The Reddick VBA (RVBA) Naming Conventions, Version 6.01

The purpose of the Reddick VBA (RVBA) Naming Conventions is to provide a guideline for naming objects in the Visual Basic for Applications (VBA) language. Having conventions is valuable in any programming project. When you use them, the name of the object conveys information about the meaning of the object. These conventions attempt to provide a way of standardizing that meaning across the body of VBA programmers.

VBA is implemented to interact with a host application-for example, Microsoft Access, Microsoft Visual Basic, AutoCAD, and Visio. The RVBA conventions cover all implementations of the VBA language, regardless of the host application. Some of the tags described in this document may not necessarily have an implementation within some of the particular host programs for VBA. The word *object,* in the context of this document, refers to simple variables and VBA objects, as well as to objects made available by the VBA host program.

While I am the editor of these conventions, they are the work of many people, including Charles Simonyi, who invented the Hungarian conventions on which these are based, and Stan Leszynski, who co-authored several versions of the conventions. Many others, too numerous to mention, have also contributed to the development and distribution of these conventions, but I'd especially like to thank Paul Litwin and Ken Getz who have made substantial contributions over the years.

These conventions are intended as a guideline. If you disagree with a particular part of the conventions, simply replace that part with what you think works better. However, keep in mind that future generations of programmers may need to understand those changes, and place a comment in the header of a module indicating what changes have been made. To be concise, the conventions are presented without rationalizations for how they were derived although each of the ideas presented has a considerable history to it.

Changes to the Conventions

Some of the tags in the version of the conventions presented here have changed from previous versions. Consider all previous tags to be grandfathered into the conventions--you don't need to go back and make changes. For new development work, I leave it up to you to decide whether to use the older tags or the ones suggested here. In a few places in this document, older tags are shown in {braces}. As updates to this document are made, the current version can be found at the Xoc Software web site, http://www.xoc.net.

An Introduction to Hungarian

The RVBA conventions are based on the Hungarian conventions for constructing object names, named for the native country of the inventor, Charles Simonyi. The objective of Hungarian is to convey information about the object concisely and efficiently. Hungarian takes some getting used to, but once adopted, it quickly becomes second nature. The format of a Hungarian object name is

```
[prefixes]tag[BaseName[Suffixes]]
```

The square brackets indicate optional parts of the object name. These components have the following meanings:

Component	Meaning
Prefixes	Modify the tag to indicate additional information. Prefixes are all lowercase. They are usually picked from a standardized list of prefixes, given later in this document.

Tag	Short set of characters, usually mnemonic, that indicates the type of the object. The tag is all lowercase. It is usually selected from a standardized list of tags, given later in this document.
BaseName	One or more words that indicate what the object represents. Capitalize the first letter of each word in the BaseName.
Suffixes	Additional information about the meaning of the BaseName. Capitalize the first letter of each word in the Suffix. They are usually picked from a standardized list of suffixes, given later in this document.

Notice that the only required part of the object name is the tag. This may seem counterintuitive; you may feel that the BaseName is the most important part of the object name. However, consider a generic procedure that operates on any form. The fact that the routine operates on a form is the important thing, not what that form represents. Because the routine may operate on forms of many different types, you do not necessarily need a BaseName. However, if you have more than one object of a type referenced in the routine, you must have a BaseName on all but one of the object names to differentiate them. In addition, unless the routine is generic, the BaseName conveys information about the variable. In most cases, a variable should include a BaseName.

Tags

Use the techniques described in the following sections to construct tags to indicate the data type of an object.

Variable tags

Use the tags listed in Table 1 for VBA data types. You can also use a specific tag instead of *obj* for any data type defined by the host application or one of its objects. (See the section “Host Application and Component Extensions to the Conventions” later in this document.)

Table 1: Tables for VBA Variables

Tag	Object Type
bool {f, bln}	Boolean
byte {byt}	Byte
cur	Currency
date {dtm}	Date
dec	Decimal
dbl	Double
int	Integer
lng	Long
obj	Object
sng	Single

str	String
stf	String (fixed length)
var	Variant

Here are several examples:

```
lngCount
intValue
strInput
```

You should explicitly declare all variables, each on a line by itself. Do not use the old-type declaration characters, such as %, &, and $. They are extraneous if you use the naming conventions, and there is no character for some of the data types, such as Boolean. You should always explicitly declare all variables of type Variant using the *As Variant* clause, even though it is the default in VBA. For example:

```
Dim intTotal As Integer
Dim varField As Variant
Dim strName As String
```

Constructing Properties Names

Properties of a class present a particular problem: should they include the naming convention to indicate the type? To be consistent with the rest of these naming conventions, they should. However, it is permitted to have property names without the tags, especially if the class is to be made available to customers who may not be familiar with these naming conventions.

Collection Tags

You treat a collection object with a special tag. You construct the tag using the data type of the collection followed by the letter *s*. For example, if you had a collection of Longs, the tag is lngs. If it was a collection of forms, the tag for the collection is frms. Although, in theory, a collection can hold objects of different data types, in practice, each of the data types in the collection is the same. If you do want to use different data types in a collection, use the objs tag. For example:

```
intsEntries
frmsCustomerData
objsMisc
```

Constants

Constants always have a data type in VBA. Because VBA will choose this data type for you if you don't specify it, you should always specify the data type for a constant. Constants declared in the General Declarations section of a module should always have a scope keyword of Private or Public, and be prefixed by the scope prefixes *m* or *g*, respectively. A constant is indicated by appending the letter *c* to the end of the data type for the constant. For example:

```
Const intcGray As Integer = 3
Private Const mdblcPi As Double = 3.14159265358979
```

Although this technique is the recommended method of naming constants, if you are more concerned about specifying that you are dealing with constants rather than their data type, you can alternatively use the generic tag *con* instead. For example:

```
Const conPi As Double = 3.14159265358979
```

Menu Items

The names of menu items should reflect their position in the menu hierarchy. All menu items should use the tag mnu, but the BaseName should indicate where in the hierarchy the menu item falls. Use *Sep* in the BaseName to indicate a menu separator bar, followed by an ordinal. For example:

```
mnuFile (on menu bar)
mnuFileNew (on File popup menu)
mnuFileNewForm (on File New flyout menu)
mnuFileNewReport (on File New flyout menu)
mnuFileSep1 (first separator bar on file popup menu)
mnuFileSaveAs (on File popup menu)
mnuFileSep2 (second separator bar on file popup menu)
mnuFileExit (on File popup menu)
mnuEdit (on menu bar)
```

Creating Data Types

VBA gives you three ways to create new data types: enumerated types, classes, and user-defined types. In each case, you will need to invent a new tag that represents the data type that you create.

Enumerated types

Groups of constants of the *long* data type should be made an enumerated type. Invent a tag for the type, append a “c,” and then define the enumerated constants using that tag. Because the name used in the Enum line is seen in the object browser, you can add a BaseName to the tag to spell out the abbreviation indicated by the tag. For example:

```
Public Enum ervcErrorValue
    ervcInvalidType = 205
    ervcValueOutOfBounds
End Enum
```

The BaseName should be singular, so that the enumerated type should be ervcErrorValue, not ervcErrorValues. The tag that you invent for enumerated types can then be used for variables that can contain values of that type. For example:

```
Dim erv As ervcErrorValue
Private Sub Example(ByVal ervCur As ervcErrorValue)
```

While VBA only provides enumerated types of groups of the long type, you can still create groups of constants of other types. Just create a set of constant definitions using an invented tag. For example:

```
Public Const estcError205 As String = "Invalid type"
Public Const estcError206 As String = "Value out of bounds"
```

Unfortunately, because this technique doesn’t actually create a new type, you don’t get the benefit of the VBA compiler performing type checking for you. You create variables that will hold constants using a similar syntax to variables meant to hold instances of enumerated types. For example:

```
Dim estError As String
```

Tags for classes and user-defined types

A class defines a user-defined object. Because these invent a new data type, you will need to invent a new tag for the object. You can add a BaseName to the tag to spell out the abbreviation indicated by the tag. User-defined types are considered a simple class with only properties, but in all other ways are used the same as class modules. For example:

```
gphGlyph
edtEdit
Public Type grbGrabber
```

You then define variables to refer to instances of the class using the same tag: For example:

```
Dim gphNext As New gphGlyph
Dim edtCurrent as edtEdit
Dim grbHandle as grbGrabber
```

Polymorphism

In VBA, you use the *Implements* statement to derive classes from a base class. The tag for the derived class should use the same tag as the base class. The derived classes, though, should use a different BaseName from the base class. For example:

```
anmAnimal (base class)
anmZebra (derived class of anmAnimal)
anmElephant (derived class of anmAnimal)
```

This logic of naming derived classes is used with forms, which are all derived from the pre-defined Form base class and use the frm tag. If a variable is defined to be of the type of the base class, then use the tag, as usual. For example:

```
Dim anmArbitrary As anmAnimal
Dim frmNew As Form
```

On the other hand, if you define a variable as an instance of a derived class, include the complete derived class name in the variable name. For example:

```
Dim anmZebraInstance As anmZebra
Dim anmElephantExample As anmElephant
Dim frmCustomerData As frmCustomer
```

Constructing Procedures

VBA procedures require you to name various items: procedure names, parameters, and labels. These objects are described in the following sections.

Constructing Procedure Names

VBA names event procedures, and you cannot change them. You should use the capitalization defined by the system. For user-defined procedure names, capitalize the first letter of each word in the name. For example:

```
cmdOK_Click
GetTitleBarString
PerformInitialization
```

Procedures should always have a scope keyword, Public or Private, when they are declared. For example:

```
Public Function GetTitleBarString() As String
Private Sub PerformInitialization
```

Naming Parameters

You should prefix all parameters in a procedure definition with ByVal or ByRef, even though ByRef is optional and redundant. Procedure parameters are named the same as simple variables of the same type, except that arguments passed by reference use the prefix “r.” For example:

```
Public Sub TestValue(ByVal intInput As Integer, ByRef rlngOutput As Long)
Private Function GetReturnValue(ByVal strKey As String, _
    ByRef rgph As Glyph) As Boolean
```

Naming Labels

Labels are named using upper and lower case, capitalizing the first letter of each word. For example:

```
ErrorHandler:
ExitProcedure:
```

Prefixes

Prefixes modify an object tag to indicate more information about an object.

Arrays of Objects Prefix

Arrays of an object type use the prefix “a.” For example:

```
aintFontSizes
astrNames
```

Index Prefix

You indicate an index into an array by the prefix *i*, and for consistency the data type should always be a long. You may also use the index prefix to index into other enumerated objects, such as a collection of user-defined classes. For example:

```
iaintFontSizes
iastrNames
igphsGlyphCollection
```

Prefixes for Scope and Lifetime

Three levels of scope exist for each variable in VBA: Public, Private, and Local. A variable also has a lifetime of the current procedure or the lifetime of the object in which it is defined. Use the prefixes in Table 2 to indicate scope and lifetime.

Table 2: Scope prefixes

Prefix	Object Type
(none)	Local variable, procedure-level lifetime, declared with “Dim”
s	Local variable, object lifetime, declared with “Static”
m	Private (module) variable, object lifetime, declared with “Private”
g	Public (global) variable, object lifetime, declared with “Public”

You also use the “m” and “g” constants with other objects, such as constants, to indicate their scope. For example:

```
intLocalVariable
mintPrivateVariable
gintPublicVariable
mdblcPi
```

VBA allows several type declaration words for backward compatibility. The older keyword “Global” should always be replaced by “Public,” and the “Dim” keyword in the General Declarations section should be replaced by “Private.”

Other Prefixes

Table 3 lists and describes some other prefixes:

Table 3: Other commonly-used prefixes

Prefix	Object Type
c	Count of some object type
h	Handle to a Windows object
r	Parameter passed by reference

Here are some examples:

```
castrArray
hWndForm
```

Suffixes

Suffixes modify the base name of an object, indicating additional information about a variable. You'll likely create your own suffixes that are specific to your development work. Table 4 lists some generic VBA suffixes.

Table 4: Commonly-used suffixes

Suffix	Object Type
Min	The absolute first element in an array or other kind of list
First	The first element to be used in an array or list during the current operation
Last	The last element to be used in an array or list during the current operation
Lim	The upper limit of elements to be used in an array or list. Lim is not a valid index. Generally, Lim equals Last + 1
Max	The absolutely last element in an array or other kind of list
Cnt	Used with database elements to indicate that the item is a Counter. Counter fields are incremented by the system and are numbers of either type Long or type Replication Id.

Here are some examples:

```
iastrNamesMin
iastrNamesMax
iaintFontSizesFirst
igphsGlyphCollectionLast
lngCustomerIdCnt
varOrderIdCnt
```

File Names

When naming items stored on the disk, no tag is needed because the extension already gives the object type. For example:

```
Test.Frm (frmTest form)
```

```
Globals.Bas (globals module)
Glyph.Cls (gphGlyph class module)
```

Host Application and Component Extensions to the Conventions

Each host application for VBA, as well as each component that can be installed, has a set of objects it can use. This section defines tags for the objects in the various host applications and components.

Access 2000, Version 9.0 Objects

Table 5 lists Access object variable tags. Besides being used in code to refer to these object types, these same tags are used to name these kinds of objects in the form and report designers.

Table 5: Access object variable tags

Tag	Object Type
aob	AccessObject
aops	AccessObjectProperties
aop	AccessObjectProperty
app	Application
bfr	BoundObjectFrame
chk	CheckBox
cbo	ComboBox
cmd	CommandButton
ctl	Control
ctls	Controls
ocx	CustomControl
dap	DataAccessPage
dcm	DoCmd
frm	Form
fcd	FormatCondition
fcds	FormatConditions
frms	Forms
grl	GroupLevel
hyp	Hyperlink
img	Image

lbl	Label
lin	Line
lst	ListBox
bas	Module
ole	ObjectFrame
opt	OptionButton
fra	OptionGroup (frame)
brk	PageBreak
pal	PaletteButton
prps	Properties
shp	Rectangle
ref	Reference
refs	References
rpt	Report
rpts	Reports
scr	Screen
sec	Section
sfr	SubForm
srp	SubReport
tab	TabControl
txt	TextBox
tgl	ToggleButton

Some examples:

```
txtName
lblInput
```

For ActiveX custom controls, you can use the tag ocx as specified in Table 5 or more specific object tags that are listed later in this document in Tables 14 and 15. For an ActiveX control that doesn't appear in the Tables 14 or 15, you can either use ocx or invent a new tag.

DAO 3.6 Objects

DAO is the programmatic interface to the Jet database engine shared by Access, Visual Basic, and Visual C++. The tags for DAO 3.6 objects are shown in Table 6.

Table 6: DAO object tags

Tag	Object Type
cnt	Container
cnts	Containers
db	Database
dbs	Databases
dbe	DBEngine
doc	Document
docs	Documents
err	Error
errs	Errors
fld	Field
flds	Fields
grp	Group
grps	Groups
idx	Index
idxs	Indexes
prm	Parameter
prms	Parameters
pdbe	PrivDBEngine
prp	Property
prps	Properties
qry	QueryDef
qrys	QueryDefs
rst	Recordset
rsts	Recordsets
rel	Relation
rels	Relations
tbl	TableDef
tbls	TableDefs

usr	User
usrs	Users
wrk	Workspace
wrks	Workspaces

Here are some examples:

```
rstCustomers
idxPrimaryKey
```

Table 7 lists the tags used to identify types of objects in a database.

Table 7: Access Database Explorer object tags

Tag	Object Type
tbl	Table
qry	Query
frm	Form
rpt	Report
mcr	Macro
bas	Module
dap	DataAccessPage

If you wish, you can use tags that are more exact or suffixes to identify the purpose and type of a database object. If you use the suffix, use the tag given from Table 7 to indicate the type. Use either the tag or the suffix found along with the more general tag, but not both. The tags and suffixes are shown in Table 8.

Table 8: Specific object tags and suffixes for Access Database Explorer objects

Tag	Suffix	Object Type
tlkp	Lookup	Table (lookup)
qsel	(none)	Query (select)
qapp	Append	Query (append)
qxtb	XTab	Query (crosstab)
qddl	DDL	Query (DDL)
qdel	Delete	Query (delete)
qflt	Filter	Query (filter)
qlkp	Lookup	Query (lookup)
qmak	MakeTable	Query (make table)

qspt	PassThru	Query (SQL pass-through)
qtot	Totals	Query (totals)
quni	Union	Query (union)
qupd	Update	Query (update)
fdlg	Dlg	Form (dialog)
fmnu	Mnu	Form (menu)
fmsg	Msg	Form (message)
fsfr	SubForm	Form (subform)
rsrp	SubReport	Form (subreport)
mmnu	Mnu	Macro (menu)

Here are some examples:

```
tblValidNamesLookup
tlkpValidNames
fmsgError
mmnuFileMnu
```

When naming objects in a database, do not use spaces. Instead, capitalize the first letter of each word. For example, instead of Quarterly Sales Values Table, use tblQuarterlySalesValues.

There is strong debate over whether fields in a table should have tags. Whether you use them is up to you. However, if you do use them, use the tags from Table 9.

Table 9: Field tags (if you decide to use them)

Tag	Object Type
lng	Autoincrementing (either sequential or random) Long (used with the suffix Cnt)
bin	Binary
byte	Byte
cur	Currency
date	Date/time
dbl	Double
guid	Globally unique identified (GUID) used for replication AutoIncrement fields
int	Integer
lng	Long
mem	Memo
ole	OLE

sng	Single
str	Text
bool	Yes/No

Visual Basic 6.0 Objects

Table 10 shows the tags for Visual Basic 6.0 objects.

Table 10: Visual Basic 6.0 object tags

Tag	Object Type
app	App
chk	CheckBox
clp	Clipboard
cbo	ComboBox
cmd	CommandButton
ctl	Control
dat	Data
dir	DirListBox
drv	DriveListBox
fil	FileListBox
frm	Form
fra	Frame
glb	Global
hsb	HScrollBar
img	Image
lbl	Label
lics	Licenses
lin	Line
lst	ListBox
mdi	MDIForm
mnu	Menu
ole	OLE

opt	OptionButton
pic	PictureBox
prt	Printer
prp	PropertyPage
scr	Screen
shp	Shape
txt	TextBox
tmr	Timer
uctl	UserControl
udoc	UserDocument
vsb	VscrollBar

Microsoft ActiveX Data Objects 2.1 Tags

Office 2000 provides version 2.1 of the ActiveX Data Objects library. Table 11 lists the recommended tags for this version of ADO.

Note: Many of the ADO, ADOX, and JRO tags overlap with existing DAO tags. Make sure you include the object library name in all references in your code, so there's never any possibility of confusion. For example, use

```
Dim rst As ADODB.Recordset
```

or

```
Dim cat As ADOX.Catalog
```

rather than using the object types without the library name. This will not only make your code more explicit and avoid confusion about the source of the object, but will also make your code run a bit faster.

Table 11: ADO 2.1 Object tags

Tag	Object Type
cmn {cmd}	Command
cnn {cnx}	Connection
err	Error
errs	Errors
fld	Field
flds	Fields
prm	Parameter
prms	Parameters

prps	Properties
prp	Property
rst	Recordset

Microsoft ADO Ext. 2.1 for DDL and Security (ADOX) Tags

In order to support DDL and security objects within Jet database, Microsoft provides ADOX, an additional ADO library of objects. Table 12 lists tags for the ADOX objects.

Table 12: ADOX Object tags

Tag	Object Type
cat	Catalog
clms	Column
clm	Columns
cmd	Command
grp	Group
grps	Groups
idx	Index
idxs	Indexes
key	Key
keys	Keys
prc	Procedure
prcs	Procedures
prps	Properties
prp	Property
tbl	Table
tbls	Tables
usr	User
usrs	Users
vw	View
vws	Views

Microsoft Jet and Replication Objects 2.1

In order to support Jet’s replication features, ADO provides another library (JRO). Table 13 lists suggested tags for the JRO objects.

Table 13: JRO object tags

Tag	Object Type
flt	Filter
flts	Filters
jet	JetEngine
rpl	Replica

Microsoft SQL Server and Microsoft Data Engine (MSDE) Objects

Table 14 lists RVBA tags for Microsoft SQL Server and the Microsoft Data Engine (a limited-connection version of SQL Server 7) objects.

Table 14: SQL Server/MSDE object tags

Tag	Object Type
tbl	table
proc	stored procedure
trg	trigger
qry	view
dgm	database diagram
pk	primary key
fk	foreign key
idx	other (non-key) index
rul	check constraint
def	default

Microsoft Common Control Objects

Windows 95 and Windows NT have a set of common controls that are accessible from VBA. Table 15 lists the tags for objects created using these controls.

Table 15: Microsoft Common Control Object tags.

Tag	Object Type
ani	Animation

btn	Button (Toolbar)
bmn	ButtonMenu (Toolbar)
bmns	ButtonMenus (Toolbar)
bnd	Band (CoolBar)
bnds	Bands (CoolBar)
bnp	BandsPage (CoolBar)
btns	Buttons (Toolbar)
cbr	CoolBar
cbp	CoolBarPage (CoolBar)
hdr	ColumnHeader (ListView)
hdrs	ColumnHeaders (ListView)
cbi	ComboItem (ImageCombo)
cbis	ComboItems (ImageCombo)
ctls	Controls
dto	DataObject
dtf	DataObjectFiles
dtp	DTPicker
fsb	FlatScrollBar
imc	ImageCombo
iml	ImageList
lim	ListImage
lims	ListImages
lit	ListItem (ListView)
lits	ListItems (ListView)
lsi	ListSubItem (ListView)
lsis	ListSubItems (ListView)
lvw	ListView
mvw	MonthView
nod	Node (TreeView)
nods	Nodes (TreeView)

pnl	Panel (Status Bar)
pnls	Panels (Status Bar)
prb	ProgressBar
sld	Slider
sbr	StatusBar
tab	Tab (Tab Strip)
tabs	Tabs (Tab Strip)
tbs	TabStrip
tbr	Toolbar
tvw	TreeView
udn	UpDown

Other Custom Controls and Objects

Finally, Table 16 lists the tags for other commonly used custom controls and objects.

Table 16: Tags for commonly-used custom controls

Tag	Object Type
cdl	CommonDialog (Common Dialog)
dbc	DBCombo (Data Bound Combo Box)
dbg	DBGrid (Data Bound Grid)
dls	DBList (Data Bound List Box)
gau	Gauge (Gauge)
gph	Graph (Graph)
grd	Grid (Grid)
msg	MAPIMessages (Messaging API Message Control)
ses	MAPISession (Messaging API Session Control)
msk	MaskEdBox (Masked Edit Textbox)
key	MhState (Key State)
mmc	MMControl (Multimedia Control)
com	MSComm (Communication Port)
out	Outline (Outline Control)

pcl	PictureClip (Picture Clip Control)
rtf	RichTextBox (Rich Textbox)
spn	SpinButton (Spin Button)

Summary

Using a naming convention requires a considerable initial effort on your part. The payoff comes when either you or another programmer has to revisit your code later. Using the conventions given here will make your code more readable and maintainable.

Greg Reddick is the President of Xoc Software, a software development company developing programs in Visual Basic, Microsoft Access, C/C++, and for the web. He leads training seminars in Visual Basic for Application Developers Training Company (AppDev). In a previous life, he worked for four years on the Access development team at Microsoft. Greg can be reached at mailto:grr@xoc.net or from the Xoc Software web site, http://www.xoc.net.