Visual Studio Magazine Online
Classic VB Corner

Prisoner of Geography

ONLINE ONLY

When it comes to our understanding of Unicode issues, the "Born in the USA!" chant (no matter how you feel about The Boss) can almost amount to a proclamation of cultural ignorance.

March 2, 2009 · by Karl E. Peterson

Classic VB is downright schizophrenic when it comes to Unicode. It's love-hate all the way. Since 32-bit VB4, every blessed String is Unicode -- unless you try to pass it outside your application, at which time VB does its damnedest to convert it to something you may not want. During the VB4 beta, we dubbed this UniMess, which is truly appropriate, as this single change broke every existing line of file i/o code ever written with MS BASIC, among numerous other problems it caused. *If you want a new type of data, you create a new datatype!* Heh, but I digress.

Not long after I wrote a column some time ago about how Vista broke the native SendKeys statement in Classic VB, e-mails started trickling in asking if my fix for this problem might be extended to include support for Unicode. Being a relatively Unicode-ignorant U.S.ian, I wasn't entirely sure. Heck, I wasn't even sure about some of the simplest concepts, such as whether shift keys were still relevant. After saying "I dunno" more than a few times, I decided I'd try to find out if this was indeed possible.

As with virtually every method of transferring string data out of VB, the native SendKeys statement converts all characters to ANSI. If one were to try sending Unicode characters, they'd end up with those irritating question mark characters in the output window. But the SendInput API offers us direct support for Unicode characters, so the only real question was how to integrate this with our existing solution.

As it turns out, the implementation of SendInput doesn't even support shift keys with Unicode characters anyway, so that made that part of the decision relatively painless. I decided to just wing it and try injecting the output buffer with Unicode characters as they were encountered in the input string. The only question at that point was how to determine what was an "actual Unicode character" within a BSTR which is, by definition, *composed of nothing but Unicode characters*.

Here comes the ignorant U.S.ian part. The best I could come up with was to treat every character in the AscW range of 0-255 as "normal" and those falling outside that range as Unicode. Is this the correct way to handle it? I didn't know, so I asked a group with more diverse experiences than mine. Is this routine sufficient to distinguish what sort of character is being sent?

```
Private Sub ProcessChar(this As String)
   Dim code As Integer
   Dim vk As Integer
   Dim capped As Boolean

   ' Determine whether we need to treat as Unicode.
   code = AscW(this)
   If code >= 0 And code < 256 Then 'ascii
      ' Add input events for character, taking capitalization
      ' into account.  HiByte will contain the shift state,
      ' and LoByte will contain the key code.
      vk = VkKeyScan(Asc(this))
      capped = CBool(ByteHi(vk) And 1)
      vk = ByteLo(vk)
      Call StuffBuffer(vk, capped)
   Else 'unicode
      Call StuffBufferW(code)
   End If
End Sub
```

Opinions varied. Apparently, the characters in the 128-255 range are dependent on what code page is active. It doesn't seem like this is really all that relevant, though, for a SendKeys type of activity. As it turns out, no one has said this isn't working for them. I'd like to know if it works for you, and if it doesn't I'd like to know how so or why.

Getting back to the guts of what's going on here, you can see that shift keys are detected and sent along with the character to be processed when working in the ANSI range. But for Unicode characters, we simply stuff the buffer with the character code. That and, of course, set the flag telling SendInput this is a Unicode character.

```
Private Sub StuffBufferW(ByVal CharCode As Integer)
   ' Unicode is relatively simple, in this context?!
   ' Press and release this key.
   With m_Events(m_EvtPtr)
      .wVK = 0
      .wScan = CharCode
      .dwFlags = KEYEVENTF_UNICODE
   End With
   m_EvtPtr = m_EvtPtr + 1
   With m_Events(m_EvtPtr)
      .wVK = 0
      .wScan = CharCode
      .dwFlags = KEYEVENTF_UNICODE Or KEYEVENTF_KEYUP
   End With
   m_EvtPtr = m_EvtPtr + 1
End Sub
```

Well, long story short, that's about all there was to it, to add support for Unicode to the SendKeys replacement module. Assuming all the assumptions embodied in that initial test are actually valid. I'm hoping that putting this before another worldwide audience will draw out any additional tweaks that might make this an even more robust solution. Nothing beats throwing code out to a merciless group, if you want to really tighten it up!

One other enhancement that came out of that thread was a trap for Windows 95 being added to the module, as SendInput isn't available there. So in that case, VB's native SendKeys will be used instead.
I asked this in my last column, and I'd like to ask again: How many of you still feel the need for Windows 95 support? Is that still a real market? I don't consciously decide I'm not going to support it, generally; rather, I tend to forget these days what the old limitations were. I very clearly remember back when I chose not to include new Windows 2000 functionality, for fear of not having a viable alternative on the older platforms. Unfortunately, my memories of what exactly that functionality was have become a little fuzzy.

My inclination is that many of you still actively support Windows 95, and even NT4, as I get reminded about my forgetfulness on these points more and more. Let me know if I should continue coding around that in what I post here.

Oh, and here's a gratuitous external link to the sample code.

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.