# Hungarian Notation Summary

## Naming Conventions

All variables, constants, data types and subprograms will be consistently named using the modified Hungarian Notation method.

Note: This section is copied from the Microsoft Visual Basic 6 Books Online

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

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 control dat datBiblio

Data-bound combo box dbcbo dbcboLanguage

Data-bound grid dbgrd dbgrdQueryResult

Data-bound list box dblst dblstJobType

Directory list box dir dirSource

Drive list box drv drvTarget

File list box fil filSource

Form frm frmEntry

Frame fra fraLanguage

Gauge gau gauStatus

Graph gra graRevenue

Grid grd grdPrices

Horizontal scroll bar hsb hsbVolume

Image img imgIcon

Key status key keyCaps

Label lbl lblHelpMessage

Line lin linVertical

List box lst lstPolicyCodes

MAPI message mpm mpmSentMessage

MAPI session mps mpsSession

MCI mci mciVideo

MDI child form mdi mdiNote

Menu mnu mnuFileOpen

MS Flex grid msg msgClients

MS Tab mst mstFirst

OLE ole oleWorksheet

Outline out outOrgChart

Pen BEdit bed bedFirstName

Pen HEdit hed hedSignature

Pen ink ink inkMap

Picture pic picVGA

Picture clip clp clpToolbar

Report rpt rptQtr1Earnings

Shape shp shpCircle

Spin spn spnPages

Text box txt txtLastName

Timer tmr tmrAlarm

UpDown upd updDirection

Vertical scroll bar vsb vsbRate

Slider sld sldScale

ImageList ils ilsAllIcons

TreeView tre treOrganization

Toolbar tlb tlbActions

TabStrip tab tabOptions

StatusBar sta staDateTime

ListView lvw lvwHeadings

ProgressBar prg prgLoadFile

RichTextBox rtf rtfReport

**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 idx 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.

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.

**Variable Naming Conventions**

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 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 upper case. 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:

mintMAXUSERLIST 'Max entry limit for User list

'(integer value,local to module)

NEWLINE 'New Line character

**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.

Variables will be prefixed in lowercase, with each remaining word starting in uppercase. This is known as Camel Casing:

Dim intNumberOfStudents As Integer

All program sections will have items appear as follows: constants, class, type definitions and variables.