Use the Registry to Save Information

by Chris Barlow

Incorporate the full power of the registry—a system database that allows multilevel keys and values.

One thing you can do to make a program more user-friendly is to have it save certain information upon exiting. This way, settings such as the user name, default database, and window position will be used the next time the app is run. For example, if you increase the font size in the Visual Basic editor (from the Options menu item under the Tools menu), the larger font will kick in the next time you run Visual Basic.

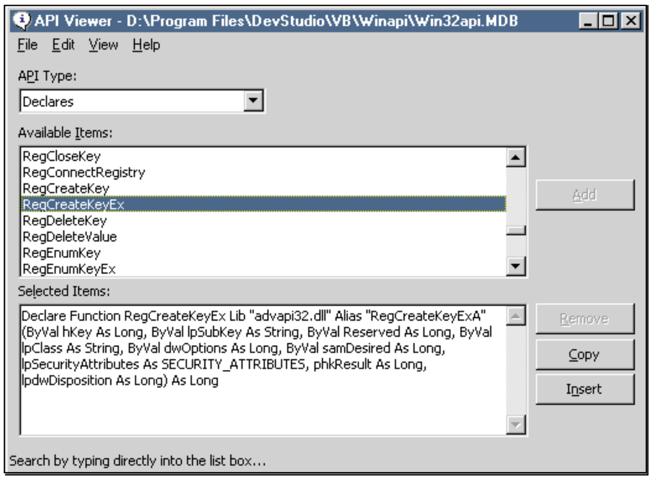
In 16-bit Windows, you save this information through INI files that contain lines of name-value pairs within bracketed sections (take a look at your WIN.INI for an example of this format). 32-bit Windows replaces this two-level structure with the registry—a system database that allows multilevel keys and values. For example, the Visual Basic font size information for the Visual Basic editor is saved in the registry under the five-level key, *Software\Microsoft\VBA\Microsoft Visual Basic\FontHeight*.

Although Visual Basic includes four statements to read and write to the registry (GetSetting, SaveSetting, DeleteSetting, and GetAllSettings), these statements are limited to a two-level structure within a special registry key for backward compatibility: *Software\VB and VBA Program Settings*. When you use one of these statements, all your data is saved within this special key. Wouldn't it be nice to use the full power of the multilevel registry from your Visual Basic programs?

At my company, we couldn't be limited to using the *VB and VBA Program Settings* key for the persistent information stored for users of our ObjectBank VB5 app. We needed to save multiple user profiles for database access. The multilevel key features of the registry let us save these profiles within a four-level key structure—the *SunOpTech\ObjectBank\ObjectTeller\Profiles* key. I wrote a registry API class that replaces Visual Basic's four intrinsic statements with more powerful counterparts. You can download this class from the free, Registered Level of The Development Exchange (see the Code Online box at the end of the column for details). In this column, I'll show you how to replace the GetSetting, SaveSetting, and DeleteSetting statements with direct calls to the registry.

Caution: *This class module will write to your registry! If you make an error in the code or delete the wrong registry value or key, you might damage your applications or operating system and need to restore them. Before working with this class module, be sure you have a current backup.*

Start a new Visual Basic project and insert a class module. Press F4 to display the Properties window, and change the class's name to CRegAPI. You need to declare the registry API functions that this class will call. Visual Basic includes a special API Viewer add-in that contains all the function declarations, constants, and types for the Win32 API. To load this add-in, select the Add-In Manager from the



```
Declare Function RegCreateKeyEx Lib "advapi32.dll" Alias "RegCreateKeyExA"
(ByVal hKey As Long, ByVal lpSubKey As String, ByVal Reserved As Long, ByVal
lpClass As String, ByVal dwOptions As Long, ByVal samDesired As Long,
lpSecurityAttributes As SECURITY_ATTRIBUTES, phkResult As Long,
lpdwDisposition As Long) As Long
```

FIGURE 1 ***Enjoy the View.*** *The Visual Basic API Viewer add-in provides the easiest way to find a Win32 function declaration and paste it directly into your application.*

Chris Barlow is president and CEO of SunOpTech, a developer of manufacturing decision-support and supply chain applications, including the ObjectBank, ObjectOrder, and ObjectJob Systems. He holds two U.S. Patents related to software for decentralized distributed asynchronous object-oriented and scheduling systems. Chris, who holds degrees from Harvard Business School and Dartmouth College, is a frequent speaker at VBITS, Tech•Ed, and DevDays and has been featured in two Microsoft videos. Reach Chris at ChrisB@SunOpTech.com.

Add-Ins menu. You can use this add-in to locate the registry functions and paste them into your class module (see Figure 1). You'll need quite a few functions for all the features of the class. For the complete registry API function declarations, download Listing 1 from the free, Registered Level of The Development Exchange.

As you design this class to replace Visual Basic's intrinsic functions, program each method's default behavior to mimic the function being replaced. This lets you easily substitute the methods of this class for the intrinsic functions in your applications. Because the intrinsic functions save all information to the *Software\VB and VBA Program Settings* within the *Hkey_Current_User* registry root, your class should default to these keys while providing optional parameters to access other keys. Create two properties, KeyPrefix and Root, and set them in the Class_Initialize event to these default values. This is also a good place to create enumerated constants for the seven registry roots supported by Windows 95 and Windows NT:

```
Public KeyPrefix As String
Public Root As Long
Public Enum RegistryRoots
   Hkey_Classes_Root = &H80000000
   Hkey_Current_User = &H80000001
   Hkey_Local_Machine = &H80000002
   Hkey_Users = &H80000003
   Hkey_Performance_Data = &H80000004
   Hkey_Current_Config = &H80000005
   Hkey_Dyn_Data = &H80000006
End Enum

Private Sub Class_Initialize()
KeyPrefix = "Software\VB and " & _
   "VBA Program Settings\"
Root = Hkey_Current_User
End Sub
```

Next, create the class's first three methods to mimic the intrinsic statements. For now, simply write the procedure definitions—you can fill in the code for these methods later. Notice that GetSetting is a function because it returns the value from the registry, but SaveSetting and DeleteSetting are subs. The GetSetting method has an optional parameter for the default value that will be returned if the key is not found in the registry:

```
Public Function GetSetting(appname$, _
   section$, key$, Optional default)
End Function

Public Sub SaveSetting(appname$, _
   section$, key$, setting As Variant)
End Sub

Public Sub DeleteSetting(appname$, _
   section$, Optional key$)
End Sub
```

FOR THE SAKE OF ARGUMENT

Because the intrinsic functions are designed to work with a two-level key structure, they use two arguments—appname and section. In the registry API calls, these arguments are combined with KeyPrefix into a single backslash-delimited key. You can create a private MakeKey function to build this single key from the method's arguments:

```
Private Function MakeKey(appname$, _
   section$) As String
If Len(section) = 0 And _
   Len(KeyPrefix) = 0 Then
   MakeKey = appname
ElseIf Len(section) = 0 Then
   MakeKey = KeyPrefix & appname
ElseIf Len(KeyPrefix) = 0 Then
```

```
   MakeKey = appname & "\" & section
Else
   MakeKey = KeyPrefix & appname & _
      "\" & section
End If
End Function
```

Given this single backslash-delimited key, you can read a value from the registry by opening that key within the proper root using the RegOpenKeyEx function. This function places the value of the open key in the hKey variable, which you can use in the QueryValueEx API function to read the actual value. Don't forget to close the registry key with RegCloseKey when you're done:

```
Public Function QueryValue(sKeyName$, _
   sValueName$) As Variant
Dim lRetVal&
Dim hKey&
Dim vValue As Variant
lRetVal = RegOpenKeyEx(Root, _
   sKeyName, 0, KEY_ALL_ACCESS, hKey)
lRetVal = QueryValueEx(hKey, _
   sValueName, vValue)
RegCloseKey (hKey)
QueryValue = vValue
End Function
```

Making the QueryValue function a public method of the class allows a more experienced programmer to call it directly when reading values from the registry, rather than forcing the programmer to use the GetSetting method. With these two functions, you can fill in the code for the GetSetting method. Notice how the default value is returned if QueryValue returns an empty string:

```
Public Function GetSetting(appname$, _
   section$, key$, Optional default)
Dim vValue As Variant
vValue = QueryValue(MakeKey(appname, _
   section), key)
If vValue = "" Then vValue = default
GetSetting = vValue
End Function
```

Your first method is done! Saving settings is nearly as easy. Because the registry supports several different types of values, you must specify the value type you're saving. I'll show you how to write this class to support either a string or a long value. Call the SetKeyValue function to save the value within the specified key. The SetKeyValue function calls the RegCreateKeyEx API function rather than RegOpenKeyEx, so the key will be created if it is not present in the registry:

```
Public Sub SaveSetting(appname$, _
   section$, key$, setting As Variant)
Dim lValueType&
If IsNumeric(setting) Then
   lValueType = REG_DWORD
Else
   lValueType = REG_SZ
End If
SetKeyValue MakeKey(appname, _
   section), key, setting, lValueType
End Sub
```

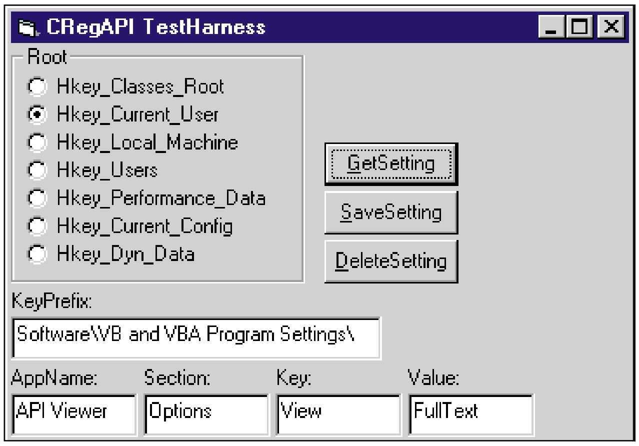
Finally, the DeleteSetting method has two slightly different actions. If a value is specified, you must delete only that registry value. But if no value is specified, you must delete the entire key. Delete from the registry by opening the key with RegOpenKeyEx and calling either RegDeleteKey or RegDeleteValue. You still need to call RegCloseKey to free up the memory used by the open key:

```
Public Sub DeleteSetting(appname$, _
   section$, Optional key$)
Dim lRetVal&
Dim hKey&
If Len(key) = 0 Then
   lRetVal = RegOpenKeyEx(Root, MakeKey(appname, ""), 0, _
      KEY_ALL_ACCESS, hKey)
   RegDeleteKey hKey, section
Else
   lRetVal = RegOpenKeyEx(Root, MakeKey(appname, section), 0, _
      KEY_ALL_ACCESS, hKey)
   RegDeleteValue hKey, key
End If
RegCloseKey (hKey)
End Sub
```

CREATE A TEST HARNESS

Your class is complete. As with all classes, you should prepare a simple test harness to debug your class. Add a form to your project, and add a button for each of your methods. Then add five text boxes for the KeyPrefix, appname, section, key, and value. If you want to experiment with other registry roots, you can add seven option buttons for the enumerated registry roots (see Figure 2). If you create these option buttons as a single control array with indexes zero through six, you'll be able to easily set the Root property of the class by adding the index to the value of *Hkey_Classes_Root.*

You can write your test harness by typing only five lines of code. First, define a public variable as a new instance of your CRegAPI class. Then, in the button Click events, add a line of code to call each of the three methods of your class, passing the values of the text-box controls as the arguments:


CRegAPI TestHarness

Root
- Hkey_Classes_Root
- [✓ Hkey_Current_User]
- Hkey_Local_Machine
- Hkey_Users
- Hkey_Performance_Data
- Hkey_Current_Config
- Hkey_Dyn_Data

KeyPrefix: Software\VB and VBA Program Settings\

AppName:	Section:	Key:	Value:
API Viewer	Options	View	FullText

FIGURE 2 ***Make Sure to Pass the Test.*** *Every class module needs a test harness—this simple one requires only five lines of code to test the methods of the CRegAPI class.*

```
Public reg As New CRegAPI
Private Sub butGet_Click()
txtValue = reg.GetSetting(txtAppName, txtSection, txtKey)
End Sub

Private Sub butSave_Click()
reg.SaveSetting txtAppName, txtSection, txtKey, txtValue
End Sub

Private Sub butDelete_Click()
reg.DeleteSetting txtAppName, txtSection, txtKey
End Sub

Private Sub Option1_Click(Index As Integer)
reg.Root = Hkey_Classes_Root + Index
End Sub
```

To test your class, try to read the *View* key from the *Options* section of the *API Viewer* appname by clicking on the GetSetting button. Set a break point in the Click event so you can step into the class and see the registry functions at work. You should see a value of *FullText* if your API Viewer is set like mine. Run the RegEdit program to find some other registry keys to explore, then compare the results to your class. When you want to use this class in one of your applications, simply include the CLS file or compile it as an ActiveX DLL and include a reference. Then change your application to call these methods and you can work with multilevel keys.

Code Online

You can find all the code published in this issue of VBPJ *on The Development Exchange (DevX) at http://www.windx.com. For details, please see "Get Extra Code in DevX's Premier Club" in Letters to the Editor.*

Use the Registry to Save Information

Locator+ Codes

Listings for the entire issue, Listing 1 for this article, and a registry API class that replaces Visual Basic's four intrinsic statements with more powerful counterparts (free Registered Level): VBPJ0398

Listings for this article only, plus the complete registry API class, including GetAllSettings, which uses multiple calls to enumerate the registry values (subscriber Premier Level): GS0398