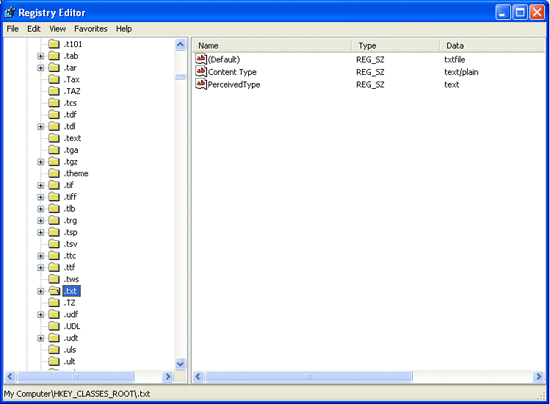
**Reading and Writing Registry Values with Visual Basic**

The Windows system Registry can be likened to a file system. Like a file system, it is organized hierarchically. Just as the file system has a root folder that contains one or more child folders, each of which in turn contains one or more child folders, and so on, the Registry has a top-level key that contains one or more child keys, each of which in turn contains one or more child keys, and so on. (Unlike the file system, though, the Registry has either five or six top-level keys, depending on the version of Windows you're using.) Just as the purpose of a folder is to store files, so the purpose of a Registry key is in part to store values.

A Registry value itself more closely resembles an item in a collection than it does a file. Like the member of a collection, a Registry value has a data type. The following are the data types the Registry supports:

Also like members of collections, most Registry values are named. Figure 1, for example, shows that two named values are stored in the HKEY\_CLASSES\_ROOT\.txt key. The first, Content Type, is a string (REG\_SZ) whose value is "txtfile". The second, PerceivedType, is also a string; its value is "text".

Figure 1 shows one additional value, indicated by the name "(Default)". A remnant of the Registry in 16-bit Windows systems, this is the default or unnamed value. As Figure 1 shows, it also is a string, or REG\_SZ, that has been assigned the value "txtfile". There can be only one unnamed value per Registry key. A Registry key, then, while it has a value that is of a definite data type, may or may not have a name.

[](http://www.windowsdevcenter.com/windows/2004/07/27/graphics/fig1.gif)  
*Figure 1. A registry key with its values (You can click on the screen shot to open a full-size view.)*

When you write a particular value to the Registry, you know precisely the name of the value, its data type, and the data you'd like to assign to it. Although this is sometimes the case when reading a particular Registry value, it also very frequently happens that you don't know the precise names of Registry values (if they have names at all), nor do you know their data types or their values. In this article, we'll show how to deal with all three of these common scenarios, as well as show you how to write values to the Registry.

**Retrieving Known Registry Values**

When you know the name and the data type of the value that you want to retrieve, you can retrieve it directly. To do this, you simply open the key containing the value, then retrieve the value itself.

For instance, suppose that you're developing a Registry utility and you intend to respect the wishes of the system administrator, who may have disabled the use of Registry tools for a particular user. In that case, you know that you want to retrieve the DisableRegistryTools value of the*HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Policies\System* key. You know thatDisableRegistryTools is a Long (a REG\_DWORD) that can take one of two possible values: 0 indicates that the user can have access to Registry tools, while 1 indicates that he or she should be prevented from directly accessing the Registry. Since both the value and the key are typically created by System Policy Editor when a user's Registry access is blocked, you must also be prepared to handle cases when either the key or the value is not present. The followingIsRegistryEditable function does that:

' Determines whether Registry tools have been disabled for the current user

Public Function IsRegistryEditable() As Boolean

Dim lValue As Long ' Variable for value

Dim sKey As String ' Key to open

Dim hKey As Long ' Handle to registry key

sKey = "Software\Microsoft\Windows\CurrentVersion\Policies\System"

If RegOpenKeyEx(HKEY\_CURRENT\_USER, sKey, 0, KEY\_READ, hKey) <> ERROR\_SUCCESS Then

' Key does not exist, return True

IsRegistryEditable = True

Else

' Determine if value exists

If RegQueryValueEx(hKey, \_

"DisableRegistryTools", \_

0, \_

REG\_DWORD, \_

lValue, \_

Len(lValue) \_

) <> ERROR\_SUCCESS Then

' value does not exist, return True

IsRegistryEditable = True

Else

' Return opposite of value (0 = Editable, 1 = Disable)

IsRegistryEditable = Not CBool(lValue)

End If

End If

End Function

Since we know what we're looking for in the Registry, we simply call RegOpenKeyEx to open the Registry key containing the value we want. If the function fails, it does not return ERROR\_SUCCESS, so we know that something has gone wrong. (For details on RegOpenKeyEx, see [Reading and Writing Registry Keys with Visual Basic](http://www.windowsdevcenter.com/pub/a/windows/2004/06/15/VB_Registry_Keys.html)). If the key is opened successfully, we call the RegQueryValueEx function, which retrieves the value. Its syntax is:

Public Declare Function RegQueryValueEx Lib "advapi32.dll" Alias "RegQueryValueExA" ( \_

ByVal hKey As Long, \_ ' Handle to open registry key

ByVal lpValueName As String, \_ ' Name of value

ByVal lpReserved As Long, \_ ' Reserved, must be 0

lpType As Long, \_ ' Registry data type constant

lpData As Any, \_ ' By reference variable for data

lpcbData As Long \_ ' Number of bytes of data written

) As Long

Two of the parameters, lpData and lpcbData, are passed to the function by reference. When the function returns, lpDatawill contain the value's data, while lpcbData will indicate the number of bytes written to lpData. This allows us to allocate a buffer that is large enough to hold the data and to know how much of the buffer must be trimmed to extract the actual data. In our case, since we're reading a long integer (a Visual Basic Long or a Registry REG\_DWORD), we don't need to be concerned about the size of our buffer and possible buffer overflows. Similarly, we've provided a literal value,Len(lValue), rather than passing a variable by reference as the lpcbData argument, since we already know how many bytes will be written when we retrieve a REG\_DWORD.

Often when retrieving values from the Registry, our code has to handle the failure to find a particular key. In the case of theIsRegsitryEditable function, for instance, the function returns True, indicating that the Registry can be accessed by user tools, if the key or value does not exist.

While this example is straightforward enough, in other cases we know which key we want to access but either aren't certain what kind of data it holds or don't know what the size of that data is. Imagine, for instance, that you're developing a utility that launches the application capable of handling a file-system object the user selects. In this case, we'll simply extract the extension from the file that the user has selected, then use its file extension to form the path to a file identification key. Once we open the file extension key, we can retrieve its unnamed value, which is the name of its file association key. From there, we can retrieve the value of the Shell\Open\Command subkey to retrieve the name of the application capable of handling the file. The following code accomplishes this:

Public Function GetAssociatedApp(sExten As String) As String

Dim sBuffer As String, sProgName As String

Dim sPath As String

Dim lBuffer As Long, lProgName As Long

Dim hKey As Long, hProgKey As Long

sBuffer = Space(20)

lBuffer = Len(sBuffer)

' Open Key

If RegOpenKeyEx(HKEY\_CLASSES\_ROOT, sExten, 0, KEY\_READ, hKey) <> ERROR\_SUCCESS Then

' Key does not exist, return null string

GetAssociatedApp = vbNullString

Else

Dim lType As Long

' Get key's unnamed value

RegQueryValueEx hKey, vbNullString, 0, 0, ByVal sBuffer, lBuffer

RegCloseKey hKey

sBuffer = Left(sBuffer, lBuffer - 1)

' Open Command key of File Association key's Open subkey

sPath = sBuffer & "\shell\open\command"

If RegOpenKeyEx(HKEY\_CLASSES\_ROOT, sPath, 0, KEY\_READ, hProgKey) = ERROR\_SUCCESS Then

' Determine data type and buffer size of key

RegQueryValueEx hProgKey, vbNullString, 0, 0, ByVal vbNull, lProgName

' Retrieve file association

sProgName = Space(lProgName + 1)

RegQueryValueEx hProgKey, vbNullString, 0, lType, ByVal sProgName, lProgName

RegCloseKey hProgKey

sProgName = Left(sProgName, lProgName - 1)

' Check if environment string is present

If lType = REG\_EXPAND\_SZ Then

Dim lProg As Long

Dim sProg As String

sProg = Space(MAX\_PATH + 1)

lProg = ExpandEnvironmentStrings(sProgName, sProg, Len(sProg))

sProgName = Left(sProg, lProg - 1)

End If

GetAssociatedApp = sProgName

Else

GetAssociatedApp = ""

End If

End If

End Function

The first part of this code is quite similar to the code in the previous example. Because we know that our code is looking up a file extension, we know that we want to open the *HKEY\_CLASSES\_ROOT\<file extension>* key and read its unnamed value. We know that that value will be a string that we can use to form a path to the next Registry key we'd like to open,*HKEY\_CLASSES\_ROOT\<file identifier>\shell\open\command*, and that we'd like to retrieve the latter key's unnamed value.

Although there is some similarity to the code in the previous example, there are also some salient differences. Whereas we were working with a named value in the previous example, here we're attempting to retrieve the unnamed value of a key. Because of this, we pass vbNullString as an argument in place of a value name. Second, because we're passing string data to and from the function, we must override Visual Basic's default method of passing strings by reference by providing the ByVal keyword. Third, because a string's length is variable, we must allocate a buffer that's sufficiently large to hold all the string data plus a null character to mark the end of the string. When the function returns, we must trim our string so that it contains only characters actually written to the buffer by the function.

Once we open the *HKEY\_CLASSES\_ROOT\<file extension>\shell\open\command* key and are about to get the name of the program responsible for handling a particular file type, we're no longer completely sure of the format of the value that we're looking for in the Registry. We don't know how long the unnamed value of the key is, nor do we know precisely what type of string it contains. Along with standard strings (of type REG\_SZ), there are also strings with embedded "macros" (REG\_EXPAND\_SZ) and string arrays (REG\_MULTI\_SZ).

To ensure that we allocate a buffer large enough to retrieve the program name, we call RegQueryValueEx twice. In the first call, we provide null values for all but one by reference argument, the number of characters written to the buffer. This preliminary function call allows us to determine how large a buffer we need to allocate in order to successfully retrieve the data. Note that the value of lpcbData includes the length not only of the data, but also of the terminating null character.

Finally, although we know in advance that we are handling string data here, we must check the precise type of the data. Along with conventional string data, Registry strings can include environmental variables (like %SystemRoot% and%WinDir%) that need to be expanded into their values. To do this, we call the ExpandEnvironmentStrings function if the result of the RegQueryValueEx function call indicates that the function has returned REG\_EXPAND\_SZ data.

**Retrieving Unknown Registry Values**

Very frequently, we're even less sure of the basic items of information that we need to access Registry values. Possibly we don't know the precise number and names of the values we want, or we don't know their precise data type, or we have no real idea of how long they are. This is frequently the case, for instance, when working with most recently used (MRU) lists, which often are stored in the Registry. Although there are a number of implementations for MRUs, most commonly the name of a file is assigned a simple, one-letter value name (a, b, c, and the like), and an MRUList named value keeps track of the entries by simply listing their names in order. For instance, the string stored to MRUList might be dcabe, indicating that the value stored to d was most recently used, followed by the value stored to c, and so on. In our case, we'll build a routine that reads the MRU list for the Windows Run dialog, which is accessed by selecting the Run option from the Start menu.

When our uncertainly about Registry values extends to most of the values of a key, as it does in this case, we can begin by calling RegQueryInfoKey to collect three useful items of information about the key's values. Its syntax is:

Public Declare Function RegQueryInfoKey Lib "advapi32.dll" Alias "RegQueryInfoKeyA" ( \_

ByVal hKey As Long, \_

ByVal lpClass As String, \_

lpcbClass As Long, \_

ByVal lpReserved As Long, \_

lpcSubKeys As Long, \_

lpcbMaxSubKeyLen As Long, \_

lpcbMaxClassLen As Long, \_

lpcValues As Long, \_ ' Number of values

lpcbMaxValueNameLen As Long, \_ ' Length of longest value name

lpcbMaxValueLen As Long, \_ ' Length of longest value data

lpcbSecurityDescriptor As Long, \_

lpftLastWriteTime As FILETIME \_

) As Long

We can substitute a null or a null string (depending on the data type) for arguments in which we're not interested. The function tells us the following:

* How many values the key contains, including the unnamed value
* The number of characters in the longest named value, not including a terminating null character
* The number of bytes of data, not including a possible terminating null character

In the case of the MRU list, calling RegQueryInfoKey once can save us from having to call RegQueryValueEx twice for each value, since we don't know the amount of data that each value contains.

In building our MRU list routine, once we know the longest value name and the size of the buffer we need to create, we can begin to read values in either of two ways. One way (which we won't use here) is to retrieve all of the values in the key by calling RegEnumValue. Its syntax is:

Public Declare Function RegEnumValue Lib "advapi32.dll" Alias "RegEnumValueA" ( \_

ByVal hKey As Long, \_ ' Handle to open key

ByVal dwIndex As Long, \_ ' Index (0 to total - 1

ByVal lpValueName As String, \_ ' name of value

lpcbValueName As Long, \_ ' kength of value name

ByVal lpReserved As Long, \_ ' reserved, must be 0

lpType As Long, \_ ' data type

lpData As Byte, \_ ' value data

lpcbData As Long \_ ' length of value data

) As Long

The major disadvantage of RegEnumValue is that it takes a byte array as the lpdata argument, and so requires that we reassemble our data from the byte array. While this is easy to do, it offers horrendous performance for keys with large numbers of values or for values with large amounts of data. A better alternative is to call RegEnumValue to get the value's name and, if necessary, its data type, and then to use this information to call RegQueryValueEx. The following code fragment illustrates this:

' Open registry key

RegOpenKeyEx HKEY\_CURRENT\_USER, sPath, 0, KEY\_READ, hKey

' Collect information on key's values

RegQueryInfoKey hKey, "", 0, 0, 0, 0, 0, nValues, lValueName, lValueData, 0, ft

' Enumerate values

For nCtr = 0 To nValues - 1

' Reinitialize buffer on each iteration

lName = lValueName + 1

sName = Space(lName)

' Get enumerated value

RegEnumValue hKey, nCtr, sName, lName, 0, lType, ByVal 0, 0

' Trim value name

sName = Left(sName, lName)

' If necessary, handle data types

ldata = lValueData + 1

sData = Space(ldata)

' Get data

RegQueryValueEx hKey, sName, 0, 0, ByVal sData, ldata

' Trim string data

sData = Left(sData, ldata)

' Process value name and data

nCtr = nCtr + 1

Next

RegCloseKey hKey

However, let's take a look at what we know about the values we want to retrieve. We know that they are all strings (of typeREG\_SZ). Once we call RegQueryInfoKey, we can determine how large a buffer must be allocated to accommodate the data. And we know the name of one value, MRUList, whose data is an ordered list of the names of the values that we want to retrieve. This means that we can omit enumerating the values and implement our routine by calling RegQueryValueExfor each key that we want to retrieve. The code appears as follows:

Public Sub ShowRunMRU()

Dim iCtr As Integer

Dim hKey As Long

Dim nValues As Long, lValueData As Long

Dim lData As Long

Dim sPath As String

Dim sList As String

Dim sMsg As String

Dim sData As String

Dim sName As String

Dim ft As FILETIME

sPath = "Software\Microsoft\Windows\CurrentVersion\Explorer\RunMRU"

' Open registry key

RegOpenKeyEx HKEY\_CURRENT\_USER, sPath, 0, KEY\_READ, hKey

' Collect information on key's values

If RegQueryInfoKey(hKey, "", 0, 0, 0, 0, 0, nValues, 0, lValueData, \_

0, ft) <> ERROR\_SUCCESS Then

sMsg = "Run MRU list not found"

Else

' initialize data buffer

lData = lValueData + 1

sData = Space(lData)

' Retrieve MRUList Value

If RegQueryValueEx(hKey, "MRUList", 0, 0, ByVal sData, lData) <> ERROR\_SUCCESS Then

sMsg = "Run MRU list not found"

Else

' Trim MRU list data

sList = Left(sData, lData - 1)

For iCtr = 1 To Len(sList)

' Get value name

sName = Mid(sList, iCtr, 1)

' Reinitialize buffer on each iteration

lData = lValueData + 1

sData = Space(lData)

' Get data

RegQueryValueEx hKey, sName, 0, 0, ByVal sData, lData

' Trim string data

sData = Left(sData, lData - 1)

sMsg = sMsg & sData & vbCrLf

Next

End If

RegCloseKey hKey

End If

' Display MRU list

MsgBox sMsg, vbOKOnly, "Run MRU List"

End Sub

**Setting a Registry Value**

To write a value to the Registry, you call the RegSetValueEx function. Its syntax is:

Public Declare Function RegSetValueEx Lib "advapi32.dll" Alias "RegSetValueExA" ( \_

ByVal hKey As Long, \_ ' Handle to registry key

ByVal lpValueName As String, \_ ' Name of value

ByVal Reserved As Long, \_ ' Reserved, must be 0

ByVal dwType As Long, \_ ' Registry data type constant

lpData As Any, \_ ' Value's data

ByVal cbData As Long \_ ' Length in bytes of data

) As Long

The function is fairly straightforward, as far as Registry functions go. A few comments are in order, though:

* If you are writing an unnamed value to the Registry, lpValueName should be a null string. You also have to be careful about platform: On Windows 95/98/ME, the unnamed value must be of type REG\_SZ. On Windows NT/2000/XP, it can be of any valid type.
* Strings (REG\_SZ and REG\_EXPAND\_SZ) must be null-terminated. In string arrays (REG\_MULTI\_SZ), each array element must be null-terminated, and the array as a whole must be terminated with two nulls (one for the last element of the array, and one for the array itself). Visual Basic, however, takes care of adding the final terminating null when we pass the string by value.
* cbData, the count of bytes in the data, must include the terminating null character or characters for Registry string data.
* The function will create a new value if one does not already exist. If it does exist, it will overwrite the old value with the new one. Because of the possibility of accidentally overwriting data, in some cases you may want to callRegQueryValueEx to check whether a Registry value exists.

The following code illustrates the use of the major Registry value types used by developers.

' Defined to save as REG\_BINARY data

Public Type AppFontInfo

Name As String \* 15

Size As Integer

Bold As Boolean

Italic As Boolean

End Type

Public Sub WriteRegistryValues()

Dim hKey As Long

Dim lDisp As Long

Dim sPath As String

Dim sa As SECURITY\_ATTRIBUTES

Dim sValueName As String

sPath = "Software\MyCompany\MyApp\Settings"

If RegCreateKeyEx(HKEY\_LOCAL\_MACHINE, sPath, 0, vbNullString, REG\_OPTION\_NON\_VOLATILE, \_

KEY\_ALL\_ACCESS, sa, hKey, lDisp) <> ERROR\_SUCCESS Then

MsgBox "Unable to create registry key."

Exit Sub

End If

' Create a REG\_DWORD value

Dim dword As Long

sValueName = "MRUItems"

dword = 5

RegSetValueEx hKey, sValueName, 0, REG\_DWORD, dword, Len(dword)

' Create a REG\_BINARY value

Dim fi As AppFontInfo

fi.Name = "Arial"

fi.Size = 12

fi.Bold = False

fi.Italic = False

sValueName = "DefaultFont"

RegSetValueEx hKey, sValueName, 0, REG\_BINARY, fi, Len(fi)

' Create a REG\_SZ value

Dim sValue As String

sValue = "GraphicsViewer"

sValueName = "AddIn"

RegSetValueEx hKey, sValueName, 0, REG\_SZ, ByVal sValue, Len(sValue) + 1

' Create a REG\_EXPAND\_SZ value

Dim sLibPath As String

sLibPath = "%windir%\twain\_32.dll"

sValueName = "Library"

RegSetValueEx hKey, sValueName, 0, REG\_EXPAND\_SZ, ByVal sLibPath, Len(sLibPath) + 1

'Create a REG\_MULTI\_SZ value

Dim ColorString As String

Dim Color As Variant, Colors As Variant

Colors = Array("Yellow", "Black", "Purple")

' Form array string

For Each Color In Colors

ColorString = ColorString & Color & Chr(0)

Next

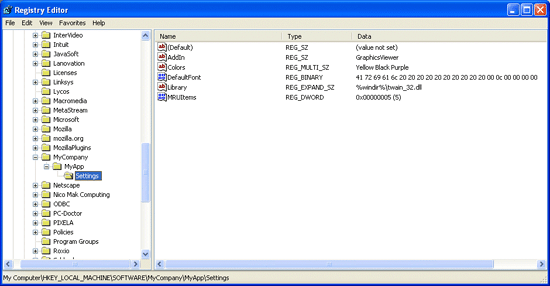
sValueName = "Colors"

RegSetValueEx hKey, sValueName, 0, REG\_MULTI\_SZ, ByVal ColorString, Len(ColorString) + 1

' Close registry key

RegCloseKey hKey

End Sub

[](http://www.windowsdevcenter.com/windows/2004/07/27/graphics/fig2.gif)  
*Figure 2. Registry after running WriteRegistryValues procedure (You can click on the screen shot to open a full-size view.)*

[*Ron Petrusha*](http://www.windowsdevcenter.com/pub/au/255)*is the author and coauthor of many books, including "VBScript in a Nutshell."*