Visual Studio Magazine Online
Classic VB Corner

Got One Right!

ONLINE ONLY

Can you name one thing Microsoft designed right the first time?

December 8, 2008 · by Karl E. Peterson

Conventional wisdom has it that version 1 of any Microsoft product or idea is really an alpha, that version 2 is the beta, and that the safe money rides on version 3. Then, following version 3, Microsoft generally proceeds to do its best to upset the success it found in that golden release. Here's a trip down memory lane, to one of the most durable version 3s ever to come out of Redmond.

Microsoft Windows 1.0 introduced the concept of INI files, which were used to store system configuration information for Windows itself. Almost immediately, people recognized the opportunity to tweak Windows behavior by hand-editing these pure text files, and "tips and tricks" began appearing in publications like PC Magazine. Knowing your way around WIN.INI and SYSTEM.INI was a self-promotion to Office Geek.

The Windows 2.0 SDK was the first to offer a standardized INI file API, providing the GetProfileString, WriteProfileString, GetProfileInt and WriteProfileInt functions. Graybeards will recall how this limited set of functions encouraged every application programmer to use WIN.INI as their configuration information depository. Unfortunately, that file became "troublesome" when it grew over 32K, so a better solution was needed.

When the Windows 3.0 SDK was shipped, it included the "private" versions of each of the above functions, thus encouraging each application to store its data within its own configuration file. Gold! This simple mechanism provided a standardized approach for all application developers to store the sort of information that's only available at or after installation. The version 1 design was perfect, and the version 3 interface sealed the deal.

An often overlooked side-benefit was how greatly this innovation eased tech support. Any user could open an INI file in Notepad, edit its contents while talking to the support person, save and try again. Separating application INI files from the system INI files meant no risk to the overall system from user tweaks, either.

The INI file format was somewhat standardized, finally, with the release of the Windows for Workgroups 3.11 Resource Kit. Not that Microsoft itself ever really stuck to this standard, of course (the prime example being the multitude of **device=** lines in SYSTEM.INI, but then that was the alpha release). You're probably well aware of the basic structure -- groupings of named sections each including various parameter-value pairs.

About this time, I was becoming very active in the MSBASIC forum on CompuServe, and INI file access from VB3 was routinely in among top three most frequently asked questions. I wrote a little INIFILE.BAS module and posted it as KPINI to that forum. It instantly became one of the most downloaded samples there. I updated it a few times so it could do just about anything to/with an INI file that one might imagine, including accessing all those pesky **device=** lines in SYSTEM.INI (that was the only one that required non-API code) and even added a demo that would iterate any INI file.

Then, the Win32 API arrived, and Microsoft began encouraging developers to use the registry to store application settings. The registry offered better hierarchical structure and also supported binary storage. Who didn't see what was coming next? It was as if Microsoft didn't learn anything from the earlier WIN.INI mess. The curse of registry bloat is still with us, and because of this the only way to get an older machine back to its peak performance is to flatten it and rebuild from scratch.

The current kick is using XML files for anything and everything user-related. Yes, they offer better hierarchical options, are easily accessible programmatically and can be stored apart from critical system information. But just try walking a relative through editing one of these (or the registry!) over the phone. Simplicity, they're not.

So, 10 years after 32-bit VB4 shipped, I finally reposted a full 32-bit implementation of KPINI on my Web site. My solution now offers a class-based, event-driven method to explore the contents of any INI file, as well as specific methods to read or write to any parameter-value pair within. Given INI files are inherently textual, there also needed to be methods that consistently reinterpret values in specific ways. For example, the ToBoolean method will return True for any number of possible user ways of expressing that:

```
Public Function ToBoolean(ByVal EntryValue As String) As Boolean
  ' Interpret entry as either true or false.
  Select Case Trim$(UCase$(EntryValue))
    Case "YES", "Y", "TRUE", "T", "ON", "1", "-1"
      ToBoolean = True
    Case "NO", "N", "FALSE", "F", "OFF", "0"
      ToBoolean = False
    Case Else
      ToBoolean = False
  End Select
End Function
```

Using CIniFile.cls should become very intuitive with just a few tries. The download includes a sample utility that iterates through any INI file pointed to, and the page you download it from offers a few more examples. Thousands of folks have found it immediately useful in both Classic VB and VBA applications. The 16-bit version is still included, for nostalgia's sake. I hope you enjoy it, and would welcome any questions or comments about it.

About the Author

Karl E. Peterson wrote Q&A, Programming Techniques, and various other columns for VBPJ and VSM from 1995 onward, until Classic VB columns were dropped entirely in favor of other languages. Similarly, Karl was a Microsoft BASIC MVP from 1994 through 2005, until such community contributions were no longer deemed valuable. He is the author of VisualStudioMagazine.com's new Classic VB Corner column. You can contact him through his Web site if you'd like to suggest future topics for this column.