Naming Conventions for Microsoft Access

The Leszynski/Reddick Guidelines for Microsoft Access

Stan Leszynski and Greg Reddick

Revised: May 25, 1994

The authors wish to thank the individuals who submitted comments on and participated in reviews of the standard.

Stan Leszynski is president of Kwery Corp., which produces several Microsoft® Access® add-on products, including Access To Word and Kwery Control Paks. Stan also manages Leszynski Company, Inc., a consulting group active in database development, which he founded in 1982. He writes and speaks on Access regularly. Stan can be reached at (206) 644-7826 or on CompuServe® at 71151,1114.

Greg Reddick is the president of Gregory Reddick & Associates, a software consulting company specializing in developing solutions in Microsoft Windows using C/C++, Access, and Visual Basic. He worked for four years on the Access development team at Microsoft, and has co-authored two Access books. Greg can be reached at (206) 881-6879 or on CompuServe at 71501,2564.

> **Editor's Note** April, 1998. Although originally written for Microsoft Access 1.*x* and 2.*x*, and Access Basic, the naming conventions in this article still apply equally well to current and forthcoming versions of Microsoft Access and to Visual Basic for Applications. The code samples in this article may need slight updating in order to run under Visual Basic for Applications.

If you've ever inherited a project from another developer, you know how frustrating it is to try to dissect another developer's style and maintain his or her code. When developers code with a common style, however, you can minimize these frustrations. To best share ideas and knowledge, the Access development community will benefit greatly if it adopts a common programming style so that we can benefit from each other's expertise with a minimum of translation overhead. Although Microsoft uses certain naming conventions in the Microsoft® Access documentation, to date they haven't published an official or comprehensive standard. Thus, we've formulated a set of naming conventions to better serve Access users and developers.

These conventions were originally published in the Charter Issue of *Smart Access*. Since then, we've logged thousands of hours of development time and scores of comments from *Smart Access* readers and other users of the conventions, and we continue to improve the style and make it more useful.

Our naming style ties Access conventions closely to Visual Basic® conventions because we recognize that many Access developers also use or plan to use Visual Basic for some data access applications. Access and Visual Basic are becoming more similar with each released version. We've written these naming conventions so you can also use them with Visual Basic database applications.

There are two levels to the naming style. Level 1 is comprehensive, but doesn't clarify objects as explicitly as Level 2. Level 1 is suitable for beginning developers, while Level 2 is intended for more experienced developers and developers involved in complex development projects and multiple-developer environments. You should experiment and choose the level that works best for you. (Please note that not all parts of the standard have two levels.)

If you're already using any previous version of our conventions, you may need to make a few changes to

accommodate the enhancements in this revision. Because Access provides little help in renaming objects, you may wish to leave existing applications as is and apply the revised conventions to new projects, going back to existing applications as time permits.

Naming Conventions: An Overview

Our naming style is based on a method of naming called *Hungarian*, referring to the nationality of its creator, Charles Simonyi (who, incidentally, worked on the first version of Access). Hungarian was first proposed in his doctoral thesis.

Some elements of Hungarian style are used in Microsoft's Visual Basic manuals and Development Kit documentation, among others. Microsoft uses Hungarian internally, and many programmers around the world use it as well. We've adapted the Hungarian style for the Access environment.

In our Access naming style, an object name is made up of four parts: one or more prefixes, a tag, a base name, and a qualifier. The four parts are assembled as follows:

[prefixes]tag[Basename][Qualifier]

Note The brackets denote that these components are optional and aren't part of the name.

The tag is the only required component, but in almost all cases the name will have the base name component because you need to be able to distinguish two objects of the same type. (Having the tag as the only required part may seem counterintuitive, but in Access Basic you can have code that deals with a generic form passed as a parameter. In this case you'd use "frm" as the parameter name; the base name isn't required except to distinguish it from a different form object variable in the code.). Here are a few examples:

Name	Prefix	Tag	Base	Qualifier
tblCustomer		tbl	Customer	
aintPartNum	a	int	PartNum	
strCustNamePrev		str	CustName	Prev

Prefixes and tags are always lowercase so your eye goes past them to the first uppercase letter where the base name begins. This makes the names more readable. The base and qualifier components begin with an uppercase letter.

The base name succinctly describes the *object*, not its *class*. This is the name you'd likely give the object if you weren't using any particular naming style. For example, in the query name *qryPartNum*, "PartNum" is the base name; it's an abbreviation for Part Number. Object tags are short and mnemonic. Object prefixes precede some object names and tags and provide further information. For example, if an integer variable *intPartNum* is an array of part numbers, the prefix "a" for array is added to the front, as in *aintPartNum()*. Further, a variable that provides an index into the array would use the name of the array prefixed with the index prefix "i", for example, *iaintPartNum*.

Applying a naming style like this requires more effort up front, but try to imagine which of these two code samples will make more sense to you a year from now when you attemptto modify or reuse your code:

```
Z = Y(X)
```

or

```
intPart = aintPartNum(iaintPartNum)
```

Object qualifiers may follow a name and further clarify names that are similar. Continuing with our parts index example, if you kept two indexes to the array, one for the first item and one for the last, the variable *iaintPartNum* above would become two qualified variables—*iaintPartNumFirst* and *iaintPartNumLast*.

Naming Database Objects

Database objects (tables, queries, forms, reports, macros, and modules) are the most frequently referenced items in an Access application. They appear in your macro code, in your Access Basic routines, and in properties. Thus, it's important that you standardize how you name them.

Microsoft's examples in the Northwind Database and Access manuals allow for spaces in object names, but we don't use them in our style. In most database engines and programming languages, including Access Basic, a space is a delimiter character between items, it isn't a logical part of an item's name. Also, spaces in field names don't work in most other database platforms or Microsoft Windows®-based applications such as Microsoft SQL Server™ or Word for Windows. Instead, use upper and lowercase designations in names, such as *tblAccountsPayable*. If spacing is still necessary, use the underscore (_) character instead of a space to be consistent with traditionally accepted SQL syntax and with Access 2.x function naming conventions.

Tags for Database Container Objects

All database container object names in our style have tags. Adding tags to these objects may make them less readable to nondevelopers, but new users will understand their value when they're trying to discern a table from a query in the listbox for a New Report wizard or a form's Control Source property. This is because Access merges table and query names into one long list. Here are Level 1 database container object name tags:

Object	Tag	Example
Table	tbl	tblCustomer
Query	qry	qryOverAchiever
Form	frm	frmCustomer
Report	rpt	rptInsuranceValue
Macro	mcr	mcrUpdateInventory
Module	bas	basBilling

At Level 1, the only name qualifier (appended to the name) that we use for database container objects is Sub, which we place at the end of a form or report name for a subform or subreport. The form frmProductSupplier would have the related subform frmProductSupplierSub. This allows objects and their subform or subreport to sort next to each other in the database container.

Level 2 tags, shown here, provide more descriptive information.

Object	Tag	Example
Table	tbl	tblCustomer
Table (lookup)	tlkp	tlkpShipper
Query (select)	qry (or qsel)	qryOverAchiever
Query (append)	qapp	qappNewProduct
Query (crosstab)	qxtb	qxtbRegionSales
Query (data definition)	qddl	qddlAddWorkColumn
Query (delete)	qdel	qdelOldAccount
Query (form filter)	qflt	qfltSalesToday
Query (lookup)	qlkp	qlkpStatus
Query (make table)	qmak	qmakShipTo
Query (pass-through)	qspt	qsptArchiveQuantity
Query (union)	quni	quniOrderDetail
Query (update)	qupd	qupdDiscount
Form	frm	frmCustomer
Form (dialog)	fdlg	fdlgLogin
Form (menu)	fmnu	fmnuUtility
Form (message)	fmsg	fmsgWait
Form (subform)	fsub	fsubOrder
Report	rpt	rptInsuranceValue
Report (subreport)	rsub	rsubOrder
Macro	mcr	mcrUpdateInventory
Macro (for form)	m[formname]	mfrmCustomer
Macro (menu)	mmnu	mmnuEntryFormFile
Macro (for report)	m[rptname]	mrptInsuranceValue
Module	bas	basBilling

Using our Level 2 style causes objects with similar functions to sort together in the database container in large applications. Imagine that you have a database container with 100 forms in it (we do!), 30 of which are messages that display during the application. Your users now want all message forms to have red text instead of black, so you must change each of the 30 forms. Having the message forms sort together in the database container (because they've all got the same tag) saves you significant effort trying to discern which forms you need to change.

Choose your table names carefully. Since changes to the names of Access objects do not propagate through the database, it is important to name things correctly when the object is created. For example, changing the name of a table late in the development cycle requires changing all the queries, forms, reports, macros, and modules that refer to that table.

You may want to name each database object that refers to a table with the same base name as the table, using the appropriate tag to differentiate them. For example, if your table is tblCustomer, its primary form would be frmCustomer, its primary report would be rptCustomer, and the macros that drive all of the events would be mfrmCustomer and mrptCustomer. We also suggest that you not make table names plural (for example, use tblCustomer, not tblCustomers), because a table usually holds more than one record, so it's plural by implication.

Database Object Prefixes

We use four database object prefixes:

- "zz" denotes objects you've deserted but may want to keep in the database for awhile for future reference or use (for example, zzfrmPhoneList). "zz" causes the object name to sort to the bottom of the database container, where it's available but out of the way.
- "zt" denotes temporary objects (for example, ztqryTest).
- "zs" denotes system objects (for example, zstblObjects). System objects are items that are part of the development and maintenance of an application not used by end users, such as error logs, development notes, documentation routines, relationship information, and so on. (Note that "zs" is a prefix. It causes the system objects to sort toward the bottom of the database container).
- "_" denotes objects under development (for example, _mcrNewEmployee). An underscore before an object name sorts to the top of the database container to visually remind you that it needs attention. Remove the underscore when the object is ready to use and it will sort normally.

Tags for Fields

Using tags in field names is a hotly debated issue, even between the authors of this article. Greg maintains that tags in field names uniformly apply the naming style across all database elements and further document your work in Access Basic routines and form or report properties. Stan prefers that the database schema remain pure (platform- and data type-independent) for migration and connectivity to other products or platforms. He prefers that a field name remain independent of its data type.

Consider both positions, along with your unique needs, when you choose whether to apply the field name tags shown here:

Field Type	Tag	Example
Binary	bin	binInternal
Byte	byt	bytFloorNum
Counter	lng	lngPKCnt
Currency	cur	curSalary
Date/Time	dtm	dtmHireDate

Double	dbl	dblMass
Integer	int (C programmers may prefer "w")	intUnit
Long	lng (C programmers may prefer "dw")	lngPopulation
Memo	mem	memComments
Ole	ole	oleEmpPhoto
Single	sng (Some users find "sgl" more mnemonic)	sngScore
Text	str (Used as opposed to "txt" because a textbox control uses "txt". C programmers may prefer "sz")	strFirstName
Yes/No	ysn (C programmers may prefer "f")	ysnDiscounted

Notes:

- The Access engine ("Jet") supports a data type called binary but the Access user interface doesn't expose it to the user. It's still possible to get a field with the binary data type by importing or attaching certain external tables. Also, some of the system table fields use this data type.
- Internally, Access treats a counter data type as a long integer with a special property called auto-increment. Because counter fields are often referenced by foreign keys and the data type in the other table is a long, Greg uses the same tag as a long. Optionally, if you want to distinguish a counter from a long, use the qualifier Cnt at the end of the name.

Tags for Control Objects

Access forms and reports automatically assign the field name to the Control Name property when you create a new bound control. Having the control name and field name the same creates some ambiguity in the database schema and in some cases may cause errors in Access Basic code referencing both a control and a field with the same name. To resolve this situation, apply the naming style to form and report controls by inserting the appropriate tag from the list below, in front of the control name suggested by Access. For example, the control name for a field whose Control Source is LastName would be *txtLastName*.

At Level 1, we recognize that users need to know the difference between an active control and a label, but may not be concerned with the type of the control. Thus the control tags are as follows:

Object	**Tag**	**Example**
Label	lbl	lblLastName
Other types	ctl	ctlLastName

Level 1 tags provide the minimum differentiation necessary to still prove useful in functions, macros, and program documentation. For example, the control tags above allow you to differentiate between labels, which aren't modifiable at runtime, and other controls, which accept values from code and users.

Level 2 control tags denote the specific type of the control on the form or report (see table below). This makes Access Basic code and macros more explicit with respect to the properties and events of the individual control.

Object	Tag	Example
Chart (graph)	cht	chtSales
Check box	chk	chkReadOnly
Combo box	cbo	cboIndustry
Command button	cmd	cmdCancel
Frame (object)	fra	fraPhoto
Label	lbl	lblHelpMessage
Line	lin	linVertical
List box	lst	lstPolicyCode
Option button	opt	optFrench
Option group	grp	grpLanguage
Page break	brk	brkPage1
Rectangle (Visual Basic uses the term "shape")	shp	shpNamePanel
Subform/report	sub	subContact
Text box	txt	txtLoginName
Toggle button	tgl	tglForm

The only prefix for controls, "zs", appears at Level 2. It denotes system-level controls used by the form or code but not displayed to the user. Such controls usually aren't visible at run time but they may store temporary values or parameters passed to the form.

Naming Access Basic and Macro Objects

Using standardized and descriptive variable, constant, and function names greatly enhances the ability of developers to share, maintain, and jointly develop code.

Procedures and Macros

Access Basic requires that each nonprivate procedure name in a database be unique. For a function called from a property on a form in Access 1.x, construct the function name as follows:

formname_controlname_propertyname

For example:

frmEmployee_cmdAdd_Push

This tells you that this function is called from the OnPush property of the control cmdAdd on the form frmEmployee. For a property that affects the entire form, just use formname_propertyname, as in frmEmployee_Open. If two or more controls on one form execute the same code, create unique functions for each using the naming style in this section, then have each of these functions call the same private function that contains the common code.

In Access 2.x, the code for controls on a form is stored attached to the form, so the form name is implied in the function and does not need to be in the function name. Thus, the example above becomes:

cmdAdd_Click

Macro names inside a macro group also use this format. In the macro group mfrmEmployee, the macro txtName_BeforeUpdate contains the actions for the txtName control's BeforeUpdate event. For example, the txtName control on your frmEmployee form would have one of these properties, depending on whether you use modules (Access 1.x), attached code (Access 2.x), or macros to implement the task:

1.x code:BeforeUpdate....=frmEmployee_txtName_BeforeUpdate()

2.x code:BeforeUpdate....=txtName_BeforeUpdate()

Macros:BeforeUpdate...mfrmEmployee.txtName_BeforeUpdate

You should prefix procedure names in library databases with a unique set of characters to prevent their names from conflicting with any other names from attached libraries. The prefix should be in uppercase letters, followed by an underscore, and be no more than four letters. For example, we prefix all the library function names for our mail-merge utility, Access To Word, with "ATW_". Global constants and variables in a library should use the same prefix because they must also be unique across the entire database name space. Similarly, it is important to use these prefixes in Declare statements to alias all external dynamic-link library (DLL) function and procedure calls.

Tags for Access Basic Variables

Every Access Basic variable should have a type tag from the following list:

Variable Type	Tag	Example
Container	con	Dim conTables as Container
Control	ctl	Dim ctlVapor As Control
Currency	cur	Dim curSalary As Currency
Database	db	Dim dbCurrent As Database
Document	doc	Dim docRelationships as Document
Double	dbl	Dim dblPi As Double
Dynaset	dyn	Dim dynTransact As Dynaset
Flag (Y/N, T/F)	f	Dim fAbort As Integer

Field	fld	Dim fldLastName as Field
Form	frm	Dim frmGetUser As Form
Group	gru	Dim gruManagers as Group
Index	idx	Dim idxOrderId as Index
Integer	int	Dim intRetValue As Integer
Long	lng	Dim lngParam As Long
Object	obj	Dim objGraph As Object
Parameter	prm	Dim prmBeginDate as Parameter
Property	prp	Dim prpUserDefined as Property
QueryDef	qdf (or qrd)	Dim qdfPrice As QueryDef
Recordset	rec (or rst)	Dim recPeople as Recordset
Relation	rel	Dim relOrderItems as Relation
Report	rpt	Dim rptYTDSales As Report
Single	sng	Dim sngLoadFactor As Single
Snapshot	snp	Dim snpParts As Snapshot
String	str	Dim strUserName As String
Table	tbl	Dim tblVendor As Table
TableDef	tdf (or tbd)	Dim tdfBooking as TableDef
Type (user-defined)	typ	Dim typPartRecord As mtPART_RECORD
User	usr	Dim usrJoe as User
Variant	var	Dim varInput As Variant
Workspace	wrk (or wsp)	Dim wrkPrimary as Workspace
Yes/No18	ysn	Dim ysnPaid As Integer

Our style doesn't use data-type suffixes such as $ and % on variable names, because the Access and Visual Basic documentation recommends against using these suffixes.

Tags for database object variables such as the Form and Report types are the same as those used for the objects. This helps when coding, because the variable you assign an object to (for example, tblVendor) usually has the same name as the object it references (tblVendor), providing you with consistent object names when coding.

Constants and User-Defined Types

It is common practice in programming for Windows to use uppercase names for constants, but the authors

differ on how to treat constants. Stan prefers using the uppercase notation and adding a scope prefix (see below), so a global constant for a specific error might be gNO_TABLE_ERROR. Greg prefers to treat constants as typed variables without scope, for example, strNoTableError.

In the above table, we've added a variable type tag of "typ" for user-defined types, and suggest a convention that matches that of constants, because you can think of both user-defined types and user-defined constants as persistent, user-created objects. The recommendations for a user-defined data type syntax include the following:

- Use uppercase letters (or upper/lower syntax if you use that optional convention for globals).
- Use a tag of "t" in front of the type name to denote that it's a type structure.
- Use "g" and "m" prefixes to denote the scope of the type (see below).

Prefixes for Scope

Level 2 of the naming convention introduces scope prefixes for variables and constants. The scope prefix comes before any other prefixes.

- Variables declared locally with a Dim statement have no prefix.
- Variables declared locally with a Static statement are prefixed with an "s", as in "sintAccumulate".
- Variables that are declared in the Declarations section of a module (or form in Visual Basic) using a Dim statement are prefixed with an "m", as in "mcurRunningSum".
- Variables declared with global scope using a Global statement in the Declarations section have the prefix "g", as in "glngGrandTotal".
- Variables that denote parameters passed to a function (in the parentheses after the function name) have a prefix of "p", as in "pstrLastName". Alternately, we sometimes use "r" instead of "p" for values passed to a function by reference, and "v" for values passed ByVal, when both types of parameters are used in a single function declaration.

Object qualifiers follow the variable name and further differentiate it from similar names. You'll probably devise a list of qualifiers relevant to the types of applications you develop, but here are some of our common ones:

Variable Property	Qualifier	Example
Current element of set	Cur	iaintCur
First element of set	First	iaintStockFirst
Last element of set	Last	iaintStockLast
Next element of set	Next	strCustomerNext
Previous element of set	Prev	strCustomerPrev
Lower limit of range	Min	iastrNameMin
Upper limit of range	Max	iastrNameMax

Source	Src	lngBufferSrc
Destination	Dest	lngBufferDest

Access Basic Labels

For Access Basic labels, we use a qualifier on the function name to create several standard labels. For On Error GoTo statements, we use the name of the function with the qualifier _Err appended, for example:

cmdAdd_Click_Err:

Some functions also have a label for jumping forward to the end of the function, because it's more appropriate to leave a function only in one place than to scatter Exit Function statements throughout a routine. We use the Done qualifier, as in:

cmdAdd_Click_Done:

Access Basic Example

Below is an example of an Access Basic routine using the naming conventions. Note these items:

- We put a header in every function that describes, at a minimum, purpose, comments, author's name/date, last revision date and notes, and parameters passed and/or returned.
- We return to the Done routine from the Error routine to ensure that open objects are closed properly before exiting the function. The temptation to simply use Exit Function from an error handler may leave files open and locked.
- The example was originally written for Access 1.*x* and uses syntax that has been modified in 2.*x* (for example, CreateDynaset is now OpenRecordset).

```
Function EliminateNulls (ByVal vstrFieldName As String, ByVal vstrTableName As String) As Integer
' What:          Replaces Null values with unique ascending integers
'                A standardized version of a routine from NWIND and Chapter 8
' Author:        Microsoft  Created: 11/92  Last Revision: 5/25/94  By: grr/swl
' Passed in:    field name and table name
' Returns:        0/-1

    On Error GoTo EliminateNulls_Err
    Dim db As Database
    Dim dynTableSrc As Dynaset
    Dim varCounter As Variant
    Dim varCriteria As Variant

    EliminateNulls = 0
    Set db = CurrentDB()
    Set dynTableSrc = db.CreateDynaset(vstrTableName)
    varCounter = DMax(vstrFieldName, vstrTableName)
    If IsNull(varCounter) Or IsEmpty(varCounter) Then
        varCounter = 1
    Else
        varCounter = Val(varCounter) + 1
    End If
    varCriteria = vstrFieldName & " = Null"

    ' Iterate over all records in the table, throw out records with Nulls
    dynTableSrc.FindFirst varCriteria
```

```
    Do Until dynTableSrc.NoMatch
        dynTableSrc.Edit
        dynTableSrc(vstrFieldName) = varCounter
        dynTableSrc.Update
        varCounter = varCounter + 1
        dynTableSrc.FindNext varCriteria
    Loop
    EliminateNulls = -1

EliminateNulls_Done:                ' Jump here to clean up and exit
    dynTableSrc.Close
    db.Close
    On Error GoTo 0
Exit Function

EliminateNulls_Err:
    Select Case Err
        ' Handle specific errors here
    Case Else
        ' Generic error handler here
    Resume EliminateNulls_Done
    End Select
End Function
```

Putting Standards into Practice

Naming conventions never replace the judicious use of comments in your table definitions, macro code, or Access Basic routines. Naming conventions are an extension of, not a replacement for, good program-commenting techniques.

Formulating, learning, and applying a consistent naming style requires a significant initial investment of time and energy. However, you'll be amply rewarded when you return to your application a year later to do maintenance or when you share your code with others. Once you implement standardized names, you'll quickly grow to appreciate the initial effort you made.

If the entire Access community, including Microsoft, coded using one common naming style, we'd all find it easier to share information about Access. With this in mind, we submit these revised guidelines to the Access community.

This document accompanied the August 1994 issue of Smart Access Journal, published by Pinnacle Publishing, Inc. Earlier versions of these guidelines were published in the February and August 1993 issues of Smart Access.
