Article Translations

French

Other Support Options

- Contact Microsoft
  Phone Numbers, Support Options and Pricing, Online Help, and more.

Customer Service

- For non-technical assistance with product purchases, subscriptions, online services, events, training courses, corporate sales, piracy issues, and more.

Newsgroups

- Pose a question to other users. Discussion groups and Forums about specific Microsoft products, technologies, and services.

INFO: Microsoft Consulting Services Naming Conventions for Visual Basic

View products that this article applies to.

Article ID : 110264
Last Review : January 9, 2003
Revision : 1.1

This article was previously published under Q110264

On This Page

- SUMMARY
- MORE INFORMATION
  - Setting Environment Options
  - Object Naming Conventions for Standard Objects
  - Object Naming Convention for Database Objects
  - Menu Naming Conventions
  - Naming Conventions for Other Controls
  - Third-party Controls
  - Variable and Routine Naming
  - The Body of Variable and Routine Names
  - Qualifiers on Variable and Routine Names
  - User Defined Types
  - Naming Constants
  - Variant Data Type
  - Commenting Your Code
  - Formatting Your Code
  - Operators
  - Scope
  - Third-party Controls
- APPLIES TO

SUMMARY

It is a good idea to establish naming conventions for your Visual Basic code. This article gives you the naming conventions used by Microsoft Consulting Services (MCS).

This document is a superset of the Visual Basic coding conventions found in the Visual Basic "Programmer's Guide."

NOTE: The third-party controls mentioned in this article are manufactured by vendors independent of Microsoft. Microsoft makes no warranty, implied or otherwise, regarding these controls' performance or reliability.

Back to the top

MORE INFORMATION

Naming conventions help Visual Basic programmers:

- **standardize the structure, coding style and logic of an application.**
- **create precise, readable, and unambiguous source code.**
- **be consistent with other language conventions (most importantly, the Visual Basic Programmers Guide and standard Windows C Hungarian notation).**
- **be efficient from a string size and labor standpoint, thus allowing a greater opportunity for longer and fuller object names.**

- define the minimal requirements necessary to do the above.

Back to the top

Setting Environment Options

Use Option Explicit. Declare all variables to save programming time by reducing the number of bugs caused by typos (for example, aUserNameTmp vs. sUserNameTmp vs. sUserNameTemp). In the Environment Options dialog, set Require Variable Declaration to Yes. The Option Explicit statement requires you to declare all the variables in your Visual Basic program. Save Files as ASCII Text. Save form (.FRM) and module (.BAS) files as ASCII text to facilitate the use of version control systems and minimize the damage that can be caused by disk corruption. In addition, you can:

- use your own editor
- use automated tools, such as grep
- create code generation or CASE tools for Visual Basic
- perform external analysis of your Visual Basic code

To have Visual Basic always save files as ASCII text, from the Environment Options dialog, set the Default Save As Format option to Text.

Back to the top

Object Naming Conventions for Standard Objects

The following tables define the MCS standard object name prefixes. These prefixes are consistent with those documented in the Visual Basic Programmers Guide.

Prefix	Object Type	Example
ani	Animation button	aniMailBox
bed	Pen Bedit	bedFirstName
cbo	Combo box and drop down list box	cboEnglish
chk	Checkbox	chkReadOnly
clp	Picture clip	clpToolbar
cmd (3d)	Command button (3D)	cmdOk (cmd3dOk)
com	Communications	comFax
ctr	Control (when specific type unknown)	ctrCurrent
dat	Data control	datBiblio
dir	Directory list box	dirSource
dlg	Common dialog control	dlgFileOpen
drv	Drive list box	drvTarget
fil	File list box	filSource
frm	Form	frmEntry
fra (3d)	Frame (3d)	fraStyle (fra3dStyle)
gau	Gauge	gauStatus
gpb	Group push button	gpbChannel
gra	Graph	graRevenue
grd	Grid	grdPrices
hed	Pen Hedit	hedSignature
hsb	Horizontal scroll bar	hsbVolume
img	Image	imgIcon
ink	Pen Ink	inkMap
key	Keyboard key status	keyCaps
lbl	Label	lblHelpMessage

lin	Line	linVertical
lst	List box	lstPolicyCodes
mdi	MDI child form	mdiNote
mpm	MAPI message	mpmSentMessage
mps	MAPI session	mpsSession
mci	MCI	mciVideo
mnu	Menu	mnuFileOpen
opt (3d)	Option Button (3d)	optRed (opt3dRed)
ole	OLE control	oleWorksheet
out	Outline control	outOrgChart
pic	Picture	picVGA
pnl3d	3d Panel	pnl3d
rpt	Report control	rptQtr1Earnings
shp	Shape controls	shpCircle
spn	Spin control	spnPages
txt	Text Box	txtLastName
tmr	Timer	tmrAlarm
vsb	Vertical scroll bar	vsbRate

Back to the top

Object Naming Convention for Database Objects

Prefix	Object Type	Example
db	ODBC Database	dbAccounts
ds	ODBC Dynaset object	dsSalesByRegion
fdc	Field collection	fdcCustomer
fd	Field object	fdAddress
ix	Index object	ixAge
ixc	Index collection	ixcNewAge
qd	QueryDef object	qdSalesByRegion
qry (suffix)	Query (see NOTE)	SalesByRegionQry
ss	Snapshot object	ssForecast
tb	Table object	tbCustomer
td	TableDef object	tdCustomers

NOTE: Using a suffix for queries allows each query to be sorted with its associated table in Microsoft Access dialogs (Add Table, List Tables Snapshot).

Back to the top

Menu Naming Conventions

Applications frequently use an abundance of menu controls. As a result, you need a different 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. For example:

Menu Caption Sequence	Menu Handler Name
Help.Contents	mnuHelpContents
File.Open	mnuFileOpen
Format.Character	mnuFormatCharacter

```
File.Send.Fax          mnuFileSendFax
File.Send.Email        mnuFileSendEmail
```

When this convention is used, all members of a particular menu group are listed next to each other in the object drop-down list boxes (in the code window and property window). In addition, the menu control names clearly document the menu items to which they are attached.

Back to the top

Naming Conventions for Other Controls

For new controls not listed above, try to come up with a unique three character prefix. However, it is more important to be clear than to stick to three characters.

For derivative controls, such as an enhanced list box, extend the prefixes above so that there is no confusion over which control is really being used. A lower-case abbreviation for the manufacturer would also typically 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. A command button from MicroHelp could use cmdm to differentiate it from the standard command button (cmd).

Back to the top

Third-party Controls

Each third-party control used in an application should be listed in the application's overview comment section, providing the prefix used for the control, the full name of the control, and the name of the software vendor:

```
Prefix    Control Type        Vendor
cmdm      Command Button      MicroHelp
```

Back to the top

Variable and Routine Naming

Variable and function names have the following structure: <prefix><body><qualifier><suffix>

Part	Description	Example
<prefix>	Describes the use and scope of the variable.	iGetRecordNext
<body>	Describes the variable.	iGetNameFirst
<qualifier>	Denotes a derivative of the variable.	iGetNameLast
<suffix>	The optional Visual Basic type character.	iGetRecordNext%

Prefixes:

The following tables define variable and function name prefixes that are based on Hungarian C notation for Windows. These prefixes should be used with all variables and function names. Use of old Basic suffixes (such as %, &, #, etc.) are discouraged.

Variable and Function Name Prefixes:

Prefix	Converged	Variable Use	Data Type	Suffix
b	bln	Boolean	Integer	%
c	cur	Currency - 64 bits	Currency	@
d	dbl	Double - 64 bit signed quantity	Double	#
dt	dat	Date and Time	Variant	
e	err	Error		
f	sng	Float/Single - 32 bit signed floating point	Single	!
h		Handle	Integer	%
i		Index	Integer	%
l	lng	Long - 32 bit signed quantity	Long	&
n	int	Number/Counter	Integer	%
s	str	String	String	$
u		Unsigned - 16 bit unsigned quantity	Long	&
	udt	User-defined type		
vnt	vnt	Variant	Variant	
a		Array		

NOTE: the values in the Converged column represent efforts to pull together the naming standards for Visual Basic, Visual Basic for Applications, and Access Basic. It is likely that these prefixes will become Microsoft standards at some point in the near future.

Scope and Usage Prefixes:

Prefix	Description
g	Global
m	Local to module or form
st	Static variable
(no prefix)	Non-static variable, prefix local to procedure
v	Variable passed by value (local to a routine)
r	Variable passed by reference (local to a routine)

Hungarian notation is as valuable in Visual Basic as it is in C. Although the Visual Basic type suffixes do indicate a variable's data type, they do not explain what a variable or function is used for, or how it can be accessed. Here are some examples:
iSend - Represents a count of the number of messages sent
bSend - A Boolean flag defining the success of the last Send operation
hSend - A Handle to the Comm interface
Each of these variable names tell a programmer something very different. This information is lost when the variable name is reduced to Send%. Scope prefixes such as g and m also help reduce the problem of name contention especially in multi-developer projects.

Hungarian notation is also widely used by Windows C programmers and constantly referenced in Microsoft product documentation and in industry programming books. Additionally, the bond between C programmers and programmers who use Visual Basic will become much stronger as the Visual C++ development system gains momentum. This transition will result in many Visual Basic programmers moving to C for the first time and many programmers moving frequently back and forth between both environments.

Back to the top

The Body of Variable and Routine Names

The body of a variable or routine 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.

Back to the top

Qualifiers on Variable and Routine Names

Related variables and routines are often used to manage and manipulate a common object. In these cases, use standard qualifiers to label the derivative variables and routines. Although putting the qualifier after the body of the name might seem a little awkward (as in sGetNameFirst, sGetNameLast instead of sGetFirstName, sGetLastName), this practice will help order these names together in the Visual Basic editor routine lists, making the logic and structure of the application easier to understand. The following table defines common qualifiers and their standard meaning:

Qualifier	Description (follows Body)
First	First element of a set.
Last	Last element of a set.
Next	Next element in a set.
Prev	Previous element in a set.
Cur	Current element in a set.
Min	Minimum value in a set.
Max	Maximum value in a set.
Save	Used to preserve another variable that must be reset later.
Tmp	A "scratch" variable whose scope is highly localized within the code. The value of a Tmp variable is usually only valid across a set of contiguous statements within a single procedure.
Src	Source. Frequently used in comparison and transfer routines.
Dst	Destination. Often used in conjunction with Source.

Back to the top

User Defined Types

Declare user defined types in all caps with _TYPE appended to the end of the symbol name. For example:

```
Type CUSTOMER_TYPE
    sName As String
  sState As String * 2
    lID as Long
  End Type
```

When declaring an instance variable of a user defined type, add a prefix to the variable name to reference the type. For example:

```
Dim custNew as CUSTOMER_TYPE
```

Back to the top

Naming Constants

The body of constant names should be UPPER_CASE with underscores (_) between words. Although standard Visual Basic constants do not include Hungarian 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:

```
<mnUSER_LIST_MAX   ' Max entry limit for User list (integer value,
                                ' local to module)
   gsNEW_LINE       ' New Line character string (global to entire
                                ' application)
```

Back to the top

Variant Data Type

If you know that a variable will always store data of a particular type, Visual Basic can handle that data more efficiently if you declare a variable of that type.

However, the variant data type can be extremely useful when working with databases, messages, DDE, or OLE. Many databases allow NULL as a valid value for a field. Your code needs to distinguish between NULL, 0 (zero), and "" (empty string). Many times, these types of operations can use a generic service routine that does not need to know the type of data it receives to process or pass on the data. For example:

```
Sub ConvertNulls(rvntOrg As Variant, rvntSub As Variant)
    ' If rvntOrg = Null, replace the Null with rvntSub
        If IsNull(rvntOrg) Then rvntOrg = rvntSub
                End Sub
```

The are some drawbacks, however, to using variants. Code statements that use variants can sometimes be ambiguous to the programmer. For example:

```
   vnt1 = "10.01" : vnt2 = 11 : vnt3 = "11" : vnt4 = "x4"
vntResult = vnt1 + vnt2  ' Does vntResult = 21.01 or 10.0111?
vntResult = vnt2 + vnt1  ' Does vntResult = 21.01 or 1110.01?
vntResult = vnt1 + vnt3  ' Does vntResult = 21.01 or 10.0111?
vntResult = vnt3 + vnt1  ' Does vntResult = 21.01 or 1110.01?
  vntResult = vnt2 + vnt4  ' Does vntResult = 11x4 or ERROR?
  vntResult = vnt3 + vnt4  ' Does vntResult = 11x4 or ERROR?
```

The above examples would be much less ambiguous and easier to read, debug, and maintain if the Visual Basic type conversion routines were used instead. For Example:

```
iVar1 = 5 + val(sVar2)    ' use this (explicit conversion)
vntVar1 = 5 + vntVar2     ' not this (implicit conversion)
```

Back to the top

Commenting Your Code

All procedures and functions should begin with a brief comment describing the functional characteristics of the routine (what it does). This description should not describe the implementation details (how it does it) because these often change over time, resulting in unnecessary comment maintenance work, or worse yet, erroneous comments. The code itself and any necessary in-line or local comments will describe the implementation.

Parameters passed to a routine should be described when their functions are not obvious and when the routine expects the parameters to be in a specific range. Function return values and global variables that are changed by the routine (especially through reference parameters) must also be described at the beginning of each routine.

Routine header comment blocks should look like this (see the next section "Formatting Your Code" for an example):

```
                    Section    Comment Description
-----------------------------------------------------------------------
               Purpose    What the routine does (not how).
      Inputs     Each non-obvious parameter on a separate line with
                                    in-line comments
Assumes    List of each non-obvious external variable, control, open file,
                                    and so on.
         Returns    Explanation of value returned for functions.
  Effects    List of each effected external variable, control, file, and
              so on and the affect it has (only if this is not obvious)
```

Every non-trivial variable declaration should include an in-line comment describing the use of the variable being declared.

Variables, controls, and routines should be named clearly enough that in- line commenting is only needed for complex or non-intuitive implementation details.

An overview description of the application, enumerating primary data objects, routines, algorithms, dialogs, database and file system dependencies, and so on should be included at the start of the .BAS module that contains the project's Visual Basic generic constant declarations.

NOTE: The Project window inherently describes the list of files in a project, so this overview section only needs to provide information on the most important files and modules, or the files the Project window doesn't list, such as initialization (.INI) or database files.

Back to the top

Formatting Your Code

Because many programmers still use VGA displays, screen real estate must be conserved as much as possible, while still allowing code formatting to reflect logic structure and nesting.

Standard, tab-based, block nesting indentations should be two to four spaces. More than four spaces is unnecessary and can cause statements to be hidden or accidentally truncated. Less than two spaces does not sufficiently show logic nesting. In the Microsoft Knowledge Base, we use a three-space indent. Use the Environment Options dialog to set the default tab width.

The functional overview comment of a routine should be indented one space. The highest level statements that follow the overview comment should be indented one tab, with each nested block indented an additional tab. For example:

```
'*************************************************************************
'Purpose:   Locate first occurrence of a specified user in UserList array.
'       Inputs:    rasUserList():  the list of users to be searched
'                  rsTargetUser:   the name of the user to search for
'     Returns:   the index of the first occurrence of the rsTargetUser
'           in the rasUserList array. If target user not found, return -1.
'*************************************************************************

            'VB3Line: Enter the following lines as one line
  Function iFindUser (rasUserList() As String, rsTargetUser as String) _
                                  As Integer
                    Dim i As Integer           ' loop counter
                  Dim bFound As Integer      ' target found flag
                                iFindUser = -1
                                    i = 0
                 While i <= Ubound(rasUserList) and Not bFound
                       If rasUserList(i) = rsTargetUser Then
                                  bFound = True
                                  iFindUser = i
                                    End If
                                    Wend
                                End Function
```

Variables and non-generic constants should be grouped by function rather than by being split off into isolated areas or special files. Visual Basic generic constants such as HOURGLASS should be grouped in a single module (VB_STD.BAS) to keep them separate from application-specific declarations.

Back to the top

Operators

Always use an ampersand (&) when concatenating strings, and use the plus sign (+) when working with numerical values. Using a plus sign (+) with non-numerical values, may cause problems when operating on two variants. For example:

```
                              vntVar1 = "10.01"
                                vntVar2 = 11
          vntResult = vntVar1 + vntVar2          ' vntResult =  21.01
          vntResult = vntVar1 & vntVar2          ' vntResult = 10.0111
```

Back to the top

Scope

Variables should always be defined with the smallest scope possible. Global 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	Variable Declared In:	Visibility
Procedure-level	Event procedure, sub, or function	Visible in the procedure in which it is declared
Form-level, Module-level	Declarations section of a form or code module (.FRM, .BAS)	Visible in every procedure in the form or code module
Global	Declarations section of a code module (.BAS, using Global keyword)	Always visible

In a Visual Basic application, only use global variables when there is no other convenient way to share data between forms. You may want to consider storing information in a control's Tag property, which can be accessed globally using the form.object.property syntax.

If you must use global variables, it is good practice to declare all of them in a single module and group them by function. Give the module a meaningful name that indicates its purpose, such as GLOBAL.BAS.

With the exception of global variables (which should not be passed), procedures and functions should only operate on objects that are passed to them. Global variables that are used in routines should be identified in the general comment area at the beginning of the routine. In addition, pass arguments to subs and functions using ByVal, unless you explicitly want to change the value of the passed argument.

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 runtime overhead.

Back to the top

Third-party Controls

NOTE: The products discussed below are manufactured by vendors independent of Microsoft. Microsoft makes no warranty, implied or otherwise, regarding these products' performance or reliability.

The following table lists standard third-party vendor name prefix characters to be used with control prefixes:

Vendor	Abbv
MicroHelp (VBTools)	m
Pioneer Software	p
Crescent Software	c
Sheridan Software	s
Other (Misc)	o

The following table lists standard third-party control prefixes:

Control Type	Control Name	Abbr	Vendor	Example	VBX File Name
Alarm	Alarm	almm	MicroHelp	almmAlarm	MHTI200.VBX
Animate	Animate	anim	MicroHelp	animAnimate	MHTI200.VBX
Callback	Callback	calm	MicroHelp	calmCallback	MHAD200.VBX
Combo Box	DB_Combo	cbop	Pioneer	cbopComboBox	QEVBDBF.VBX
Combo Box	SSCombo	cbos	Sheridan	cbosComboBox	SS3D2.VBX
Check Box	DB_Check	chkp	Pioneer	chkpCheckBox	QEVBDBF.VBX
Chart	Chart	chtm	MicroHelp	chtmChart	MHGR200.VBX
Clock	Clock	clkm	MicroHelp	clkmClock	MHTI200.VBX
Button	Command	cmdm	MicroHelp	cmdmCommandButton	MHEN200.VBX
Button	DB_Command	cmdp	Pioneer	cmdpCommandButton	QEVBDBF.VBX
Button (Group)	Command	cmgm	MicroHelp	cmgmBtton (multiple)	MHGR200.VBX
Button	Command	cmim	MicroHelp	cmimCommandButton (icon)	MHEN200.VBX
CardDeck	CardDeck	crdm	MicroHelp	crdmCard	MHGR200.VBX
Dice	Dice	dicm	MicroHelp	dicmDice	MHGR200.VBX
List Box (Dir)	SSDir	dirs	Sheridan	dirsDirList	SS3D2.VBX
List Box (Drv)	SSDrive	drvs	Sheridan	drvsDriveList	SS3D2.VBX
List Box (File)	File List	film	MicroHelp	filmFileList	MHEN200.VBX
List Box (File)	SSFile	fils	Sheridan	filsFileList	SS3D2.VBX
Flip	Flip	flpm	MicroHelp	flpmButton	MHEN200.VBX
Scroll Bar	Form Scroll	fsrm	MicroHelp	fsrmFormScroll	???
Gauge	Gauge	gagm	MicroHelp	gagmGauge	MHGR200.VBX
Graph	Graph	gpho	Other	gphoGraph	XYGRAPH.VBX
Grid	Q_Grid	grdp	Pioneer	grdpGrid	QEVBDBF.VBX
Scroll Bar	Horizontal Scroll Bar	hsbm	MicroHelp	hsbmScroll	MHEN200.VBX
Scroll Bar	DB_HScroll	hsbp	Pioneer	hsbpScroll	QEVBDBF.VBX
Graph	Histo	hstm	MicroHelp	hstmHistograph	MHGR200.VBX
Invisible	Invisible	invm	MicroHelp	invmInvisible	MHGR200.VBX
List Box	Icon Tag	itgm	MicroHelp	itgmListBox	MHAD200.VBX
Key State	Key State	kstm	MicroHelp	kstmKeyState	MHTI200.VBX
Label	Label (3d)	lblm	MicroHelp	lblmLabel	MHEN200.VBX
Line	Line	linm	MicroHelp	linmLine	MHGR200.VBX
List Box	DB_List	lstp	Pioneer	lstpListBox	QEVBDBF.VBX
List Box	SSList	lsts	Sheridan	lstsListBox	SS3D2.VBX
MDI Child	MDI Control	mdcm	MicroHelp	mdcmMDIChild	???
Menu	SSMenu	mnus	Sheridan	mnusMenu	SS3D3.VBX
Marque	Marque	mrqm	MicroHelp	mrqmMarque	MHTI200.VB
Picture	OddPic	odpm	MicroHelp	odpmPicture	MHGR200.VBX
Picture	Picture	picm	MicroHelp	picmPicture	MHGR200.VBX
Picture	DB_Picture	picp	Pioneer	picpPicture	QEVBDBF.VBX
Property Vwr	Property Viewer	pvrm	MicroHelp	pvrmPropertyViewer	MHPR200.VBX
Option (Group)	DB_RadioGroup	radp	Pioneer	radqRadioGroup	QEVBDBF.VBX
Slider	Slider	sldm	MicroHelp	sldmSlider	MHGR200.VBX
Button (Spin)	Spinner	spnm	MicroHelp	spnmSpinner	MHEN200.VBX
Spreadsheet	Spreadsheet	sprm	MicroHelp	sprmSpreadsheet	MHAD200.VBX
Picture	Stretcher	strm	MicroHelp	strmStretcher	MHAD200.VBX
Screen Saver	Screen Saver	svrm	MicroHelp	svrmSaver	MHTI200.VBX

Switcher	Switcher	swtm	MicroHelp	swtmSwitcher	???
List Box	Tag	tagm	MicroHelp	tagmListBox	MHEN200.VBX
Timer	Timer	tmrm	MicroHelp	tmrmTimer	MHTI200.VBX
ToolBar	ToolBar	tolm	MicroHelp	tolmToolBar	MHAD200.VBX
List Box	Tree	trem	MicroHelp	tremTree	MHEN200.VBX
Input Box	Input (Text)	txtm	MicroHelp	inpmText	MHEN200.VBX
Input Box	DB_Text	txtp	Pioneer	txtpText	QEVBDBF.VBX
Scroll Bar	Vertical Scroll Bar	vsbm	MicroHelp	vsbmScroll	MHEN200.VBX
Scroll Bar	DB_VScroll	vsbp	Pioneer	vsbpScroll	QEVBDBF.VBX

Back to the top

APPLIES TO

- Microsoft Visual Basic 4.0 Standard Edition
- Microsoft Visual Basic 4.0 Professional Edition
- Microsoft Visual Basic 4.0 Professional Edition
- Microsoft Visual Basic 4.0 16-bit Enterprise Edition
- Microsoft Visual Basic 4.0 32-Bit Enterprise Edition
- Microsoft Visual Basic 2.0 Standard Edition
- Microsoft Visual Basic 3.0 Professional Edition
- Microsoft Visual Basic 2.0 Professional Edition
- Microsoft Visual Basic 3.0 Professional Edition

Back to the top

Keywords: kbinfo kbtophit kbref kbprogramming kbdocs kb3rdparty KB110264

Back to the top