Click & Retrieve Source CODE!

Hacking the Windows Registry

BY KEITH PLEAS

It's a jungle out there, but with some guidance, an intrepid developer can unlock the secrets of the Win32 Registry.

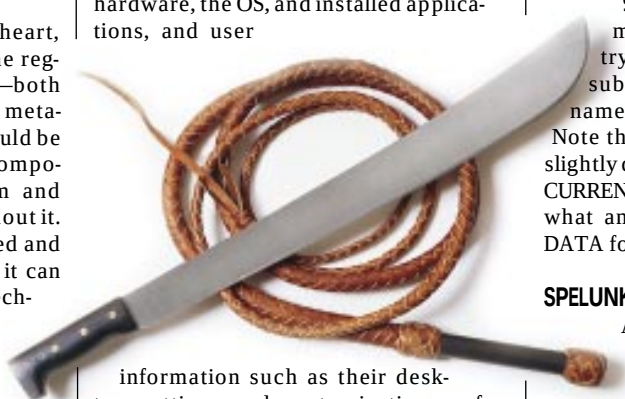
If USER, Kernel, and GDI are the heart, brain, and eyes of Windows, the registry would be the memory—both long and short term. OK, maybe this metaphor is a bit weak, but the point should be obvious: the registry is a critical component of a well-functioning system and you're not going to get very far without it.

The registry is lightly documented and not well understood. Programming it can be similar to the old neurological technique of zapping part of the cerebral cortex with an electrode and seeing what happens: the patient may remember a baseball game or experience a war-related flashback. In Windows, you may enable a cool new feature or render your system unbootable. But it's the thrill of the hunt that makes it so exciting.

After a brief introduction to get our terminology straight, I'll skip the fundamentals of the registry—MSDN would be an ideal place to find this information—and leap into advanced aspects.

Along the way I'll note a variety of thing you can take advantage of immediately: some of them are particular to the new Windows shell (first delivered on Windows 95 but currently in beta on Windows NT), some work only with NT (also known as "Microsoft's *real* operating system"), and some will work for everybody. So, grab your tools (primarily a copy of RegEdit) and prepare for an exciting round of hacking the registry.

The registration database, commonly called the registry, contains a substantial amount of data about the computer and users. It includes computer data such as hardware, the OS, and installed applications, and user information such as their desktop settings and customization preferences. The registry stores data in a hierarchically structured tree. Each node in the tree is called a key. Each key can contain additional keys called subkeys (see Figure 1). Keys are composed of printable characters and cannot include backslashes (\) or wildcard characters (* or ?). Several predefined keys, represented with uppercase words separated by underscores, can be accessed using numeric constants. These keys are always "open," so it's not necessary to use the RegOpen... functions on them. It's important to note that the root key for machine information HKEY_LOCAL_MACHINE (HKEY_CLASSES_ROOT and HKEY_CURRENT_CONFIG map to subkeys) and the root key for user information is HKEY_USERS (HKEY_CURRENT_USER also maps to a subkey). Keys beneath the root are referenced by building a string key by concatenating each node together, separated by backslashes.



Each key also contains data stored in values: a key may have no values, a default value, or any number of named values in addition to the default. The data in the values may be in a variety of forms, though text and binary data types are by far the most common. While key names and value names are never localized, text data often is. Using the Windows 95 RegEdit utility shows you a much compacted view of the registry including the root keys, several subkeys, a default (text) value, and a named (binary) value (see Figure 2). Note that Windows NT has a similar but slightly different structure: it omits HKEY_CURRENT_CONFIG and substitutes a somewhat analogous HKEY_PERFORMANCE_DATA for HKEY_DYN_ DATA.

SPELUNKING THE REGISTRY

A variety of common components can be found in the registry, especially if they have anything to do with OLE. Here are some examples so you'll know what you're looking at when you go spelunking with RegEdit.

Creatable OLE classes, provided by OLE servers, must be in the registry. Each class is registered separately in the HKEY_CLASSES_ROOT\CLSID key under its CLSID and must, at minimum, have enough information for the OLE system to locate and start the server. For example, Access registers the Application object with the key name on the left and the default value on the right:

```
{B54DCF20-5F9C-101B-AF4E 00AA003F0F07}    Microsoft Access Database
InprocHandler32                           ole32.dll
LocalServer32                             C:\MSOFFICE\ACCESS\MSACCESS.EXE
ProgID                                    Access.Application.7
```

Keith Pleas is an independent developer, author, and trainer. He is the author of the forthcoming book, Visual Basic Tips & Tricks, *from Addison-Wesley. He can be reached on Compu-Serve at 71333,3014 (from the Internet: 71333.3014@compu-serve.com).*

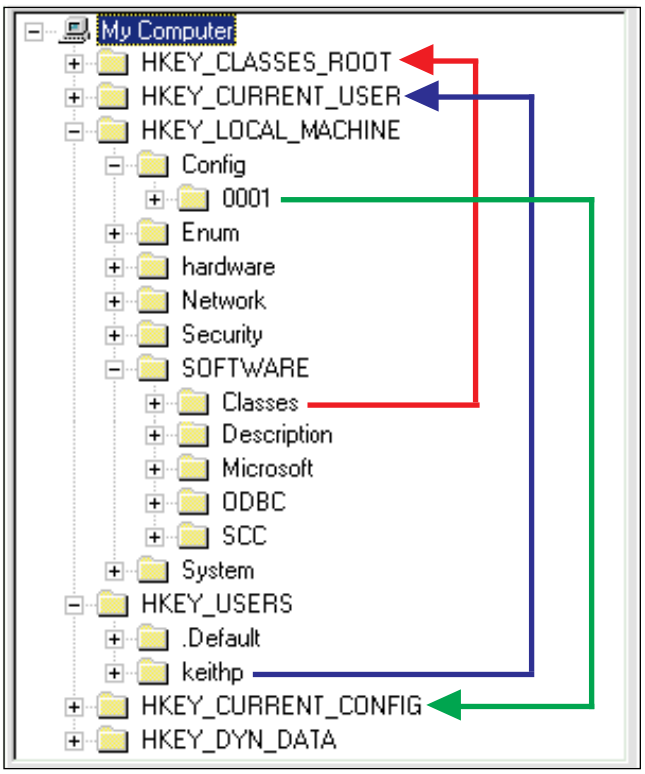


FIGURE 1 ***Related Entries in the Registry.*** *Expanded (Win95) registry keys depict how root keys map to major subkeys for current user, classes, and current configuration.*

OLE controls, being specialized in-process OLE servers, must be in the registry. If an OLE control is referenced by an application but is not in the registry, it can autoregister itself if the system can locate it by searching along the normal DLL search path.

OLE controls are registered as classes and can also be found in the HKEY_CLASSES_ROOT\CLSID key by referencing their CLSID. For example, the PicClip control that ships with VB4 has the following registry entries:

```
{27395F85-0C0C-101B-A3C9-08002B2F49FB}      PicClip Control
Control
InprocServer32                              C:\WINDOWS\SYSTEM\PICCLP32.OCX
Insertable
MiscStatus
ProgIDPicClip.PictureClip
ToolboxBitmap32                             C:\WINDOWS\SYSTEM\PICCLP32.OCX, 1
TypeLib                                     {27395F85-0C0C-101B-A3C9-08002B2F49FB}
Version                                     1.0
```

The Control key is used when dialog boxes like the OLE Insert Object dialog or VB4's Custom Controls dialog is displayed with the Controls box checked. InprocServer32 contains the fully qualified path to the control.

ProgID contains the so-called "friendly" name, which can also be found in a separate key under HKEY_CLASSES_ROOT: this separate key contains a pointer back to the CLSID where all the information for the control is maintained. The Insertable key behaves similarly to the Control key, though it may be duplicated under the ProgID key for backward compatibility with OLE 1.0 servers.

The type library for a control is indicated in the TypeLib key. Type libraries are stored separately in the registry under their own GUIDs in the HKEY_CLASSES_ROOT\TypeLib key. The entries for the PicClip control's type library are:

```
{27395F85-0C0C-101B-A3C9-08002B2F49FB}
1.0                                         Microsoft PictureClip Control
0
win32 C:\WINDOWS\SYSTEM\PICCLP32.OCX
FLAGS                                       2
HELPDIR                                     C:\VB4
```

Note that the type library itself can be stored as a separate file on disk (typically with a TLB or OLB extension) or attached as a resource to a DLL or EXE. Because OLE controls are in fact DLLs, their type libraries are most often stored with the control itself.

The HELPDIR key is notable because it points to the fully qualified location for the accompanying WinHelp file containing additional programming documentation about the control. This location can obviously vary by installation and is typically determined when the control is first installed: if the WinHelp file is moved the link can obviously be broken.

Licenses, such as those used by OLE controls, are also commonly stored in the registry. They can be found under the HKEY_CLASSES_ROOT\Licenses key, where you'll also find the warning that "Copying the keys may be a violation of established copyrights." No kidding. Anyway, each license is stored under its own GUID. This example from my registry database has both design and run keys (with the key values changed, naturally):

```
{B54DCF20-5F9C-101B-AF4E-00AA003F0F07}
    Retail                                  abcdefghijklmnopqrstuvwxyzabcdefghij
    Runtime                                 abcdefghijklmnopqrstuvwxyzabcdefghij
```

VB4 itself uses this technique: when it's installed it merges the contents of one of the three REG files (for Standard, Professional, and Enterprise editions) into the registry.

Finally, the registry contains information about *remoted OLE servers* in both their local and remote configurations. Like the other OLE object described here, this VB4-created OLE Automation server registers a Clerk class under its own GUID in the HKEY_CLASSES_ROOT\CLSID key. Of course, VB4 handles all the registration automatically and it's typically not necessary to modify these entries directly.

Running the Remote Automation Connection Manager (RacMgr32) utility included with VB4 Enterprise Edition adds additional keys for a remote machine name, RPC protocol, and RPC authentication level. When run locally, this particular class is registered as:

```
{8435CD47-D6BE-11CE-A842-00AA00688747}
    _AuthenticationLevel                    2
    _NetworkAddress                         NT
    _ProtocolSequence                       ncacn_ip_tcp
    InprocHandler32                         OLE32.DLL
    LocalServer32                           D:\PROJ\MSJ\CAR RENTAL\RENTAL OBJECTS.EXE
    ProgID                                  RentalObjects.Clerk
    TypeLib                                 {8435CD4E-D6EB-11CE-A842-00AA00688747}
```

When the class is remote, RacMgr32 changes the registration entries to:

```
{8435CD47-D6BE-11CE-A842-00AA00688747}
    _LocalServer32                          D:\PROJ\MSJ\CAR RENTAL\RENTAL OBJECTS.EXE
    AuthenticationLevel                     2
    InprocHandler32                         OLE32.DLL
    InprocServer32                          C:\WINDOWS\SYSTEM\autprx32.dll
    NetworkAddress                          NT
    ProgID                                  RentalObjects.Clerk
    ProtocolSequence                        ncacn_ip_tcp
    TypeLib                                 {8435CD4E-D6EB-11CE-A842-00AA00688747}
```

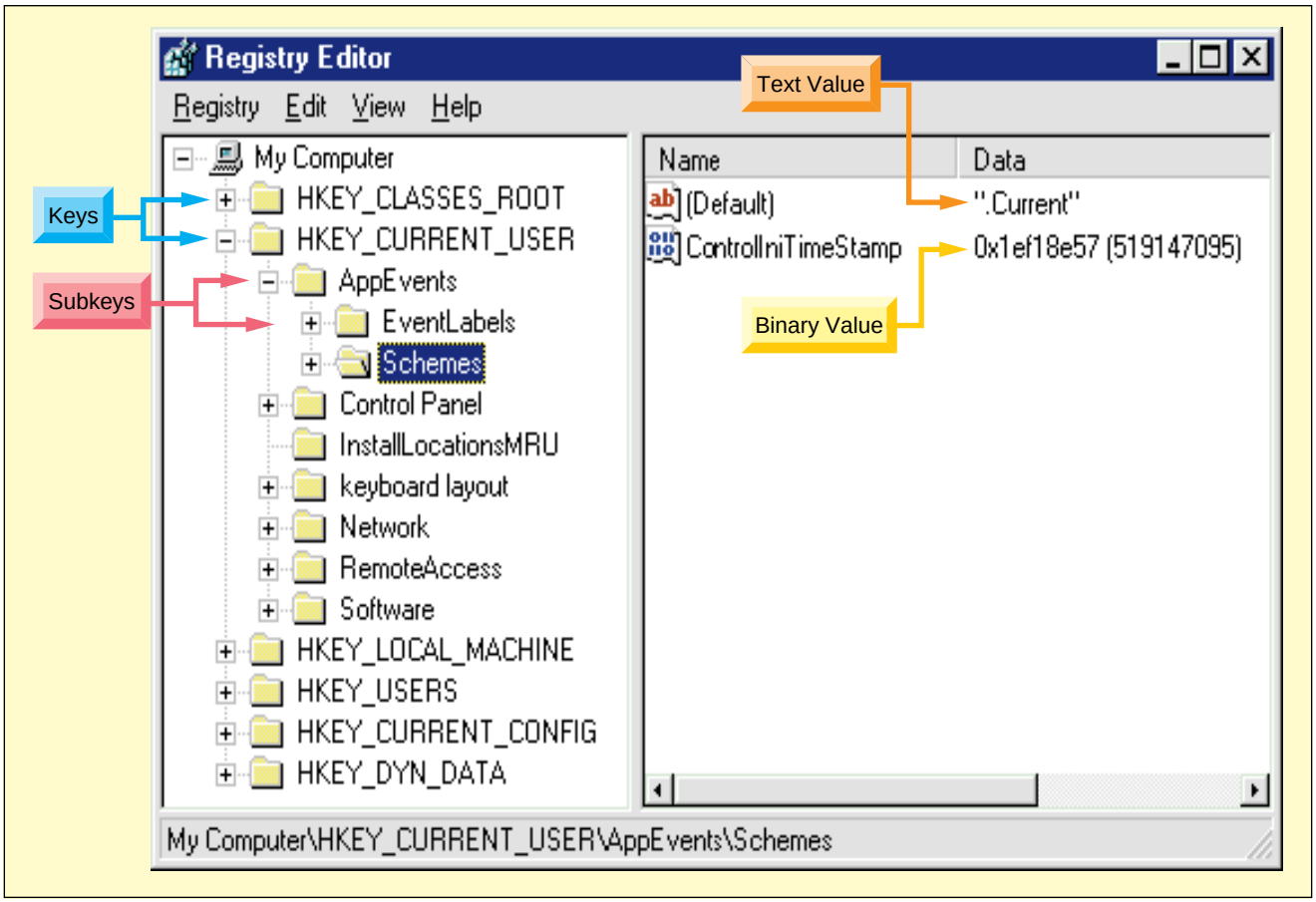


FIGURE 2 ***Keys to the Windows Registry.*** *The hierarchical structure of the registry consists of keys and subkeys. The associated values for each key can be named (text) or a non-string data type (binary).*

Notice how the LocalServer32 key gets renamed (actually, keys cannot be renamed, so it is destroyed and re-created) and an additional InprocServer32 key is created. This new key points to the remote automation proxy on the local machine, initiating a conversation with the AutMgr utility running on the remote machine.

Of course, you'll never want to touch these registration entries directly. In addition to using RacMgr32, we can also call the RacReg OLE Automation server in code to examine and change server settings. To do so add a reference to the RacReg32.DLL, create a RacReg.RegClass object, and use the GetAutoServerSettings function and SetAutoServerSettings method.

Unfortunately, the documentation for these functions is a little obscure: it's only found in the ReadMe file that ships with VB4. But it's pretty obvious how the RacReg32 server reads/writes the registry settings shown in this function prototype:

```
object.SetAutoServerSettings (Remote, [ProgID], [CLSID], _
   [ServerName], [Protocol], [Authentication])
```

A side benefit of using the RacReg.RegClass object is that Microsoft's VB group promises that your code will be upwardly compatible with future versions of VB, which will support true Networked OLE: they'll do the work of encapsulating the changes so that you don't have to change your code.

USING REGISTRY FUNCTIONS

The Win32 API provides a function group of 26 APIs, many of them with both "A" (ANSI) and "W" (Wide, or Unicode) versions, for working with the registry. Five of the 26 APIs are provided for backward compatibility only and shouldn't be used (the corresponding ...Ex functions, which support named values and access to keys other than HKEY_CLASSES_ROOT, should be used instead).

Rather than torture you with a complete list of the APIs, I'll point you to a couple of useful samples that highlight their implementation such as the RegTool sample that ships on the VB4 disc. The RegTool sample is buried down in the \Tools\Dataex32\Source\Regtool subdirectory and has a reusable class with routines for creating, updating, and deleting keys. Unfortunately, while it can read both string and numeric (dword) data, it can only write strings.

A much better example can be found in the file REGVB4.ZIP in the Magazine Library of the *VBPJ* Forum on CompuServe. Written by Don Bradner, *VBPJ* Forum Section Leader of the "32-Bit Bucket," REGVB4 is a handy VB4 version of RegEdit that has well-commented source code for reading and writing both string and numeric values.

Several of the registry functions deserve a bit more comment. While we do not yet have built-in support for a distributed registry (where part or all of your registry is stored on another machine), the RegConnectRegistry function can be used programmatically to connect to remote registries and get/set values from their registries. They can connect only through the root keys (HKEY_LOCAL_MACHINE and HKEY_USERS), but because of the subkey mappings to HKEY_CURRENT_USER, HKEY_CLASSES_ROOT, and HKEY_CURRENT_CONFIG this isn't a major limitation.

There are also a few differences between the Win95 and WinNT implementations of the registry functions. Of course, Win95 knows nothing about security, so Get/SetKeySecurity aren't implemented

on that platform. Also, while Win95 does implement QueryInfoKey, it doesn't track the last write time, so don't be surprised when the FILETIME structure comes up empty. Another thing to watch out for, particularly if you develop under Win95, is that RegDeleteKey on that platform deletes key and descendants, whereas on NT it can only delete keys that have no subkeys.

Because of its architecture, Win95 has very limited support for kernel synchronization objects, and thus RegNotifyChangeKeyValue is not supported at all. Win95 also doesn't implement RegRestoreKey, which can be worked around tediously by writing code to re-create the keys or, much easier, by using a REGEDIT4 file.

Interestingly, RegQueryMultipleValues is only implemented on Win95 (though its primary value appears to be as a coding shortcut). Finally, if you must store Unicode data in the Win95 registry you must store it as REG_BINARY, because Win95 is an ANSI system.

It's also worth pointing out that VB4 includes built-in functions for working with the registry, though they only work with information from a specific location in the registry:

```
HKEY_CURRENT_USERS\Software\VB and VBA Program _
   Settings\<program name>
```

I've seen a number of people experience problems with the built-in VB functions.GetSetting and GetAllSettings are functions, but SaveSetting and DeleteSetting are statements and thus don't use parentheses. While SaveSetting and DeleteSetting were originally specified as functions, later they became statements.

IMPORT DATA INTO THE REGISTRY

It's common to use registration (REG) files for importing data into the registry. REG files have two formats: REGEDIT and REGEDIT4. REGEDIT4 was introduced to deal with named values. RegEdit can run from the command line, but in this configuration, it will not be able to load REGEDIT4 files. If you're working on NT, you should use the RegIni utility from the NT Resource Kit.

CONTINUED ON PAGE 30.

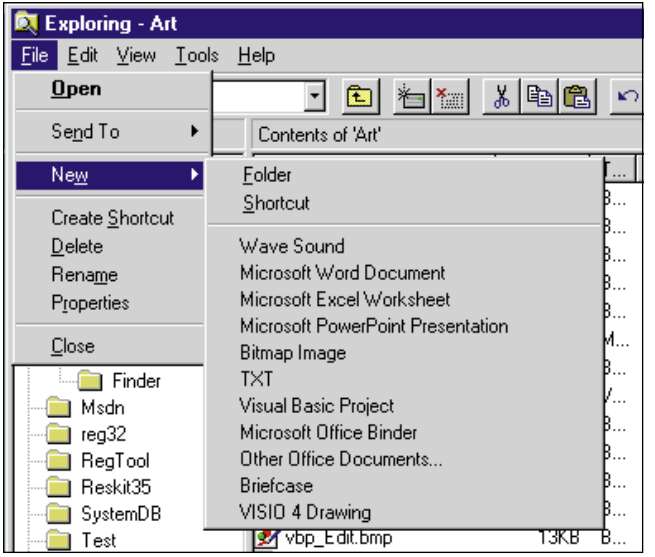


FIGURE 3 ***Adding the TXT File Type to the Explorer.*** *This view of the New menu in the Win95 explorer is fairly typical, except that by using the registry, I added the TXT file type to the menu. Selecting it launches Notepad, the file associated with TXT files.*

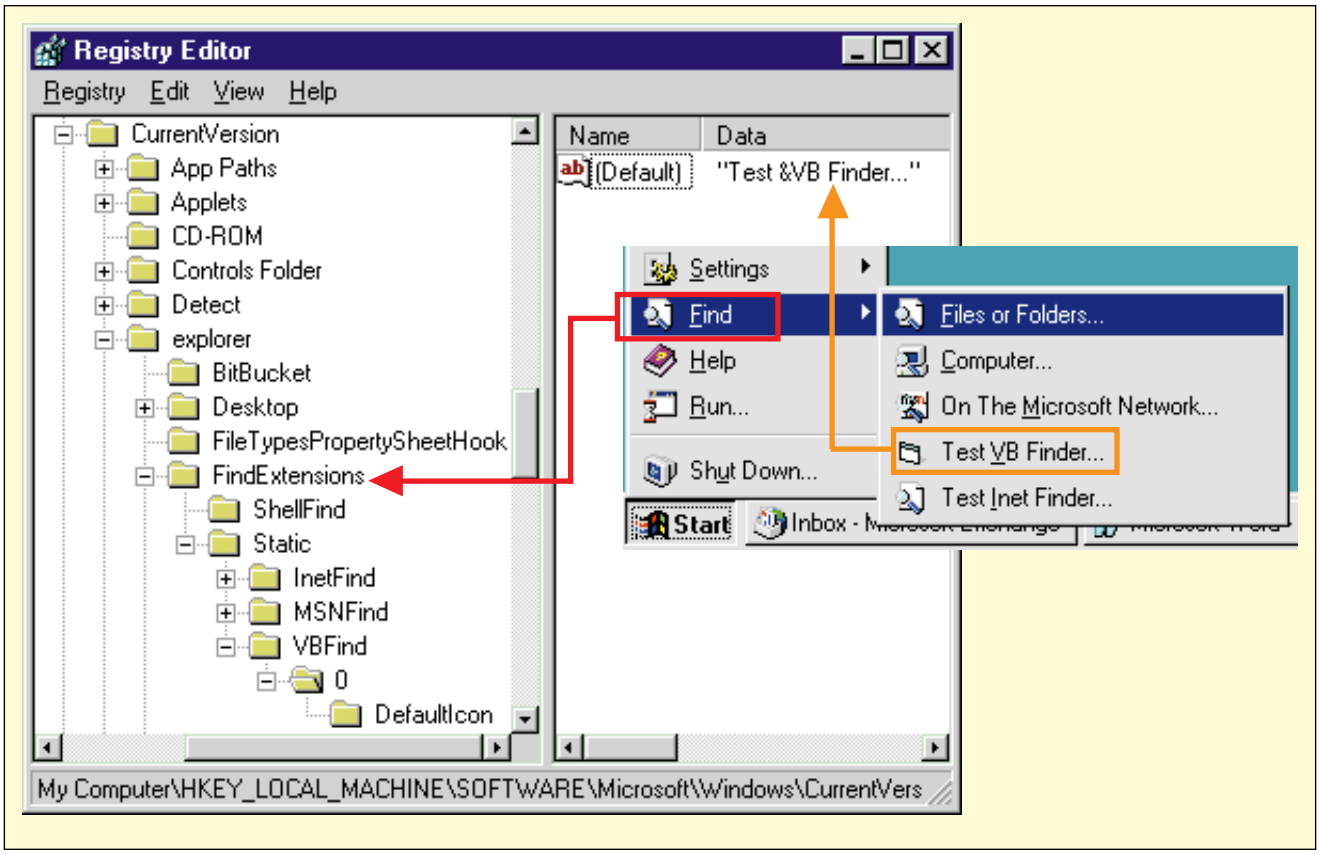


FIGURE 4 ***Adding the Test VB Finder to the Find Menu in Explorer.*** *The registry structure for dynamically added Find items illustrates how simple it is to add items to the menu. A modified Find Menu in the new shell's Explorer show an entry added by MSN as well as two custom entries described here. It's just as easy to add an entry for something like Yahoo for finding files on the Internet.*

CONTINUED FROM PAGE 26.

This shows the contents of a trivial REG file using the old format:

```
REGEDIT
HKEY_CLASSES_ROOT\.txt = txtfile
```

And, this shows this new format (with a named value):

```
REGEDIT4
[HKEY_CLASSES_ROOT\.txt]
@="txtfile"
"Content Type"="text/plain"
```

If you distribute a REG file with your application, be aware that Setup Toolkit has somewhat limited support for this. You can add a REG file with the Add Files button and the Setup Toolkit will register those keys on the user's machine. However, you are limited to embedding relative paths and there's no automated support for uninstalling the REG file entries.

If you've been following along on your machine, your registry might be getting a little wonky. It's not uncommon for your registry to get whacked: hacking around manually just tends to accelerate this process. Eventually, you're going to want to use the little-known RegClean utility (16- and 32-bit) that ships with VB4 and is located in the \Tools\PSS subdirectory. It can correct a number of these problems in your registry:

- Mismatched GUID in TypeLib.
- Missing TypeLib GUID.
- Missing CLSID for ProgID.
- Useless NumMethods or BaseInterface keys.
- Invalid ProgID key.
- Missing OLE key.
- Wrong value for OLE key.
- Missing file.
- Empty subkey.
- Conflicting local/remote keys.
- Improper InprocServer registration.
- Server isn't AUTPRX16.DLL/AUTPRX32.DLL.
- Differing server paths.
- Missing InprocServer key.

RegClean also gives you the option of creating a pending change file or just letting it rip and make the changes for you (guess which one I chose).

EXTENDING THE NEW SHELL

If you've selected New from the File menu within the Windows 95 Explorer, after what seems like an inordinate delay you've seen a cascading menu (see Figure 3).

The shell is searching through the registry looking for valid file extensions (those beginning with ".") that have a subkey of ShellNew. Each time it finds one, it reads the value in the extension's key to determine the ProgID, looks up the ProgID, and adds the value of that key to the menu.

For example, to add the TXT item to the menu shown in Figure 3, I added the ShellNew key to the CLSID key for ".txt" files:

```
HKEY_CLASSES_ROOT\.txt = txtfile
    ShellNew
```

CONTINUED ON PAGE 34.

VB4

```
Dim CRLF As String
Dim QT As String
Dim sFile As String
For x = 1 To Len(txtFile) 'Double \\
  If Mid$(txtFile, x, 1) = "\" Then sFile = _
    sFile & "\"
  sFile = sFile & Mid$(txtFile, x, 1)
Next x
CRLF = Chr$(13) & Chr$(10)
QT = Chr$(34)
txtScript = ""
txtScript = "REGEDIT4"
txtScript = txtScript & CRLF & "[HKEY_LOCAL_MACHINE\_

SOFTWARE\Microsoft\Windows\CurrentVersion\explorer\_
    FindExtensions\Static\" & txtShort & "]"
txtScript = txtScript & CRLF & "@=" & QT & _
  txtGUID & QT
txtScript = txtScript & CRLF & "[HKEY_LOCAL_MACHINE\_
  SOFTWARE\Microsoft\Windows\CurrentVersion\_
  explorer\FindExtensions\Static\" & txtShort & _
  "\0]"
txtScript = txtScript & CRLF & "@=" & QT & _
  txtDescription & QT
txtScript = txtScript & CRLF & "[HKEY_LOCAL_MACHINE\_
  SOFTWARE\Microsoft\Windows\CurrentVersion\_
  explorer\FindExtensions\Static\" & txtShort & _
  "\0\DefaultIcon]"
txtScript = txtScript & CRLF & "@=" & QT & sFile & _
  ",0" & QT
txtScript = txtScript & CRLF
txtScript = txtScript & CRLF & _
  "[HKEY_CLASSES_ROOT\CLSID\" _
  & txtGUID & "\FindCmd]"
txtScript = txtScript & CRLF & "@=" & QT & sFile & QT
txtScript = txtScript & CRLF & _
  "[HKEY_CLASSES_ROOT\CLSID\" _
  & txtGUID & "\InprocServer32]"
txtScript = txtScript & CRLF & "@=" & QT & _
  "FindExt.dll" & QT
txtScript = txtScript & CRLF & QT & _
  "ThreadingModel" & QT _
  & "=" & QT & "Apartment" & QT
txtScript = txtScript & CRLF
```

LISTING 1 ***REGEDIT4 Script Generation.*** *This script is pretty standard string manipulation code, with one exception. Note the required doubled backslashes and trailing blank line.*

which, when accessed by the shell, was translated into the:

```
HKEY_CLASSES_ROOT\txtfile = TXT
```

Of course, the point of this isn't that you can launch Notepad (though that is somewhat useful), but that you add your program to the New item from your users File menu with very little effort.

The shell can be extended in many other ways. For example, you can add a destination application to the Send To menu for all Explorer items by placing a shortcut to the destination application in the \Windows\SendTo folder. I suggest you create a shortcut in the \Windows\SendTo directory for RegSvr32.EXE. Heck, you don't even have to run RegEdit to do this one.

You may have clicked on files in the shell that don't have any extension: the resulting dialog is annoying but at least you can associate the file with a particular application. Unfortunately, that association doesn't "stick" and you have to do this every time. Files without an extension are of class "." and you must manually add this type to the registry. You can either add a single key that points to whatever (for instance) a "txtfile" might be:

```
HKEY_CLASSS_ROOT\. = "txtfile"
```

or you can enter your own class as in this example:

```
HKEY_CLASSS_ROOT\. = "none"
HKEY_CLASSS_ROOT\none\DefaultIcon = "notepad,1"
HKEY_CLASSS_ROOT\none\shell\open\command = "notepad.exe "%1.""
```

If you just want to add a single menu command to the context menu of a specific file type, you can use a similar technique method: these two entries will add an Edit menu item to VB project (VBP) files and load them into Notepad:

```
HKEY_CLASSS_ROOT\VisualBasic.Project\shell\Edit = ""
HKEY_CLASSS_ROOT\VisualBasic.Project\shell\Edit\command = "notepad.exe "%1.""
```

EXTENDING THE FIND MENU

The new shell can be extended in a number of ways using, not surprisingly, a mechanism called shell extensions. Shell extensions are implemented as specialized DLLs that create OLE COM objects and support specific OLE interfaces. One example is the built-in "Files or Folders..." and "Computer..." menu items found on the Find submenu. While it's possible to add to this menu, just as MSN does with the "On The Microsoft Network..." item, shell extensions cannot currently be written in VB.

Fortunately, Jeff Richter has written a custom FindExt.DLL that encapsulates the necessary functionality and allows attachment of any program to the Find submenu (see Figure 4). You generate custom CLSIDs that point to this DLL: when one is invoked, the DLL looks up the associated command line and executes it. This compiled DLL is included with the sample code for this article available on *VBPJ*'s Development Exchange on CompuServe (GO WINDX), The Microsoft Network (GO WINDX) and the World Wide Web (http://www.windx.com) and can be freely distributed. Richter will be writing about and publishing the source code later this year.

Extensions to the Find submenu are stored in the registry, buried in the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\explorer\FindExtensions subkey. Extensions stored at that level are loaded automatically when the Explorer is first loaded (normally the shell boots when Windows 95 is first loaded). The Static subkey beneath that contains extensions that are loaded dynamically: they are invoked when the user selects the item on the Find submenu. This is where you should put your custom find utilities.

To do so you need to create three additional nested subkeys: the extension that points to the CLSID of the InProc server, the menu text, and the menu icon. The first item to add is the extension that points to the CLSID of the InProc OLE server.

The name of this key (InetFind, MSNFind, and VBFind in the figure) is unimportant: Windows never displays it and the submenu items are actually drawn from the registry in the order they were added, not alphabetically. The value of this key is the text version of a CLSID that points to FindExt.DLL, in this case. Next, add the menu text itself (including an accelerator key if desired). The name of this key must be "0."

Finally, add the icon to be displayed in the menu, which has a value that includes the file name of the executable and the index of the icon (typically zero) to be used. The name of this key must be "DefaultIcon."

To see the new menu item, it's necessary to restart the Explorer. You can either restart Windows 95, which is slow and inconvenient, particularly if you have multiple applications open, or you can shut down and restart the shell. To shut down the shell, choose "Shutdown" from the Start menu and, when you see the "Shut Down Windows" dialog box, hold down the

CONTINUED ON PAGE 38.

VB4

```
Declare Function RegNotifyChangeKeyValue Lib _
  "advapi32.dll" _
  (ByVal hKey As Long, ByVal bWatchSubtree As Long, _
  ByVal dwNotifyFilter As Long, ByVal hEvent As Long, _
  ByVal fAsynchronus As Long) As Long
Declare Function WaitForSingleObject Lib "kernel32" _
  (ByVal hHandle As Long, ByVal dwMilliseconds As _
  Long) As Long
Declare Function CreateEvent Lib "kernel32" Alias _
  "CreateEventA" (lpEventAttributes As Long, ByVal _
  bManualReset As Long, ByVal bInitialState As Long, _
  ByVal lpName As String) As Long
Declare Function CloseHandle Lib "kernel32" (ByVal _
  hObject As Long) As Long
Public Const HKEY_CLASSES_ROOT = &H80000000
Public Const REG_NOTIFY_CHANGE_ATTRIBUTES = &H2
Public Const REG_NOTIFY_CHANGE_LAST_SET = &H4
Public Const REG_NOTIFY_CHANGE_NAME = &H1
Public Const REG_NOTIFY_CHANGE_SECURITY = &H8

Private Sub cmdRegistry_Click()
  Dim lChange As Long
  mhEvent = CreateEvent(0&, False, False, vbNullString)
  lChange = RegNotifyChangeKeyValue_
    (HKEY_CLASSES_ROOT,  True, _
    REG_NOTIFY_CHANGE_NAME, mhEvent, True)
  tmrRegistry.Enabled = True
  Me.Caption = "Waiting for registry change..."

End Sub

Private Sub tmrRegistry_Timer()
  Static lSignal As Long
  Static lResult As Long

  lSignal = WaitForSingleObject(mhEvent, 0&)
  If lSignal = 0 Then
    Me.Caption = "Registry Changed"
    tmeRegistry.Enabled = False
    lResult = CloseHandle(mhEvent)
  End If

End Sub
```

LISTING 2 ***Declarations and Code for Handling Registry Change Notification.*** *The cmdRegistry_Click subroutine creates the event object, passes its handle to the system signalling when the registry changes, and starts the polling timer. Details about Registry Change Notification messages are shown in Table 1.*

Ctrl-Alt-Shift key combination and click on the "No" button. This leaves you in something like the old shell, where pressing Ctrl-Escape brings up the Task Manager, from which you can select "Run" from the File menu and restart Explorer.

Although the menu item is visible at this point, it won't actually do anything. To make it work, you must add the CLSID to the HKEY_CLASSES_ROOT\CLSID key and create a couple of additional subkeys: the CLSID of the OLE InProc server referenced by the Find extension, the command line to be executed by FindExt.DLL, which must be stored under the FindCmd key, and finally the InprocServer32 key with two values. The first, which is the default, contains the path (if appropriate) and file name of the FindExt.DLL, which will typically be located in the \Windows\System subdirectory.

The second key, "ThreadingModel," should be set to "Apartment" because the FindExt.DLL uses that mechanism and is, in fact, thread safe. The threading model applies only to OLE Servers that are loading in process. The steps I've outlined are a bit tedious, yet they must be carried out exactly for this to work properly. To ease the procedure, I wrote a small Finder Installation utility that automates the whole process (available for download from the online services described elsewhere in this article).

The first step in using this utility is to generate a new CLSID, which is equivalent to a GUID (for "Globally Unique ID" in Microsoft terminology) or UUID (for "Universally Unique ID," in DCE/RPC terminology).

VB creates GUIDs for us automatically when we create OLE Servers, and the GUIDGen utility included in the Win32 SDK can be used to generate them manually. Anyway, I want to create a CLSID programmatically so I need to create a GUID structure and fill it in by calling the OLE function CoCreateGuid, which in turn calls the RPC function UuidCreate.

The Win32 documentation states that UuidCreate is not implemented on Windows 95, but that isn't true: it can be found in RPCRT4.DLL.

The Win32 header files give this structure for a GUID:

```
typedef struct _GUID {     // size is 16
   DWORD Data1;
   WORD Data2;
   WORD Data3;
   BYTE Data4[8];
} GUID;
```

which I translated into this VB code:

```
Type tGUID
   P1 As Long
   P2 As Integer
   P3 As Integer
   P4 As Byte
   P5 As Byte
   P6 As Byte
   P7 As Byte
   P8 As Byte
   P9 As Byte
   P10 As Byte
   P11 As Byte
End Type
```

The CoCreateGuid declaration was pretty obvious:

```
Declare Function CoCreateGuid Lib _
   "OLE32.DLL"  (guid As tGUID) As Long
```

Calling it is dead simple:

```
Dim tmp As tGUID
lRet = CoCreateGuid(tmp)
```

Unfortunately, the GUID you end up with is binary. You need a string in this format: "{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}". The Win32 API does provide a UuidToString function located in RPCRT4.DLL and the Win32 SDK header files provides this prototype:

```
UuidToStringA (
   IN UUID __RPC_FAR * Uuid,
   OUT unsigned char __RPC_FAR * __RPC_FAR _
      * StringUuid
   );
```

But, it turns out that this function isn't callable from VB. However, another function, StringFromGUID2, gets us on the right track using this declaration:

```
Declare Function StringFromGUID2 Lib _
   "OLE32.DLL"  (guid As tGUID, lpszString As _
   Byte, lMax As Long) As Long
```

MESSAGE	DESCRIPTION
REG_NOTIFY_CHANGE_NAME	Changes to key names that occur in the specified key or in the specified key and its subkeys cause a change notification. This includes key creations and deletions.
REG_NOTIFY_CHANGE_ATTRIBUTES	Attribute changes that occur in a key or in a key and its subkeys cause a change notification.
REG_NOTIFY_CHANGE_LAST_SET	Changes to the last write time that occur in a key or in a key and its subkeys cause a change notification.
REG_NOTIFY_CHANGE_SECURITY	Security-descriptor changes that occur in a key or in a key and its subkeys cause a change notification.

TABLE 1 ***What's Changed?*** *Registry change notification messages, and their descriptions. Be aware that some messages that exist on Windows NT aren't supported by Windows 95.*

Calling this function and putting the result into the Text control is a piece of cake:

```
Dim bBuff(256) As Byte
lRet2 = StringFromGUID2(tmp, bBuff(0), 256&)
txtGUID = bBuff
```

These three lines of code are doing a lot. The contents of the bBuff byte array are actually a Unicode string. If you examine it in detail, you'll see that every element contains the ASCII value of a character that you want in the string version. Assigning the contents of the buffer to a string (or, in this case, the text property of a Text control) converts it correctly because VB4 strings are internally Unicode.

The second and third steps are to simply fill in the extension key name (which is not used), menu text, and complete command line that we wish to execute.

The fourth step is to generate a complete REGEDIT4 script that contains all of the entries in the appropriate format. This is straightforward VB string manipulation code (see Listing 1) with these caveats: any key value containing a backslash character must be doubled and the script must have a blank line at the end for the previous line to be registered correctly. The last step is to copy this script into a REG file and execute it from the shell.

Again, because you create your own CLSID, you can have any number of Find extensions on a system without worrying about colliding with one written and installed by someone else. Because the FindExt.DLL is internally calling the new Win32 ShellExecute function, you can even substitute the executable file name with something like this:

```
http://www.yahoo.com
```

You might associate this with the menu description "On The &Internet..." Choosing this automatically brings up the Internet Explorer, logs you on to the Internet, and take you to the Yahoo finder. Other ideas for Find extensions might include a company-wide address book, a shortcut to MSDN, or virtually anything else that makes sense to you.

DIFFERENCES BETWEEN NT AND 95

As developers are all too painfully aware, there are major differences between the Windows 95 and Windows NT platforms. Some of these differences will disappear over time: the NT Shell Update Release (SUR) will add the new shell, TAPI support, and so on, while some of the most glaring differences, like Windows 95's lack of security, will remain. One of the gray areas is support for theWin32 Kernel synchronization objects: while support for the file change notifications is supported through the FindXXXChangeNotification family of APIs on both platforms, support for registry change notifications (through RegNotifyChangeKeyValue) is supported only on NT.

While a full discussion of kernel synchronization objects—such as mailslots, processes, threads, mutexes, events, semaphores, file handles, file mappings, named pipes—will have to wait until another time, I'll cover only registry synchronization for now.

Kernel event objects can exist in either a signaled or not-signaled state. Basically, we create an event object, tell the system to signal that object when the registry changes, and wait for the object to get signaled. Normally this is done synchronously by suspending the calling thread until the signal occurs.

Unfortunately, because VB apps can currently use only a single thread, this would have the effect of hanging the entire app until the change occurs. Freezing an application is considered to be sub-optimal from an implementation standpoint (users generally don't like this), so I programmed around this limitation using a Timer and periodically checking the state of the event. While polling is usually a sign of a bad application architecture, in this case there's no other choice.

To illustrate this, I created a small testing application that's easy to follow (see Listing 2). The code starts in the cmdRegistry_Click subroutine, which creates the event object, passes its handle to the system to get signaled when the registry changes, and starts the polling timer. The timer calls WaitForSingleObject (with a time of 0 milliseconds) and returns immediately.

When the event gets signaled, the timer is disabled and the event object is destroyed by closing its handle. This particular example looks for changes to key names at the root level of HKEY_CLASSES_ROOT and includes subkeys: it's probably the most useful, although you may want to examine the other options from the Win32 SDK (see Table 1).

As a final reminder, since the RegNotifyChangeKeyValue function is implemented only on Windows NT, this tester won't do anything on Windows 95.

Here are some useful tips. First, any long file names stored in the registry should be enclosed in quotes, like this:

```
shell\open\command = "C:\Program Files\My Accessories\WinWord.Exe" %1
```

Alternately, the short file name could be stored so it will work on all systems. An example of this is the system-supplied Find utility that supplies the "Files or Folders..." and "Computer..." menu items:

```
C:\Progra~1\TheMic~1\findstub.dll
```

While type and size of data you can store in the registry is relatively unlimited, in general you should not store frequently accessed data in the registry. Registry access is much slower than shared memory and even slower than file access. You should also be aware that named values consume less space than keys consume. You might also consider packing data together into a structure and storing the entire structure as a single binary value.

If your application is adding more than a couple of kilobytes to the registry, consider storing a pointer to that data and locating it elsewhere, either as a file or perhaps as a type library.

Also, while it's certainly possible, Microsoft strongly encourages developers not to store binary, executable programs in the registry. If you're still interested in the registry and are looking for a place to jump in where you're likely to see familiar stuff, I'll leave you with these keys as "suggested reading:"

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\SharedDlls
HKEY_LOCAL_MACHINE\System\CurrentControlSet\control\SessionManager\KnownDLLs
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\IniFileMapping
HKEY_LOCAL_MACHINE\\SOFTWARE\Microsoft\Windows\CurrentVersion\AppPaths
HKEY_LOCAL_MACHINE\\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
```

