Visual Studio Magazine Online
Classic VB Corner

Discarding More Dependencies

Nearly all applications need to open or save data files. Use the common file dialog APIs to easily avoid another unneeded dependency and sidestep some system bugs as a bonus.

December 22, 2010 · by Karl E. Peterson

In my last column, I demonstrated how you can easily avoid adding an external dependency by using an API directly, rather than an OCX wrapper around that API, to let your users choose a particular font setting. But realistically, how many apps need that capability? Well, nearly all apps have to open and/or save data files, and the same lesson applies. The GetOpenFileName API is only a bit more complicated than ChooseFont, and well worth the effort to master!

As luck has it, Bruce McKinney also provided us a wonderful starting point for this API call, in his classic *Hardcore Visual Basic* (MSPress, 1997), which I have again embellished with my own bells and whistles. My first task was to disentangle his routines from the nasty dependencies they had on type libraries for API (and other) declarations. My philosophy has always been to keep the declares local, for maximum flexibility.

Moving on, I'm at that point now where I simply don't support Win9x all that much anymore. I have that luxury, if you will, as a great deal of my work is in-house or at least targeted at the non-consumer market. So the first substantive change I made was changing from ANSI to Unicode APIs for all calls, so that any file or folder name could be easily supported. In reality, this change was almost as simple as modifying the declares, and assigning StrPtr(sData) to parameters rather than simply passing sData directly.

Now here's where it gets a bit more interesting. The real substantive change I made was to provide a callback for GetOpenFileName, which Windows notifies when certain events occur in the dialog. As time goes on, I find situation after situation where this callback can save my bacon. There are at least two rather unpleasant bugs in the way the common dialog handles certain situations on certain platforms, both of which can be avoided with this callback.

In both Windows 2000 and Windows XP, if the user selects an existing file in a standard File-Open dialog (that uses the OFN_FILEMUSTEXIST flag) and then navigates to a different folder, a non-standard error dialog will pop up. If you still have a machine or VM running XP, you can test this easily with Notepad. Just go File-Open, pick any file, navigate up or down a folder level, and press OK. Compare that error with what you get if you just type in a non-existent filename and press OK.

If you setup a OFNHookProc callback, you will have the opportunity to test the input file's existence when Windows sends you a CDN_FILEOK notification. This message is

sent when the user presses the OK button, and is your opportunity to run any sort of validation you want on the input file(s). You can display your own custom error, and reject the dialog dismissal, by returning a non-zero value from OFNHookProc, which instructs the dialog box procedure to ignore the message.

Well, that was the way it was. Unfortunately, in Windows 7, Microsoft must have decided that we shouldn't have to do that ourselves anymore and CDN_FILEOK is now suppressed if the OFN_FILEMUSTEXIST flag is set and the file doesn't actually exist. Instead, Windows just shows its dirt-standard error dialog, and your routine is none the wiser. (This may be true in Vista, as well, but who would know?)

So the easy answer, given we already have the hook, is to simply remove the file/path-must-exist flags and do the test ourselves. This gives us back the ability to show a consistent and potentially customized dialog alerting the user which file(s) may have a problem. That's another issue with Windows dialog -- the user isn't told which of several selected files are purported to not exist.

Now, if your mind is already racing ahead, perhaps you're sensing where I'm going with this next. If testing for file existence makes sense, maybe your application or its users would benefit from other validation tests as well! For example, if you were asking for a file to upload, maybe it'd make sense to do an "Are you sure about that?" dialog if the user selected one that was over some set size that made sense.

Maybe you know that uploading more than 5MB would cause the receiving system to fail (which can easily be the case with email attachments). Would you, as a user, rather have this information presented to you while the file dialog is still up, or after it was dismissed? With the OFNHookProc in place, you can provide your users with the most timely presentation possible.

As usual, you may download the Dialogs sample from my Web site for a fully coded demo of these techniques. It includes drop-in ready BAS modules for each supported common dialog, and a form that shows with a single Command button click routine how to call each.

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.