The Reddick VBA Naming Conventions

Greg Reddick

The purpose of the Reddick VBA (RVBA) Naming Conventions is to provide a guideline for naming objects in the Microsoft 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 provide a way of standardizing what that meaning is across the programming industry.

VBA is implemented to interact with a host application—for example, Microsoft Access, Visual Basic, Microsoft Excel, and Microsoft Project. In contrast to previous versions of these conventions, the RVBA conventions cover all implementations of the VBA language, regardless of the host application. Note that some of the tags described in this article may not necessarily have an implementation within some particular host program. The word *object,* in the context of this document, refers to simple variables, as well as to objects presented in the interface of the VBA host program.

While I'm the editor of these conventions and in 1992 proposed the original conventions for Microsoft Access, they are the work of many people, including Charles Simonyi, who invented the Hungarian conventions on which these are based; Stan Leszynski, who co-authored several versions of the conventions; and Paul Litwin, for his contributions and for getting the conventions in front of the public. Many others, too numerous to mention, have also contributed to the development of these conventions.

These conventions are intended as a guideline. If you disagree with a particular part, simply replace that part with what you think works better. However, keep in mind who will see those changes and place a comment in the header of a module indicating what changes have been made. The conventions are presented without rationalizations for how they were derived; you may assume that there are good reasons for the choices that have been made. Send me any questions or comments about the conventions (see the addresses at the end of the article). Suggestions for future versions are welcome.

Changes to the conventions

These conventions first appeared in print in the charter issue of *Smart Access* in February of 1993. A significantly revised version appeared in the August 1993 issue.

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, we leave it up to you to decide whether to use the older tags or the ones suggested here.

An introduction to Hungarian

The RVBA conventions are based on the Hungarian style, named for the native country of Charles Simonyi,the inventor of this style of naming objects. 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 as follows:

[prefixes]tag[BaseName[Suffixes]]
The square brackets indicate optional parts of the object name. These components have the following meanings:

- **Prefixes**—Modify the tag to indicate additional information. Prefixes are in all lowercase letters. They are usually picked from a standardized list of prefixes, given later in this article.
- **Tag**—Short set of characters, usually mnemonic, that indicates the type of the object. The tag is in all lowercase letters. It's usually selected from a standardized list of tags, given later in this article.
- **BaseName**—One or more words that indicate what the object represents. The first letter of each word in the base name is capitalized.
- **Suffixes**—Additional information about the meaning of the BaseName. The first letter of each word in the Suffix is capitalized. They are usually picked from a standardized list of suffixes, given later in this article

Notice that the only required part of the object name is the tag. This may seem counterintuitive; you may feel that the base name 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 don't necessarily need a base name. However, if you have more than one object of a type referenced in the routine, you must have a base name on all but one of the object names to differentiate them. Also, unless the routine is generic, the base name conveys information about the variable. In most cases a variable should include a base name.

Tags

You use tags to indicate the datatype of an object, and you construct them using the techniques described in the following sections.

Variable tags

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

Table 1. *Tags for VBA variables.*

Tag	Object Type
byt	Byte
f	Boolean
int	Integer
lng	Long
sng	Single
dbl	Double
cur	Currency
dat	Date
obj	Object
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. Don't use the old type declaration characters, such as %, &, and $. They are extraneous if you use the naming conventions, and there's no character for some of the datatypes, such as Boolean. You should explicitly declare all variables of type Variant, the default, as type Variant. For example:

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

Constant tags

You should indicate generic constants by using the tag "con." If you need to differentiate one class of constants from another, you can invent a class name, such as glr (for Getz, Litwin, and Reddick), and append the letter "c" to the class—for example, glrcPi. You may want to do this if you have some specific component that has global constants and you want to ensure they don't conflict with other constants. For example:

```
conPi
glrcError205
```

Tags for user-defined types and classes

User-defined types and user-created class objects are treated the same because user-defined types are really a kind of simple user-defined class. These objects have two components: the class name that defines the structure of the class and a tag that is used for instances of that class. Choose an appropriate name for the class. For example, if you had a user-defined class that described a glyph bitmap created at runtime on a form, the class name would be glyph. The tag would be an abbreviation of glyph—perhaps gph. If you had another class that was a collection of these objects, it would use glyphs and gphs, respectively. Some host applications, such as Access, don't support class modules yet; however, you can also treat a form as a user-defined class with a user interface. For example:

```
gphGlyph
nclName
```

Collection tags

You treat a collection object with a special tag. You construct the tag using the datatype of the collection followed by the letter "s." For example, if you had a collection of Longs, the tag would be lngs. If it were a collection of user-defined types with the tag gph, the collection would be gphs. 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 tag objs. For example:

```
intsEntries
erhsHandler
bscsBaseClass
```

Constructing procedures

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

Constructing procedure names

VBA names event procedures, and you can't 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 call with ByVal or ByRef, even though ByRef is optional and redundant. Procedure arguments are named the same as simple variables of the same type, except that arguments passed by reference use the prefix "r". For example:

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

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," regardless of the datatype of the index. 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 length of the program. Use the prefixes in **Table 2** to indicate scope and lifetime.

Table 2. *Prefixes for scope and lifetime.*

Prefix	Object Type
(none)	Local variable, procedure-level lifetime
s	Local variable, program-level lifetime (static variable)
m	Private (module) variable, program-level lifetime
g	Public (global) variable, program-level lifetime

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

```
intLocalVariable
mintPrivateVariable
gintPublicVariable
mconPi
```

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 several examples:

```
cstrArray
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 isn't 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 ReplicationId.

Here are some examples:

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

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. Future versions of the conventions will include tags for other VBA hosts and components.

Access 95, version 7.0 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
app	Application
chk	CheckBox
cbo	ComboBox
cmd	CommandButton
ctl	Control
ctls	Controls
ocx	CustomControl
dcm	DoCmd
frm	Form
frms	Forms
grl	GroupLevel
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
rpt	Report
rpts	Reports
scr	Screen
sec	Section
sfr	SubForm
srp	SubReport
txt	TextBox
tgl	ToggleButton

Here are some examples:

```
txtName
lblInput
```

For OLE custom controls, you can use the tag OCX as specified in **Table 5** or more specific object tags that are listed later in this article in **Tables 11** and **12**.

DAO 3.0 objects

DAO is the programmatic interface to the Jet database engine shared by Access, VB, and VC++. The tags for DAO 3.0 objects are shown in **Table 6**.

Table 6. *DAO 3.0 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 (or qdf)	QueryDef
qrys (or qdfs)	QueryDefs
rst	Recordset
rsts	Recordsets
rel	Relation
rels	Relations
tbl (or tdf)	TableDef
tbls (or tdfs)	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

If you wish, you can use more exact tags 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, don't 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
bin	Binary
byt	Byte
guid	Globally unique identified (GUID) used for replication AutoIncrement fields
lng	Autoincrementing (either sequential or random) Long (used with the suffix Cnt)
cur	Currency
dat	Date/time
dbl	Double
int	Integer
lng	Long
mem	Memo
ole	OLE
sng	Single
str	Text
f	Yes/No

Visual Basic 4.0 objects

Table 10 shows the tags for suggested Visual Basic 4.0 objects.

Table 10. *Visual Basic 4.0 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
hsb	HScrollBar
img	Image
lbl	Label
lin	Line
lst	ListBox
mdi	MDIForm
mnu	Menu
ole	OLE
opt	OptionButton
pic	PictureBox
prt	Printer
scr	Screen
shp	Shape
txt	TextBox
tmr	Timer
vsb	VScrollBar

Microsoft common control objects

Windows 95 and Windows NT have a set of common controls that are accessible from VBA. **Table 11** lists the tags for objects created using these controls.

Table 11. *Microsoft common control object tags.*

Tag	Object Type
btn	Button (Toolbar)
btns	Buttons (Toolbar)
hdr	ColumnHeader (ListView)
hdrs	ColumnHeaders (ListView)
iml	ImageList (ImageList)
lit	ListItem (ListView)
lits	ListItems (ListView)
lvw	ListView (ListView)
nod	Node (TreeView)
nods	Nodes (TreeView)
pnl	Panel (Status Bar)
pnls	Panels (Status Bar)

prb	ProgressBar (Progress Bar)
sld	Slider (Slider)
sbr	StatusBar (Status Bar)
tab	Tab (Tab Strip)
tabs	Tabs (Tab Strip)
tbs	TabStrip (Tab Strip)
tbr	Toolbar (Toolbar)
tvw	TreeView (TreeView)

Other OLE custom controls and objects

Finally, **Table 12** lists the tags for other commonly used OLE custom controls and objects.

Table 12. *Tags for commonly-used OLE custom controls.*

Tag	Object Type
ani	AniPushButton (Animated Push Button)
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)
key	MhState (Key State)
mmc	MMControl (Multimedia Control)
com	MSComm (Communication Port)
msg	MAPIMessages (Messaging API Message Control)
msk	MaskEdBox (Masked Edit Textbox)
out	Outline (Outline Control)
pcl	PictureClip (Picture Clip Control)
rtf	RichTextBox (Rich Textbox)
ses	MAPISession (Messaging API Session Control)
spn	SpinButton (Spin Button)

Summary

Using a naming convention requires a considerable initial effort on your part. It also requires that you conform to rules specified by other parties, which is difficult for many programmers. The payoff comes when either you or another programmer has to revisit your code at a later time. Using the conventions makes your code more readable and maintainable.

Greg Reddick is the President of Gregory Reddick & Associates, a consulting company specializing in software development in Microsoft Access, VB, and C/C++. He worked for four years on the Access development team at Microsoft. He's a coauthor of the Microsoft Access 95 Developer's Handbook*, published by Sybex. He can be reached at 71501,2564 on CompuServe or 71501.2564@compuserve.com from the Internet.*