Visual FoxPro Coding Standards

Overview

Good coding standards are important in any development project, but particularly when multiple developers are working on the same project. Having standards helps ensure that the code is of a high quality, has fewer bugs, and is easily maintained.

These standards are based on years of developing Visual FoxPro applications and learning what coding techniques work best. Also, many concepts from *Code Complete* (Microsoft Press, ISBN 1-55615-484-4) by Steve McConnell have been adapted. This book is considered one of the premier guides on coding practices. You may not agree with these standards, and that's ok. These have worked well for me.

Coding Standards

Variable Usage

Do not use underscores in variable names. Mixed case should be used to improve readability. The first letter of the variable name should indicate its scope and should always be lower case. Try to avoid the use of global (PUBLIC) and private variables. Instead, use application object properties, form properties, and local variables.

l	Local
g	Global
p	Private
t	Parameter

The second letter indicates the data type

c	Character
n	Numeric
d	Date
t	DateTime
l	Logical
m	Memo
a	Array
o	Object
x	Indeterminate

Examples of valid variable names:

```
lcFirstName
tdBeginDate
```

Keep variable scoping in mind. LOCAL variables should be used as much as possible. PUBLIC variables should be avoided if at all possible. Variable declarations such as LOCAL lcMyVar should be placed at the beginning of a routine rather than spread throughout the routine. Declare all variables at the beginning of the routine rather than interspersed throughout the code and have a default value assigned to it.

Wrong way:

```
LOCAL lnMyNum
lnMyNum = 12

LOCAL lcMyString
lcMyString = "ABCD"

LOCAL lnCounter
lnCounter = 0
```

Correct way:

```
LOCAL lnMyNum, lcMyString, lnCounter

lnMyNum = 12
lcMyString = "ABCD"
lnCounter = 0
```

Object Naming Standards

The first three letters of an object name should be used to indicate the type of object.

chk	Check box
cbo	Combo box
cmd	Command button
cmg	Command Group
cnt	Container
ctl	Control
cus	Custom
edt	Edit box
frm	Form
frs	Form set
grd	Grid
grc	Grid Column
grh	Grid Column Header
img	Image
lbl	Label
lin	Line
lst	List box
olb	OLE Bound Control
ole	OLE Object such as an ActiveX Control
opg	Option Group
pag	Page
pgf	Pageframe
sep	Separator
shp	Shape
spn	Spinner
txt	Text box

tmr	Timer
tbr	Toolbar

Source Code Standards

1. Use white space liberally. It will improve readability.
2. Use tabs instead of spaces for indenting.
3. FoxPro commands and functions should be capitalized and spelled out completely. Everything else should be in mixed case. When possible, keep lines short to avoid line wrap when printed. If a line needs to be continued put “join” statements as the first character of the next line. Join statements are things like +, AND, OR, NOT, etc. Also remember that the line must be valid as if all on one line. Put a space before the semi-colon.

Examples of bad continuations:

```
lcCommand = "Today is Wednesday, October 16, 2003" + ;
    "The time is 2:00 PM"

IF ldBeginDate >= DATE() OR ;
    ldEndDate >= DATE()
```

Examples of good continuations:

```
lcCommand = "Today is Wednesday, October 16, 2003" ;
    + "The time is 2:00 PM"

IF ldBeginDate >= DATE() ;
OR ldEndDate >= DATE()
```

5. It seems that everyone does case statements differently. While there is no right or wrong way, some seem to help readability. Also, use a CASE statement instead of an IF when it appears that more options could be added at a later date, even if there are only two options at the time the code is written, or to get rid of IF, ELSE, IF constructs. Separate each CASE with a blank line. The comment for the CASE should go underneath it.

Wrong way:

```
DO CASE
* This is the comment for case 1
CASE lnCount = 1
lcRetVal = "One"
* This is the comment for case 2
CASE lnCount = 2
lcRetVal = "Two"
* This is the comment for otherwise
OTHERWISE
lcRetVal = "Another"
ENDCASE
```

Correct way:

```
DO CASE
   CASE lnCount = 1
      * This is the comment for case 1
      lcRetVal = "One"

   CASE lnCount = 2
      * This is the comment for case 2
      lcRetVal = "Two"

   OTHERWISE
     * This is the comment for otherwise
     lcRetVal = "Another"
ENDCASE
```

6. Try to avoid macro substitution. There are times when macro substitution is the only way to accomplish something. Make sure that you document why you used macro substitution and what the purpose of the code is. In most cases, macro substitution makes the code less readable. If possible use the EVALUATE() function, but again, good comments are important to aid in code readability.
7. Avoid use of STORE.
8. Use "[ ]" instead of parenthesis for arrays. It improves readability.
9. Put spaces around math operators. This improves readability.
10. Use parenthesis when calling methods or functions, even if no parameters are passed.
11. Avoid the use of m. on the left side of an equal sign. It will improve performance.
12. When doing a string concatenation, put the variable on the left side of the + sign. It will improve perfomance.

Commenting Standards

Comments are an important part of any application. Use them liberally. Comments should explain why something is being done and indicate what lines of code are affected. You should only explain how something is done if it is using complex algorithms or calculations.

Do not use comments at the end of a line with &&. Each comment should be on a line by itself.

Program, Method and Procedure Headers

Program, method and procedure headers should indicate the name of the routine the date it was originally created, the author, and a description of the procedure or method's purpose. Include a description of parameters and return values, if any. For methods, include the object hierarchy.

Example 1:

```
*********************************************************
* Method........: frmQueue.cmdNext.Click
* Description...: Displays the next item in the selected queue
* Date..........: 01-Oct-2001
* Author........: Fred Flintstone
*********************************************************
* Modification Summary
*
*********************************************************
```

Example 2:

```
********************************************************
* Function......: CalcIntrest
* Description...: Calculate the interest in dollars on the loan
* Parameters....: tnBalance: Required: The balance amount
*               : tnRate: Required: The interest rate to apply
* Returns.......: Numeric: The dollar amount of the interest
* Date..........: 01-Oct-2001
* Author........: Bullwinkle J. Moose
********************************************************
* Modification Summary
*
********************************************************
```

Commenting Modifications

It is important when making modifications to know what modifications were made, and why you made them. The Modification Summary section in the header should explain the why of a modification, when it was made, and who made it. At each place in the code where the modification was made, you should comment out the old code and indicate what new code was added. Each modification should be numbered. Changed code can be removed after one year, but the summary comments in the header should always remain.

Example:

```
********************************************************
* Modification Summary
*
* /01     05-Oct-2001    George Jetson
*         Changed interest calculation to include a date range factor.
* /02     10-Oct-2001    Tennessee Tuxedo
*         1. Added code to handle interest on widgets. The calculation is
different.
*         2. Changed the return value from numeric to character.
********************************************************

*/01 lcNote = "This is the old line commented out"
*/01
lcNote = "This is the new line of code."



*/02-1 lnInterest = a * b
*/02-1 lnInterest = lnInterest / 43

*/02-1 - Begin - Multiple lines are being added, so indicate they start here
IF UPPER(tcIntType) = "WIDGETS"
   lnInterest = a * c
ELSE
   lnInterest = a * b
   lnInterest = lnInterest / 43
ENDIF
*/02-1 - End
```

Inline comments should be indented at the same level as the code.

This example is wrong:

```
IF lnTotalInterest = 0
*/01 - Begin
   FOR lnCount = 1 TO lnTotal
      lnTotalInterest = lnTotalInterest + laLoans[lnCount, 2]
   ENDFOR
*/02 - End
ENDIF
```

This example is correct:

```
IF lnTotalInterest = 0
   */01 - Begin
   FOR lnCount = 1 TO lnTotal
      lnTotalInterest = lnTotalInterest + laLoans[lnCount, 2]
   ENDFOR
   */02 - End
ENDIF
```

When commenting out a continued line, comment each physical line.

Incorrect:

```
*/01 lcString = "This string contains lots of text because it is an " ;
+ "example of a really long line"
```

Correct:

```
*/01 lcString = "This string contains lots of text because it is an " ;
*/01 + "example of a really long line"
```

User Interface Standards

General UI Standards

When possible, the user interface should comply with the Windows Standards guidelines as described in the book "Microsoft Windows User Experience", Microsoft Press, ISBN 0-7356-0566-1. This book is also available online at http://msdn.microsoft.com/library/en-us/dnwue/html/welcome.asp. Always keep the user in mind. The easier the function can be for the user, the better the application. This may mean that the code behind the function is more complex.

Data Entry Forms

1. All input capable fields should use Select On Entry.
2. Numeric Fields should be formatted with commas and negative signs where appropriate.
3. If a field is not editable, set the TabStop property to .F. Set the ReadOnly or the disabled property to .T. where appropriate. ReadOnly is used when a field is never editable. Disabled is used when the field is editable when a specific condition evaluates to .T.
4. Use the status bar to display a message to the user that indicates the purpose of the field.

5. Always display a default value where applicable.
6. Disable objects when needed. This gives a visual indication to the user that the object can't be changed at that time.

Messages

1. Do not use WAIT WINDOW to display important messages to the user. It is easy for the user to miss the message. Use the MESSAGEBOX instead. Always include the appropriate icon on the message box. Avoid using the TIMEOUT parameter as the user could miss important information.
2. Use the status bar to display a help prompt for the current object, whether on a form or on the menu.

Form objects

1. *Text box*: A textbox is the most common control used. It can contain character, numeric, or date values.
2. *Check box*: A checkbox is a single control that is set ON or OFF. Check boxes are typically used to indicate a Yes or No status.
3. *Command button*: A command button is used to initiate an action. The most common are OK and Cancel. Command buttons are sometimes grouped together into a command group. Try to avoid using command groups. Individual buttons are easier to maintain.
4. *Option button*: An option button is sometimes called a radio button. It is used to indicate that the user can choose one option from a group of options. Option buttons are frequently grouped into an option group. This makes the selection process easier to code. It is recommended that the option group be used instead of individual option buttons. Option groups should be arranged vertically rather than horizontally.
5. *Drop-down list*: A drop down list allows the user to make one choice from a list of several objects, much like an option button. However, the drop-down list requires less screen real estate.
6. *Combo box*: Combo boxes are identified by a text box, with a drop down arrow separated from it. It is called a combo box because it is a combination of a text box and a drop-down list. The user can type in a new value or pick on from the list.
7. *List box*: A list box can be used to select one or many objects from a list.
8. *Spinner*: A spinner looks like a text box with up and down arrows. It is normally used for numeric data. The number entered is either increased or decreased by clicking on the arrows or the user can enter a specific value.
9. *Edit Box*: An edit box behaves much like a text box, but is normally used for memo fields. An edit box can have scroll bars and word wrap.