| | |
|---|---|
| Issue | January 2001 |
| Section | essential tips |
| Main file name | **vc0101Ett9a.rtf** |
| Screen capture file names | **vc0101ETf1.bmp** (use if we have space; no caption necessary; delete fig ref if we don't use the figure) |
| Notes to ME | Nick, I confirmed with author that "PS" can be used for "PS DLL." All instances are OK as-is.—NG |
| Special instructions for Art dept. | Please place the Send Your Tips box (pick-up from VCDJ March). |
| Editor | **EN** |
| Status | Tech reviewed(t1), edited (t2); ar(t3); screen cap removed (t4); final review by ng, rma, final revs showing (t5); jl review (t6); ng review, rma, spellcheck (t7); NF copyedit (t8); NG review (t9a) |
| EN review | t2 |
| Character count | |
| Package length | 1 page |
| ToC blurb | n/a |

Overline:

Essential Tips

Level: Beginner
VC++ 6.0

# *USE A TEMPLATE VC++ PROJECT FOR VISUAL AUTOMATED IDL BUILDS*

After writing your IDL file, you typically feed it to the Microsoft Interface Definition Language (MIDL) compiler to generate header and C files as source files for the proxy-stub (PS) DLL and type library. You then must provide a DEF file with the DLL entry points in it, and pass it all to a C compiler and linker to build the PS DLL. You should then register your PS. If you're using ATL, it runs MIDL on your IDL file and generates a makefile (*.mk) you can use to compile the MIDL-generated files into a PS DLL. You still must register the PS manually for each module in your application, then deploy and install all the proxies for your components.

Instead, you can use a Visual C++ project to automate the whole process while enjoying the benefits of a visual environment including the class wizard, the ability to copy files, and error messages that trace to your IDL code. Doing so, you can view the IDL files as input source files and output a registered PS DLL. The IDL project should look and behave like a normal project in VC++ (see Figure 1).

Why automate? For one, you don't want to trigger a massive PS DLL build and regenerate the header files if the IDL hasn't changed. For that, you need dependencies—something you take for granted in your other Visual C++ projects.

Another reason to automate: You can have the project perform auto-registration at the end of the PS compilation and copy the header files to other locations, reducing the likelihood of mistakes. If you forget to register the new PS DLL, all hell could break loose if you run with the old PS or old header and TLB files.

Need more reasons?

- Wizard support for adding new methods to the interfaces.
- You have, in one place, all the compilation warnings and errors for all IDL and C files. Error messages map directly to the IDL source.
- One PS for your product even if it consists of many modules. This means a smaller memory footprint and easier deployment (only one PS to install).
- One MIDL build environment—because you want all your IDL files compiled with the same MIDL switch set (such as /Oicf).
- Ease of use—one step (literally one mouse click) from IDL to a registered and deployed PS.
- You can redirect the outputs—copy the headers, the PS file, and the TLB to shared locations.
- You can have consecutive GUIDs. You run uuidgen.exe once with this command line: uuidgen.exe -n500 > guidlist.txt. Now you have 500 consecutive GUIDs for your project (Microsoft has all its GUIDs consecutively). Having your entire project in one place when you open the Registry makes your life a lot easier. In the IDL project, simply add the guidlist.txt to the project resources.
- The ability to use custom-named source files (dlldata.c is not a proper name).

Every PS is a COM DLL. The template project is actually a VC++ DLL project, whose DEF file contains the PS entry points and whose source files are MIDL outputs. All the IDL files have custom build steps that exclude them from the build, except one IDL file (the "main" one) that has a custom build step to run a complex MIDL command line to tweak MIDL as necessary. File-level dependencies provide VC++-like source file dependencies (if there's no change, there's no build).

View the MIDL-outputted C files as temporary files (like *.obj). You can have one IDL file per set of interfaces, and one IDL file (the "main" one) that #includes them all—just like in a C++ project—and has the CLSID definitions in the typelib's scope. To add a new IDL file, simply include it and add it to the dependency. You can download a template VC++ IDL project from the *VCDJ* Web site; see the Go Online box for details.

***—Juval Lowy***

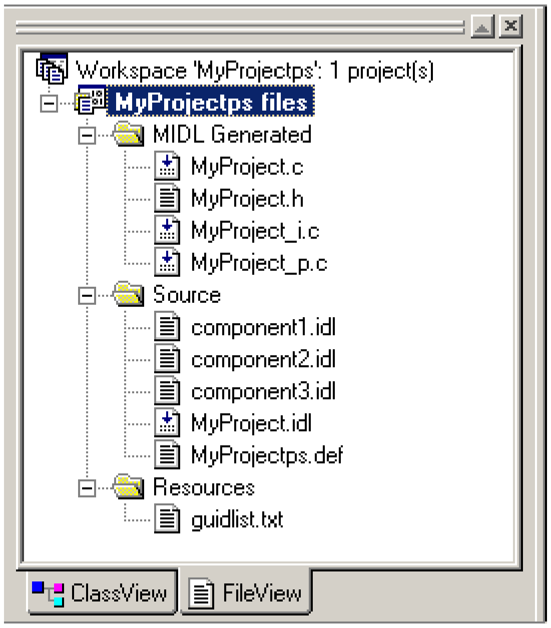
```
Workspace 'MyProjectps': 1 project(s)
  MyProjectps files
    MIDL Generated
      MyProject.c
      MyProject.h
      MyProject_i.c
      MyProject_p.c
    Source
      component1.idl
      component2.idl
      component3.idl
      MyProject.idl
      MyProjectps.def
    Resources
      guidlist.txt
```

***Figure 1***

## Go Online

Use these DevX Locator+ codes at www.vcdj.com to go directly to these related resources.

**VC0101** Download all the code for this issue of *VCDJ*.

**VC0101ET** Download the code for this article separately. This article's code includes a template VC++ IDL project.

**VC0101ET_T** Read this article online. DevX Premier Club membership is required.

**Want to subscribe to the Premier Club?** Go to www.devx.com.