Visual Studio Magazine Online
Classic VB Corner

# Customizing the Ride (Part 1)

*There are countless tricks folks have learned to use within the classic Visual Basic IDE over the years. Here are a few of mine.*

April 6, 2010 · by Karl E. Peterson

Seems that contrary to Microsoft's best laid plans, I'm still installing Classic VB in machine after machine. It's really nuts to think of how many computers this language has outlived! And if you toss virtual machines into the mix, the count grows really impressive.

One of the dreary tasks we all face anytime we install software for the umpteenth time is reconfiguring its default settings to those we long ago became accustomed to. For example, in every VB5 or VB6 installation I use, I get into Tools-Options and toggle:

- Auto Syntax Check (off)
- Require Variable Declaration (on)
- Tab Width (3)
- Grid Units (60)
- Break on Unhandled Errors (on)
- Compile On Demand (off)
- Prompt to Save Changes (on)
- Background Project Load (off)

That's a bit to remember. Next thing I do is get rid of the Form Layout window. (Does anyone use that?) Then I add the Locals window, such that it and the Immediate window stretch across the entire IDE along the bottom. That requires repositioning the Toolbox window on the left, and the docked-together Project Explorer and Properties windows on the right.

Yikes! That's a lot of busy work, and it's probably only scratching the surface of toolbars, menus and other bits Classic VB developers have to arrange, etc. What if I told you there's an easy way to cache your settings, and reapply them to any and all future installations? Fire up your Registry Editor (regedit32), navigate to and export this key:

```
HKEY_CURRENT_USER\Software\Microsoft\Visual Basic\6.0
```

Next time you do a fresh install, just double click the REG file created in the step above, and all your cached settings will be merged back into the registry. You can do this for VB5 and VBA installations, as well, by (respectively) exporting these keys:

```
HKEY_CURRENT_USER\Software\Microsoft\Visual Basic\5.0
HKEY_CURRENT_USER\Software\Microsoft\VBA
```

These REG files not only store all the Tools-Options settings, but docking information, custom colors and much more. No reason to go through all that reconfiguration again!

## Missing IDE Elements

Every so often, someone will wander into the groups to ask how to "get something back" after mysteriously losing it. It's not always the same something. Sometimes, it's one of the IDE's toolwindows (eg, the Project Explorer). Sometimes, it's a toolbar button or some random menu item. There's never any good idea how it disappeared, but now that it's gone it's also seemingly very hard to Google for the right answers.

Here's one that works in a great number of cases. Right-click anywhere in the empty space that's used for toolbars, and select "Customize...". Highlight the suspected item, which more often than not is "Menu Bar", and press the "Reset..." button. You'll be prompted with a paternalistic "Are you sure?" msgbox. Unless you've done some fairly extensive mods, I can't see any reason not to be sure here, so just say Yes. Odds are, the missing functionality will now be back.

By the way, if it was a missing toolwindow, be sure to check the View menu that was just reset. There's probably an option to restore the toolwindow there, now. If the menu item is there, but selecting it still doesn't bring back the missing toolwindow, you may need to dig a little deeper into the registry.

It's possible that one of the two binary entries, Dock and/or Tool, found in the location(s) discussed above, has become corrupt. Make a new backup of that settings key, then delete those entries. All your toolwindows will now use their default positions the next time you start the IDE. If you don't like what you see, you can always restore what was there from your backup with a quick double-click.

## Encouraging Code Reuse

I have a very bad habit of writing code I can use over and over again. (This is bad, apparently, if you follow the Microsoft recommendation of rewriting functional code with some regularity. Kind of like buying a new wardrobe to match the season. ) And I'm betting that one of the most overlooked features in Classic VB is the ability to leverage the Templates Directory setting.

Over the years, I've provided probably close to a hundred or more "drop-in ready" code modules, most of which are still readily downloadable from my Web site. A great number of these have found their way into the various folders under my Templates folder. So if I'm working on a project, and I need to pick apart a UDT in memory, I just Add-Module-MHexDump.bas. I can instantly get a hex editor view of any area of my virtual address space.

If I'm wondering which of two or three approaches might be fastest, it's Add-Class Module-CStopWatch.cls. If I want to start a new console project, it's File-New Project-Console Application. Nothing could be easier. The only real trick is to follow the file system to your own custom Templates Directory folder. I tend to keep mine under a root level Code folder, and it's just about the first thing copied to a new dev machine when I'm setting up Classic VB.

As I alluded to, you're not limited to just adding frequently used modules to your templates collection. You can also add frequently used project types, such as console apps. You do this by just taking an existing VBP file, and modifying its contents to be as generic as possible. The common modules you'll want to pull in should be stored in the appropriate places under the templates folder. Here's the guts of my "Console Application.vbp" file:

```
Type=Exe
Reference=*\G{00020430-0000-0000-C000-000000000046}
Module=MMain; ..\modules\SubMainCon.bas
Module=Con; ..\modules\MConsole.bas
Class=CCmdLine; ..\Classes\CCmdLine.cls
Startup="Sub Main"
Command32=""
Name="Project1"
HelpContextID="0"
CompatibleMode="0"
MajorVer=1
MinorVer=0
RevisionVer=0
AutoIncrementVer=0
ServerSupportFiles=0
VersionCompanyName="http://vb.mvps.org/"
VersionLegalCopyright="Copyright ©2007-10 Karl E. Peterson"
CompilationType=0
OptimizationType=0
FavorPentiumPro(tm)=0
CodeViewDebugInfo=0
NoAliasing=0
BoundsCheck=0
OverflowCheck=0
FlPointCheck=0
FDIVCheck=0
UnroundedFP=0
StartMode=0
Unattended=0
ThreadPerObject=0
MaxNumberOfThreads=1

[vbAdvance]
IsConsole=1
HasStubFile=0
GenerateMap=0
ResBuildName=.\SubMain.dll
ReplaceIcon=0
SendCommandArgs=0
SymbDbgPref=0
TSAware=0
XPManifest=0
RevisionVersion=0
XPManifestUIAccess=0
XPManifestTrustLevel=0
```

Note that I can even include the vbAdvance section to make sure that when I compile, it'll indeed be a console application. When this project type is selected in the New Project dialog, all the associated BAS and CLS files are loaded, but not saved anywhere. You'll be prompted where to save them, the first time you save the project. This keeps the template copies pristine, but also means you need to remember to update the template copies if you tweak them in any project.

## 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.*