Visual Basic Coding Standards and Guidelines

Naming Conventions

- Use mixed case for variable names and begin variable names with a lowercase letter.
- Use mixed case for subroutine, function, and type names and begin such names with an uppercase letter.
- Use only uppercase characters for named constants.
- Use multiword identifiers. In mixed case names, capitalize the first letters of the contained words in multiword identifiers to enhance readability. For example: maxElement, currentFile, and so on. In single case names, separate the words in a multiword name with underscores. For example: BUFFER_SIZE.
- Use names that describe the roles of variables, functions, subroutines, types, and constants—this generally means that names should be more than two or three characters long. For example, use length rather than l. The exception is integer loop control variables used to index arrays, which are traditionally i, j, and k.
- Declare everything explicitly and enforce this with an Option Explicit statement in every module.
- Name controls using one of the following prefixes:

Check Box	chk	Image	img
Combo Box	cmb	Label	lbl
Command Button	btn	Line	lin
Common Dialog	dlg	List Box	lst
Data	dta	Menu	mnu
Dir List Box	dir	OLE	ole
Drive List Box	drv	Option Button	opt
File List Box	fil	Picture Box	pix
Form	form	Vertical Scroll	vsb
Frame	frm	Shape	shp
Grid	grd	Text Box	txt
Horizontal Scroll	hsb	Timer	tmr

These naming conventions allow program readers to recognize many program objects at a glance. For example, DocumentType is a type, btnStart is a command button, and MEMORY_ERROR is a named constant.

Formatting

- Use standard indentation conventions for block structured languages.
- Indent 3 spaces at a time.
- Place at most one variable declaration on a single line of code. Use the rest of the line for comments.
- Use vertical white space to separate code into segments that do parts of a whole task carried out in a block.
- Use horizontal white space to reflect precedence in expressions.

Types

- Avoid the Variant and Byte types.
- Beware of the size limitations of the Integer type (-32768 to 32767)—use Long when a large integer is needed.

Expressions

- Use parentheses liberally.
- Avoid mixed types in expressions—change values to the needed types using the built-in conversion functions.
- Simplify complex Boolean expressions by factoring, and by using DeMorgan's Laws to drive negations inward.
- Use <= and < instead of >= and >.
- Avoid the Choose and Switch functions.
- Make loop termination expressions as weak as possible.

Control Structures

- Avoid the GoTo and GoSub (and hence the Return) statements.
- In Select Case statements, always have Case Else or case conditions that exhaust all possibilities.
- Use only constants in Select Case statement expressions.

Functions and Subroutines

- Avoid declaring a function or subroutine Static (thus making all its local variables static).
- Pass parameters ByVal (by value) in preference to the default (by reference).

Forms, Modules, and Access to Program Objects

Visual Basic code appears in *forms* and *modules*. Forms are containers for controls; modules contain code not associated with controls. The content of forms is dictated by the controls that appear on them, but additional code may be added to them. All functions and subroutines appearing in forms are inaccessible outside the form, so only code for local form processing should be included in a form. The content of modules is entirely under the control of the programmer. Functions and subroutines placed in modules are accessible outside the module unless declared Private. Code placed in modules should reflect program structure, with emphasis on principles of cohesion, coupling, and information hiding. In particular, place implementations of abstract data types in separate modules.

- In both forms and modules, make the scope of declarations as local as possible.
- Declare functions and subroutines not referenced outside the module in which they are declared Private.
- Declare all global constants and variables (if any) in a single module containing nothing else.

Comments

- Start each form and module with a banner comment stating the module name, purpose, writer, and notes explaining any special features. In maintenance, record the date, programmer, purpose, and description of all changes to the form or module.
- Precede each function and subroutine definition with a banner comment stating the name and purpose of the procedure, and notes explaining any side effects, references to sources, explanations of algorithms, and so on.
- In long sequences of code, break the code into cohesive blocks and precede each one with a summary of the processing carried out in the block.
- If necessary, supplement role information captured in an object's name by a comment at the object's definition or declaration.
- Explain an object's purpose at its point of definition or declaration.
- Explain non-role-based operations when they occur.
- Document unexpected side-effects in code segment comments (like header comments), at the point of declaration of the affected object and at the points where the side-effect occurs.

Error Checking

- Write error handlers and use `On Error GoTo` to invoke them for any unusual events.
- Check the return codes of functions that return error codes, and handle or propagate any errors indicated by return codes.
- Check parameters likely to be used improperly before doing any processing.

Target Metric Values

- Aim for function comment-to-code ratios of at least 0.8.
- Try to write functions and subroutines with no more than 60 NCSL.
- Try to write modules with no more than 500 NCSL.
- Do not exceed a nesting level of 7.