Visual Basic Coding Conventions

This appendix presents a set of suggested coding conventions for Visual Basic programs.

Coding conventions are programming guidelines that focus not on the logic of the program but on its physical structure and appearance. They make the code easier to read, understand, and maintain. Coding conventions can include:

- Naming conventions for objects, variables, and procedures.
- Standardized formats for labeling and commenting code.
- Guidelines for spacing, formatting, and indenting.

In the sections that follow, each of these areas are discussed, along with examples of good usage.

Topics

Why Coding Conventions?

The main reason for using a consistent set of coding conventions is to standardize the structure and coding style of an application so that you and others can easily read and understand the code.

Object Naming Conventions

Objects should be named with a consistent prefix that makes it easy to identify the type of object. This section lists recommended conventions for controls, data access objects, and menus.

Constant and Variable Naming Conventions

This topic lists recommended conventions for constants and variables supported by Visual Basic. It also discusses the issues of identifying data type and scope.

Structured Coding Conventions

In addition to naming conventions, structured coding conventions, such as code commenting and consistent indenting, can greatly improve code readability. This topic discusses standards for these areas.

Why Coding Conventions?

The main reason for using a consistent set of coding conventions is to standardize the structure and coding style of an application so that you and others can easily read and understand the code.

Good coding conventions result in precise, readable, and unambiguous source code that is consistent with other language conventions and as intuitive as possible.

Minimal Coding Conventions

A general-purpose set of coding conventions should define the minimal requirements necessary to accomplish the purposes discussed above, leaving the programmer free to create the program's logic and functional flow.

The object is to make the program easy to read and understand without cramping the programmer's natural creativity with excessive constraints and arbitrary restrictions.

To this end, the conventions suggested in this appendix are brief and suggestive. They do not list every possible object or control, nor do they specify every type of informational comment that could be valuable. Depending on your project and your organization's specific needs, you may wish to extend these guidelines to include additional elements, such as:

- Conventions for specific objects and components developed in-house or purchased from third-party vendors.
- Variables that describe your organization's business activities or facilities.
- Any other elements that your project or enterprise considers important for clarity and readability.

For more information For information about restrictions on naming procedures, variables, and constants, see "Code Basics" in "Programming Fundamentals."

Object Naming Conventions

Objects should be named with a consistent prefix that makes it easy to identify the type of object. Recommended conventions for some of the objects supported by Visual Basic are listed below.

Suggested Prefixes for Controls

Control type	prefix	Example
3D Panel	pnl	pnlGroup

ADO Data	ado	adoBiblio
Animated button	ani	aniMailBox
Check box	chk	chkReadOnly
Combo box, drop-down list box	cbo	cboEnglish
Command button	cmd	cmdExit
Common dialog	dlg	dlgFileOpen
Communications	com	comFax
Control (used within procedures when the specific type is unknown)	ctr	ctrCurrent
Data	dat	datBiblio
Data-bound combo box	dbcbo	dbcboLanguage
Data-bound grid	dbgrd	dbgrdQueryResult
Data-bound list box	dblst	dblstJobType
Data combo	dbc	dbcAuthor
Data grid	dgd	dgdTitles
Data list	dbl	dblPublisher
Data repeater	drp	drpLocation
Date picker	dtp	dtpPublished
Directory list box	dir	dirSource
Drive list box	drv	drvTarget
File list box	fil	filSource
Flat scroll bar	fsb	fsbMove
Form	frm	frmEntry
Frame	fra	fraLanguage
Gauge	gau	gauStatus
Graph	gra	graRevenue
Grid	grd	grdPrices
Hierarchical flexgrid	flex	flexOrders
Horizontal scroll bar	hsb	hsbVolume
Image	img	imgIcon

Image combo	imgcbo	imgcboProduct
ImageList	ils	ilsAllIcons
Label	lbl	lblHelpMessage
Lightweight check box	lwchk	lwchkArchive
Lightweight combo box	lwcbo	lwcboGerman
Lightweight command button	lwcmd	lwcmdRemove
Lightweight frame	lwfra	lwfraSaveOptions
Lightweight horizontal scroll bar	lwhsb	lwhsbVolume
Lightweight list box	lwlst	lwlstCostCenters
Lightweight option button	lwopt	lwoptIncomeLevel
Lightweight text box	lwtxt	lwoptStreet
Lightweight vertical scroll bar	lwvsb	lwvsbYear
Line	lin	linVertical
List box	lst	lstPolicyCodes
ListView	lvw	lvwHeadings
MAPI message	mpm	mpmSentMessage
MAPI session	mps	mpsSession
MCI	mci	mciVideo
Menu	mnu	mnuFileOpen
Month view	mvw	mvwPeriod
MS Chart	ch	chSalesbyRegion
MS Flex grid	msg	msgClients
MS Tab	mst	mstFirst
OLE container	ole	oleWorksheet
Option button	opt	optGender
Picture box	pic	picVGA
Picture clip	clp	clpToolbar
ProgressBar	prg	prgLoadFile
Remote Data	rd	rdTitles
RichTextBox	rtf	rtfReport

Shape	shp	shpCircle
Slider	sld	sldScale
Spin	spn	spnPages
StatusBar	sta	staDateTime
SysInfo	sys	sysMonitor
TabStrip	tab	tabOptions
Text box	txt	txtLastName
Timer	tmr	tmrAlarm
Toolbar	tlb	tlbActions
TreeView	tre	treOrganization
UpDown	upd	updDirection
Vertical scroll bar	vsb	vsbRate

Suggested Prefixes for Data Access Objects (DAO)

Use the following prefixes to indicate Data Access Objects.

Database object	**Prefix**	**Example**
Container	con	conReports
Database	db	dbAccounts
DBEngine	dbe	dbeJet
Document	doc	docSalesReport
Field	fld	fldAddress
Group	grp	grpFinance
Index	ix	idxAge
Parameter	prm	prmJobCode
QueryDef	qry	qrySalesByRegion
Recordset	rec	recForecast
Relation	rel	relEmployeeDept
TableDef	tbd	tbdCustomers
User	usr	usrNew

Workspace	wsp	wspMine

Some examples:

```
Dim dbBiblio As Database
Dim recPubsInNY As Recordset, strSQLStmt As String
Const DB_READONLY = 4            ' Set constant.
'Open database.
Set dbBiblio = OpenDatabase("BIBLIO.MDB")
' Set text for the SQL statement.
strSQLStmt = "SELECT * FROM Publishers WHERE & _
   State = 'NY'"
' Create the new Recordset object.
Set recPubsInNY = db.OpenRecordset(strSQLStmt, _
   dbReadOnly)
```

Suggested Prefixes for Menus

Applications frequently use many menu controls, making it useful to have a unique set of naming conventions for these controls. Menu control prefixes should be extended beyond the initial "mnu" label by adding an additional prefix for each level of nesting, with the final menu caption at the end of the name string. The following table lists some examples.

Menu caption sequence	Menu handler name
File Open	mnuFileOpen
File Send Email	mnuFileSendEmail
File Send Fax	mnuFileSendFax
Format Character	mnuFormatCharacter
Help Contents	mnuHelpContents

When this naming convention is used, all members of a particular menu group are listed next to each other in Visual Basic's Properties window. In addition, the menu control names clearly document the menu items to which they are attached.

Choosing Prefixes for Other Controls

For controls not listed above, you should try to standardize on a unique two or three character prefix for consistency. Use more than three characters only if needed for clarity.

For derived or modified controls, for example, extend the prefixes above so that there is no confusion over which control is really being used. For third-party controls, a lower-case abbreviation for the manufacturer could be added to the prefix. For example, a control instance created from the Visual Basic Professional 3D frame could uses a prefix of fra3d to avoid confusion over which control is really being used.

Constant and Variable Naming Conventions

In addition to objects, constants and variables also require well-formed naming conventions. This section lists recommended conventions for constants and variables supported by Visual Basic. It also discusses the issues of identifying data type and scope.

Variables should always be defined with the smallest scope possible. Global (Public) variables can create enormously complex state machines and make the logic of an application extremely difficult to understand. Global variables also make the reuse and maintenance of your code much more difficult.

Variables in Visual Basic can have the following scope:

Scope	Declaration	Visible in
Procedure-level	'Private' in procedure, sub, or function	The procedure in which it is declared
Module-level	'Private' in the declarations section of a form or code module (.frm, .bas)	Every procedure in the form or code module
Global	'Public' in the declarations section of a code module (.bas)	Everywhere in the application

In a Visual Basic application, global variables should be used only when there is no other convenient way to share data between forms. When global variables must be used, it is good practice to declare them all in a single module, grouped by function. Give the module a meaningful name that indicates its purpose, such as Public.bas.

It is good coding practice to write modular code whenever possible. For example, if your application displays a dialog box, put all the controls and code required to perform the dialog's task in a single form. This helps to keep the application's code organized into useful components and minimizes its run-time overhead.

With the exception of global variables (which should not be passed), procedures and functions should operate only on objects passed to them. Global variables that are used in procedures should be identified in the declaration section at the beginning of the procedure. In addition, you should pass arguments to subs and functions using ByVal, unless you explicitly need to change the value of the passed argument.

Variable Scope Prefixes

As project size grows, so does the value of recognizing variable scope quickly. A one-letter scope prefix preceding the type prefix provides this, without greatly increasing the size of variable names.

Scope	Prefix	Example
Global	g	gstrUserName
Module-level	m	mblnCalcInProgress
Local to procedure	None	dblVelocity

A variable has global scope if it is declared Public in a standard module or a form module. A variable has *module-level* scope if declared Private in a standard module or form module, respectively.

> **Note** Consistency is crucial to productive use of this technique; the syntax checker in Visual Basic will not catch module-level variables that begin with "p."

Constants

The body of constant names should be mixed case with capitals initiating each word. Although standard Visual Basic constants do not include data type and scope information, prefixes like i, s, g, and m can be very useful in understanding the value and scope of a constant. For constant names, follow the same rules as variables. For example:

```
mintUserListMax      'Max entry limit for User list
                   '(integer value,local to module)
gstrNewLine            'New Line character
                   '(string, global to application)
```

Variables

Declaring all variables saves programming time by reducing the number of bugs caused by typos (for example, aUserNameTmp vs. sUserNameTmp vs. sUserNameTemp). On the Editor tab of the Options dialog, check the Require Variable Declaration option. The Option Explicit statement requires that you declare all the variables in your Visual Basic program.

Variables should be prefixed to indicate their data type. Optionally, especially for large programs, the prefix can be extended to indicate the scope of the variable.

Variable Data Types

Use the following prefixes to indicate a variable's data type.

Data type	Prefix	Example
Boolean	bln	blnFound
Byte	byt	bytRasterData
Collection object	col	colWidgets
Currency	cur	curRevenue
Date (Time)	dtm	dtmStart

Double	dbl	dblTolerance
Error	err	errOrderNum
Integer	int	intQuantity
Long	lng	lngDistance
Object	obj	objCurrent
Single	sng	sngAverage
String	str	strFName
User-defined type	udt	udtEmployee
Variant	vnt	vntCheckSum

Descriptive Variable and Procedure Names

The body of a variable or procedure name should use mixed case and should be as long as necessary to describe its purpose. In addition, function names should begin with a verb, such as InitNameArray or CloseDialog.

For frequently used or long terms, standard abbreviations are recommended to help keep name lengths reasonable. In general, variable names greater than 32 characters can be difficult to read on VGA displays.

When using abbreviations, make sure they are consistent throughout the entire application. Randomly switching between Cnt and Count within a project will lead to unnecessary confusion.

User-Defined Types

In a large project with many user-defined types, it is often useful to give each such type a three-character prefix of its own. If these prefixes begin with "u," they will still be easy to recognize quickly when you are working with a user-defined type. For example, “ucli” could be used as the prefix for variables of a user-defined Client type.

Structured Coding Conventions

In addition to naming conventions, structured coding conventions, such as code commenting and consistent indenting, can greatly improve code readability.

Code Commenting Conventions

All procedures and functions should begin with a brief comment describing the functional characteristics of the procedure (what it does). This description should not describe the implementation details (how it does it) because these often change over time, resulting in unnecessary comment maintenance work, or worse yet, erroneous comments. The code itself and any necessary inline comments will describe the implementation.

Arguments passed to a procedure should be described when their functions are not obvious and when the procedure expects the arguments to be in a specific range. Function return values and global variables that are changed by the procedure, especially through reference arguments, must also be described at the beginning of each procedure.

Procedure header comment blocks should include the following section headings. For examples, see the next section, "Formatting Your Code."

Section heading	Comment description
Purpose	What the procedure does (not how).
Assumptions	List of each external variable, control, open file, or other element that is not obvious.
Effects	List of each affected external variable, control, or file and the effect it has (only if this is not obvious).
Inputs	Each argument that may not be obvious. Arguments are on a separate line with inline comments.
Returns	Explanation of the values returned by functions.

Remember the following points:

- Every important variable declaration should include an inline comment describing the use of the variable being declared.
- Variables, controls, and procedures should be named clearly enough that inline commenting is only needed for complex implementation details.
- At the start of the .bas module that contains the project's Visual Basic generic constant declarations, you should include an overview that describes the application, enumerating primary data objects, procedures, algorithms, dialogs, databases, and system dependencies. Sometimes a piece of pseudocode describing the algorithm can be helpful.

Formatting Your Code

Because many programmers still use VGA displays, screen space should be conserved as much as possible while still allowing code formatting to reflect logic structure and nesting. Here are a few pointers:

- Standard, tab-based, nested blocks should be indented four spaces (the default).
- The functional overview comment of a procedure should be indented one space. The highest level statements that follow the overview comment should be indented one tab, with each nested block indented an additional tab. For example:

```
'*************************************************
' Purpose:   Locates the first occurrence of a
'            specified user in the UserList array.
' Inputs:
'   strUserList():   the list of users to be searched.
'   strTargetUser:   the name of the user to search for.
' Returns:   The index of the first occurrence of the
'            rsTargetUser in the rasUserList array.
'            If target user is not found, return -1.
'*************************************************

Function intFindUser (strUserList() As String, strTargetUser As _
   String)As Integer
   Dim i As Integer            ' Loop counter.
   Dim blnFound As Integer      ' Target found flag.
   intFindUser = -1
   i = 0
   While i <= Ubound(strUserList) and Not blnFound
      If strUserList(i) = strTargetUser Then
         blnFound = True
         intFindUser = i
      End If
      i = i + 1
   Wend
End Function
```

Grouping Constants

Variables and defined constants should be grouped by function rather than split into isolated areas or special files. Visual Basic generic constants should be grouped in a single module to separate them from application-specific declarations.

& and + Operators

Always use the **&** operator when linking strings and the **+** operator when working with numerical values. Using the **+** operator to concatenate may cause problems when operating on two variants. For example:

```
vntVar1 = "10.01"
vntVar2 = 11
vntResult = vntVar1 + vntVar2    'vntResult = 21.01
vntResult = vntVar1 & vntVar2   'vntResult = 10.0111
```

Creating Strings for MsgBox, InputBox, and SQL Queries

When creating a long string, use the underscore line-continuation character to create multiple lines of code so that you can read or debug the string easily. This technique is particularly useful when displaying a message box (MsgBox) or input box (InputBox) or when creating an SQL string. For example:

```
Dim Msg As String
Msg = "This is a paragraph that will be " _
& "in a message box. The text is" _
& " broken into several lines of code" _
& " in the source code, making it easier" _
& " for the programmer to read and debug."
MsgBox Msg

Dim QRY As String
QRY = "SELECT *" _
```

```
& " FROM Titles" _
& " WHERE [Year Published] > 1988"
TitlesQry.SQL = QRY
```
