\OLE\SENDMAIL.SCX

This sample illustrates using the mailbtn class in SAMPLES\CLASSES\BUTTONS.VCX to create a simple messaging form that can send Visual FoxPro data to an e-mail address.

Mailbtn is a container class that contains a command button and two ActiveX controls: MAPI Session and MAPI Messages, both of which are defined in MSMAPI32.OCX. The MAPI Session control establishes a MAPI session, and the MAPI Messages control allows the user to perform a variety of messaging system functions.

The mailbtn class starts a new Mail session, collects data from the current record, and brings up the Send Mail dialog with the data inserted as the message text.

This class includes two custom methods AddTabs and StripPath for formatting the information gathered from the table and inserted in the mail message.

This class also takes advantage of another custom method called Signon as well as a custom property called logsession, which is set to .F. initially.

When the user clicks the cmdMail button, code in the Click event calls the signon method, setting logsession to true (.T.):

```
this.logsession = .T.  
this.OLEMSess.signon
```

If there is a failure on signon, the Error event of the class is called and logsession is set to .F.

The MAPI controls are invisible at run time. In addition, there are no events for the controls. To use them, you must specify the appropriate methods. For these controls to work, MAPI services must be present. MAPI services are provided in Microsoft Mail and Exchange electronic mail systems for Microsoft Windows version 3.0 or later.

Sort List Box Items

File: SAMPLES\SOLUTION\CONTROLS\LISTS\LSORT.SCX

This sample demonstrates allowing a user to rearrange or sort the items in a list. The key properties to set are:

```
List.MoverBars = .T.  
List.Sorted = .T.
```

Set the MoverBars property to true (.T.) to allow a user to reorder an item in the list by dragging the button to the left of the item to the new position.

Set the Sorted property to true (.T.) to display the list items in alphabetical order. You can still rearrange the items in the list after they have been sorted. Set the Sorted property to .T. again to sort the list in alphabetical order again.

The Sorted property applies only if the RowSourceType property of the list is set to 0 (None) or 1 (Value).

Sort or Order A Table At Run Time

File: SAMPLES\SOLUTION\ADB\ORDER.SCX

This sample illustrates changing the order that records in a table are listed. The customer table has index tags on the fields displayed in the grid. In the MouseUp event of each of the headers in the grid, the order of the table is set to the index tag associated with the field displayed in the column. For example, the following code is associated with the MouseUp event of the Company header:

```
LPARAMETERS nButton, nShift, nXCoord, nYCoord
IF nShift = 2 && CTRL
    SET ORDER TO Company DESCENDING
ELSE
    SET ORDER TO Company ASCENDING
ENDIF

GO TOP
THISFORM.Refresh
```

Use API Functions that Require a STRUCT

File: SAMPLES\SOLUTION\WINAPI.SYSTIME.SCX

This example illustrates calling the Windows API function `GetSystemTime`. `GetSystemTime` fills in a structure of WORD (16-bit unsigned integer) values with system time information.

C Function Declaration and Struct Definition

```
VOID GetSystemTime(  
    LPSYSTEMTIME lpSystemTime    // address of system time structure  
);
```

This is the struct definition

```
typedef struct _SYSTEMTIME {  
    WORD wYear;  
    WORD wMonth;  
    WORD wDayOfWeek;  
    WORD wDay;  
    WORD wHour;  
    WORD wMinute;  
    WORD wSecond;  
    WORD wMilliseconds;  
} SYSTEMTIME;
```

Calling the Function in Visual FoxPro

The Visual FoxPro code passes `GetSystemTime` a reference to a character variable, which is filled in with the WORD values.

```
* Visual FoxPro Code: cmdSystemTime.Click  
DECLARE INTEGER GetSystemTime IN win32api STRING @  
cBuff=SPACE(40)
```

```
GetSystemTime(@cBuff)
```

To retrieve the information from the character variable, `cBuff`, the following code converts 8 bit ASCII characters for year and month in the variable into 16 bit equivalents.

```
THIS.Parent.lblYear.Caption = ALLTRIM(STR(ASC(SUBSTR(cBuff,2)) * 256 +  
ASC(SUBSTR(cBuff,1))))
```

```
THIS.Parent.lblMonth.Caption = MONTH_LOC + ALLTRIM(STR(ASC(SUBSTR(cBuff,4))  
* 256 + ASC(SUBSTR(cBuff,3))))
```

Use API Functions that Require Pointers to Arrays

File: SAMPLES\SOLUTION\WINAPI\SYSCOLOR.SCX

This example uses two Windows API functions to first get the system color settings (GetSysColor) and then to reset the system colors to new values (SetSysColors). Two of the three arguments required by SetSysColors are pointers to native C arrays.

GetSysColor

The GetSysColor function accepts an integer parameter, a number between 0 and 18, and returns a DWORD (32-bit unsigned integer) indicating the current color setting.

```
DWORD GetSysColor(  
    int nIndex // display element  
);
```

nIndex is a value representing an area of the interface, as defined in the following list.

```
#define COLOR_SCROLLBAR          0  
#define COLOR_BACKGROUND        1  
#define COLOR_ACTIVECAPTION     2  
#define COLOR_INACTIVECAPTION   3  
#define COLOR_MENU              4  
#define COLOR_WINDOW            5  
#define COLOR_WINDOWFRAME      6  
#define COLOR_MENUTEXT          7  
#define COLOR_WINDOWTEXT       8  
#define COLOR_CAPTIONTEXT      9  
#define COLOR_ACTIVEBORDER     10  
#define COLOR_INACTIVEBORDER   11  
#define COLOR_APPWORKSPACE     12  
#define COLOR_HIGHLIGHT        13  
#define COLOR_HIGHLIGHTTEXT    14  
#define COLOR_BTNFACE           15  
#define COLOR_BTNSHADOW        16  
#define COLOR_GRAYTEXT         17  
#define COLOR_BTNTEXT          18  
#define COLOR_INACTIVECAPTIONTEXT 19  
#define COLOR_BTNHIGHLIGHT     20  
  
#if(WINVER >= 0x0400) /* Windows 95 differences */  
#define COLOR_3DDKSHADOW       21  
#define COLOR_3DLIGHT          22  
#define COLOR_INFOTEXT         23  
#define COLOR_INFOBK           24  
  
#define COLOR_DESKTOP           COLOR_BACKGROUND  
#define COLOR_3DFACE            COLOR_BTNFACE  
#define COLOR_3DSHADOW          COLOR_BTNSHADOW  
#define COLOR_3DHIGHLIGHT       COLOR_BTNHIGHLIGHT  
#define COLOR_3DHILIGHT         COLOR_BTNHIGHLIGHT  
#define COLOR_BTNHILIGHT        COLOR_BTNHIGHLIGHT  
#endif /* WINVER >= 0x0400 */
```

SetSysColors

The SetSysColors function requires three arguments: the number of elements in two arrays and the

addresses of the two arrays.

```
BOOL WINAPI SetSysColors(  
    int cElements,           // number of elements to change  
    CONST INT *lpaElements, // address of array of elements  
    CONST COLORREF *lpaRgbValues // address of array of RGB values  
);
```

Calling SetSysColors in Visual FoxPro

The following code is extracted from cmdSetSysColors.Click:

```
DECLARE INTEGER SetSysColors IN win32api INTEGER, STRING, STRING
```

Construct the elements of the first array in a character variable.

```
cElements = ""  
FOR i = 0 TO 18  
    cElements = cElements + THISFORM.DecToHex(i)  
ENDFOR
```

Construct the elements of the second array in a character variable. The cursor that is scanned was filled with GetSysColor values in the Init of the form.

```
cColors = ""  
SCAN  
    cColors = cColors + THISFORM.DecToHex(INT(color))  
ENDSCAN
```

```
=SetSysColors(18,cElements,cColors)
```

Use Conditional Formatting in a Report

File: SAMPLES\SOLUTION\REPORTS\COLORS.FRX

This sample report demonstrates how you can use Print When conditions for controls to change field formatting based on the values in the record. The sample prints a "Product Inventory Report." If the product item is discontinued, it is printed in ~~strikeout~~. If the In Stock amount is below the reorder level and the product is not discontinued, the item is printed in red. Otherwise, the product item is printed in standard black.

This report uses the table, PRODUCTS, from the database, TESTDATA, in the Data Environment. The Detail band has field controls for the product information such as ID, name, amount in stock, and amount on order. Another field control prints an reminder of the inventory level the reader of the reorder amount.

To handle the three cases, three sets of field controls for the product information are layered on top of each other in the Detail band. For the first set, the text color is red; for the second, the color is black; and for the third, the font effect is ~~Strikeout~~.

Each control set has a different Print When expression.

- For the plain black fields:
`(products.in_stock > products.reorder_at) and products.discontinuu = .F.`
- For the ~~strikeout~~ fields:
`products.discontinuu = .T.`
- For the red text color fields:
`products.in_stock <= products.reorder_at and products.discontinuu = .F.`

Use Slider and Status Bar Controls

File: SAMPLES\SOLUTION\MOLE\SLIDER.SCX

This sample illustrates using the Slider control and the StatusBar control.

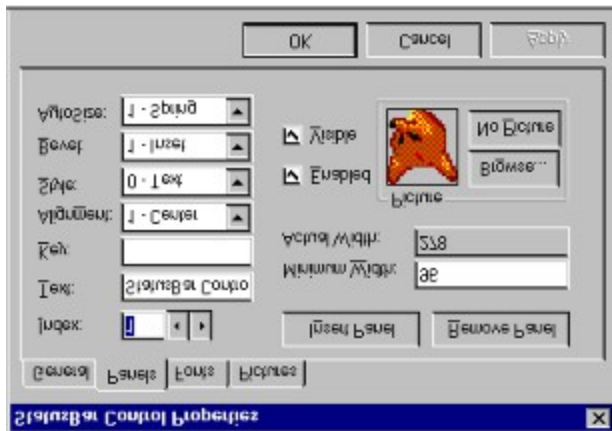
Slider Controls

Code in the Scroll event of each slider control sets the Value property of the other controls:

```
#DEFINE TXT_LOC "slider value: "  
THISFORM.Olecontrol2.Panels(2).Text = TXT_LOC + ALLTRIM(STR(THIS.value))  
THIS.Parent.spn1.Value = THIS.Value  
THIS.Parent.oleV.Value = THIS.Value
```

StatusBar Control

The StatusBar Control Properties dialog box makes setting status bar properties easy.



To open the StatusBar Control Properties dialog, choose StatusBar Control Properties from the shortcut menu of a StatusBar Control.

Use the RichText Control

File: SAMPLES\SOLUTION\OLE\RTF.SCX

This sample illustrates using a RichText control to display and edit RTF text stored in the memo field of a table.

You can set the ControlSource of a RichText control directly to a character field, but not a memo field. Instead, create a form property to mediate between the RichText control and the memo field.

► To store RTF text in a memo field

- 1 Set the ControlSource property of the RichText control to the form property.

```
THISFORM.oleRTF.ControlSource = THISFORM.cText
```

- 2 Store the memo field contents to the form property.

```
THISFORM.cText = rtf.source
```

- 3 Before moving the record pointer, store the TextRTF property of the RichText control to the memo field.

```
REPLACE rtf.Source WITH THISFORM.oleRTF.TextRTF
```

The text stored in the memo file is standard RTF format, for example:

```
{\rtf1\ansi\deff0\deftab720{\fonttbl{\f0\fswiss MS Sans Serif;}{\f1\froman\fcharset2 Symbol;}{\f2\fswiss Arial;}{\f3\fswiss Arial;}} {\colortbl\red0\green0\blue0;\red255\green0\blue0;} \deflang1033\pard\qc\plain\fs32li The RichTextBox Control \par \pard\plain\fs20
```

View Type Library Information

File: SAMPLES\SOLUTION\WINAPI\TYPELIB.SCX

The TYPELIB sample uses a Visual FoxPro class named Typelib which is stored in SAMPLES\CLASSES\TYPELIB.VCX. This container class has an ActiveX control called FoxTLib that contains a number of methods to read Type Library information from any DLL, EXE, or TLB file.

The Typelib wrapper class saves you time by handling much of the dirty work for you. There is a method called ExportTypeLib in this class which exports the contents of a specified Type Library to a text file:

```
THISFORM.typelib1.TypeLibName = THISFORM.txtFileName.Value  
THISFORM.typelib1.ExportTypeLib( )
```

If you take a look at the generated text file, you will notice that a Type Library can consist of multiple Type Infos. Each Type Info represents a specific class. When you generate a new EXE or DLL from a project using VFP, each class in the project which is marked as OLEPUBLIC will generate a separate Custom OLE Server. Note: you only have a single EXE/DLL file, however, this file can contain multiple servers (one for each OLEPUBLIC class). And VFP generates a single TLB file for the project. This Type Library file contains a separate Type Info for each OLEPUBLIC class.

Within each Type Info (OLEPUBLIC class), there are also Function descriptions. These represent all of the properties and methods of a class. For methods, the Type Library contains both parameter and return types. Some of the possible types are boolean, string, and variant. Because VFP does not support strong type memory variables, many of the types used in your custom methods will be variant.

If you take a close look at the functions exported, you will notice double entries for many of the properties. This is because users can *set* or *get* the value of this property. And some languages such as Visual Basic allow you to have code executed when one of these properties is accessed or assigned. Hence, the Type Library will represent a single property with 2 entries. If only a single entry exists, then that property is read-only. Properties and methods which are marked as Hidden or Protected will not appear in the Type Library.

