# Run on Reboot

Click & Retrieve Source CODE!

by Karl E. Peterson

## Q Restart an App That was Running at Shutdown

I'd like to have my application restart automatically when Windows reboots if the app was running when the user last shut down Windows. Ideally, it should behave like Explorer, restoring its state on restart. How can I implement this feature?

**A** All 32-bit versions of Windows recognize four special registry entries that support this feature. Each does essentially the same thing—starts a program after the user logs in—but each has its own special meaning. You can find the two subkeys, Run and RunOnce, under SOFTWARE\Microsoft\Windows\CurrentVersion in both the HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE top-level keys.

The Run key is, for all intents, identical to having a shortcut in the StartUp folder of the Start menu. The only real difference is that the Run key is harder for the user to find and edit. Programs listed under Run start each time the user logs in. The RunOnce key instructs Windows to run that program only the *next* time the user logs in. It is removed from the registry automatically upon successful execution.

Choice of top-level key is the final factor that rounds out your implementation. If you place your entry under HKEY_CURRENT_USER, Windows starts the program the next time the *current* user, but no other user, logs in. Conversely, if you put the entry under HKEY_LOCAL_MACHINE, Windows starts the program regardless of who logs in next.

I've written a simple routine that takes all these factors into consideration (see Listing 1). Pass the RunNextBoot function two to four parameters indicating how you'd like your program to restart. The AppName parameter is the actual name used for the registry entry; It doesn't need to correspond to anything in particular, but should be meaningful to someone browsing the registry. The CmdLine parameter is the actual command Windows executes to start your application. The remaining two optional parameters, ThisUserOnly and RunEveryBoot, are Booleans that determine exact placement within the registry for your new entry.

The ideal place to call RunNextBoot is in the QueryUnload event of your main form. You can test the UnloadMode parameter to this event for vbAppWindows, which indicates your program is unloading because Windows is shutting down. For example:

```
If UnloadMode = vbAppWindows Then
   ' Windows is shutting down. Store app name
   ' and command line for execution next boot.
   Call RunNextBoot(AppName:="VBPJ QA9903", _
      CmdLine:= App.Path & "\" & App.EXEName, _
      ThisUserOnly:=True, RunEveryBoot:=False)
```

Following the call to RunNextBoot, you might also want to store other values to the registry to indicate the state your program should resume upon restart. For example, you might want to stash the WindowState, position, data file(s) being worked on, and anything else unique to the current state. Your app can check these values the next time it starts, and return the state to where it was when last shut down.

## Q Employ Single-Instance UserControls

I've written a special-purpose control that absolutely must be limited to a single instance per form. But VB has no apparent support for this requirement. I'll take whatever slimy hack you can throw my way at this point. Any ideas?

**ABOUT THIS COLUMN**

Ask the VB Pro provides you with free advice on programming obstacles, techniques, and ideas. Read more answers from our crack VB Pros on the Web at www.inquiry.com/thevbpro. You can submit your questions, tips, or ideas on the site, or access a comprehensive database of previously answered questions.

**A** You're right. Microsoft doesn't consider this need to be worth offering support for. Yet I can easily envision circumstances when you really want only a single instance to be used, such as when the control must subclass its parent form. Granted, you could document that "Very Bad Things" will happen if a user should place two or more instances on a form, but we all get tired of answering "RTFM" to the plaintive cries of the easily confused, don't we?

**I'll show you one method that stops most users dead in their tracks if they should ignore your warnings.**

**VB5, VB6 | It's Déjà Vu All Over Again**

```
Option Explicit
'
' Win32 Registry functions
'
Private 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 Any, phkResult As Long, _
  lpdwDisposition As Long) As Long
Private Declare Function RegSetValueEx Lib _
  "advapi32.dll" Alias "RegSetValueExA" (ByVal hKey As _
  Long, ByVal lpValueName As String, ByVal Reserved As _
  Long, ByVal dwType As Long, lpData As Any, ByVal _
  cbData As Long) As Long
  ' Note that if you declare the lpData parameter as
  ' String, you must pass it By Value.
Private Declare Function RegCloseKey Lib "advapi32.dll" _
  (ByVal hKey As Long) As Long
'
' Constants for Windows 32-bit Registry API
'
Private Const HKEY_CURRENT_USER = &H80000001
Private Const HKEY_LOCAL_MACHINE = &H80000002
'
' Reg Create Type Values...
'
Private Const REG_OPTION_NON_VOLATILE = 0
'
' Reg Key Security Options
'
Private Const SYNCHRONIZE = &H100000
Private Const STANDARD_RIGHTS_ALL = &H1F0000
Private Const KEY_QUERY_VALUE = &H1
Private Const KEY_SET_VALUE = &H2
Private Const KEY_CREATE_SUB_KEY = &H4
Private Const KEY_ENUMERATE_SUB_KEYS = &H8
Private Const KEY_NOTIFY = &H10
Private Const KEY_CREATE_LINK = &H20
Private Const KEY_ALL_ACCESS = ((STANDARD_RIGHTS_ALL Or _
  KEY_QUERY_VALUE Or KEY_SET_VALUE Or _
  KEY_CREATE_SUB_KEY Or KEY_ENUMERATE_SUB_KEYS Or _
  KEY_NOTIFY Or KEY_CREATE_LINK) And (Not SYNCHRONIZE))
```

```
Private Const ERROR_SUCCESS = 0&

Private Const REG_SZ = 1  ' Unicode nul terminated string

Public Function RunNextBoot(ByVal AppName As String, _
  ByVal CmdLine As String, Optional ThisUserOnly As _
  Boolean = False, Optional RunEveryBoot As Boolean = _
  False) As Boolean

  Dim SubKey As String
  Dim TopKey As Long
  Dim nRet As Long
  Dim hKey As Long
  Dim nResult As Long

  ' Assign subkey string appropriately.
  If RunEveryBoot Then
    SubKey = _
      "SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
  Else
    SubKey = _
      "SOFTWARE\Microsoft\Windows\
      CurrentVersion\RunOnce"
  End If

  ' Select appropriate top-level key.
  If ThisUserOnly Then
    TopKey = HKEY_CURRENT_USER
  Else
    TopKey = HKEY_LOCAL_MACHINE
  End If

  ' Open (or create and open) key
  nRet = RegCreateKeyEx(TopKey, SubKey, 0&, _
    vbNullString, REG_OPTION_NON_VOLATILE, _
    KEY_ALL_ACCESS, ByVal 0&, hKey, nResult)
  If nRet = ERROR_SUCCESS Then
    ' Write new value to registry
    nRet = RegSetValueEx(hKey, AppName, 0&, REG_SZ, _
      ByVal CmdLine, Len(CmdLine))
    Call RegCloseKey(hKey)
  End If
  RunNextBoot = (nRet = ERROR_SUCCESS)
End Function
```

**Listing 1** This handy routine writes an application name and command line into either the Run or RunOnce registry entries for either the current user or all users. Windows uses what you pass to the RunNextBoot function to determine whether and how to restart your application the next time a user logs in.

**The RunOnce key instructs Windows to run that program only the next time the user logs in.**

**VB5, VB6** | **Forcing a Single Instance**

```
Option Explicit

Private Sub UserControl_InitProperties()
   Call EnforceSingleInstance
End Sub

Private Sub UserControl_ReadProperties(PropBag As _
   PropertyBag)
   Call EnforceSingleInstance
End Sub

Private Function EnforceSingleInstance() As Boolean
   Dim n As Long
   Dim cntl As Control

   ' Make sure Parent property is valid.
   On Error GoTo BailOut
      n = UserControl.Parent.Controls.Count
   On Error GoTo 0
   n = 0

   ' Check to see how many instances of this control
   ' type (SingleUse, in this case) exist on Parent.
   For Each cntl In UserControl.Parent.Controls
      If TypeOf cntl Is SingleUse Then
         n = n + 1

         ' Raising an error from InitProperties or
         ' ReadProperties causes siting to abort.
         If n > 1 Then
            Err.Raise Number:=vbObjectError + 513, _
               Source:="SingleUse.ocx", _
               Description:= _
               "Only one instance allowed."
            EnforceSingleInstance = True
            Exit For
         End If
      End If
   Next cntl
BailOut:
End Function
```

**Listing 2** By raising an error during either the InitProperties or ReadProperties events of a UserControl, you effectively prevent the control from finishing its initialization. In fact, **all** instances of this control type are rendered dead at that point. The client has no way to trap such an error, so this is a measure to be taken only in extreme circumstances.

So I'll show you one method that stops most users dead in their tracks if they should ignore your warnings. It's not without its own perils, so consider your use of it carefully. In short, if you raise an error during either the InitProperties or ReadProperties events, the client app has absolutely no way to trap it. The control instance currently executing stops dead, as do all other control instances originating from the same module.

Call the code in EnforceSingleInstance from both InitProperties—for new controls—and ReadProperties—for controls pasted from the clipboard (see Listing 2). EnforceSingleInstance loops the Controls collection of the UserControl's Parent, counting the number of instances of the SingleUse type control. Note that, in this case, SingleUse is the name of the control and has no other inherent meaning. EnforceSingleInstance raises an error as soon as your control encounters a second instance.

At this point, the function redraws the original instance with diagonal lines over it, indicating it's no longer functional. The second instance is never drawn. The code in the OCX is totally nonresponsive. It's good if the user edits the form in the IDE. The only way for a user to regain response from the OCX is to close, then reopen, the form designer or run the application.

If a user attempts to add a second instance while the OCX is in limbo, funny things start to happen. The second instance actually appears on the form, but the error will be raised when the app is started. If the form designer is closed with two instances in place—one alive, the other dead—no one can open it again without editing the FRM file manually.

Hope this one's slimy enough for you. VBPJ

## About the Author

Karl E. Peterson is a GIS analyst with a regional transportation planning agency and serves as a member of **Visual Basic Programmer's Journal** Technical Review and Editorial Advisory Boards. Online, he's a Microsoft MVP and a section leader on several **VBPJ** online forums. Find more of Karl's VB samples at www.mvps.org/vb.

## DOWNLOAD FREE CODE

Download the code for this issue of **VBPJ free** from www.vbpj.com.

To get the free code for this entire issue, click on Locator+, the right-most option on the menu bar at the top of the VBPJ home page, and type **VBPJ0399** into the box. (You first need to register, for free, on DevX.) The free code for this article includes all code listings, plus the MRunOnce module, a sample control that allows only one instance of itself per form.

To get the bonus code for this article, available to DevX Premier Club members, type **VBPJ0399AP** into the Locator+ field. The bonus code includes all the free code described above, plus a class that emulates VB's intrinsic registry functions without the restriction on where entries may be placed, and a demo that ties this together with the RunNextBoot routine.