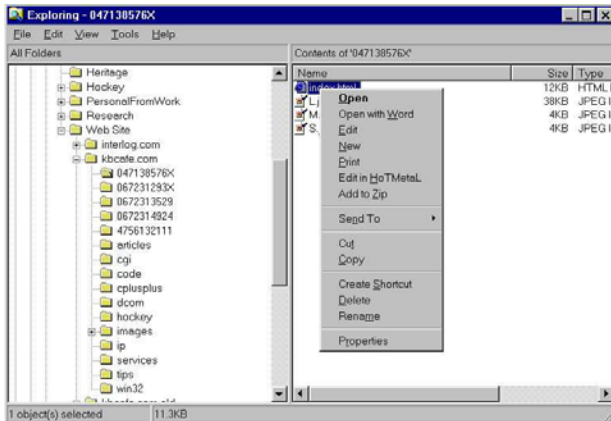


HowTo Write Simple Win32 Shell Extensions

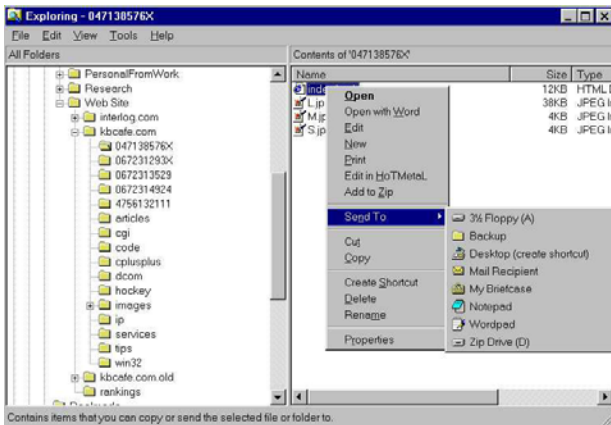
by Randy Charles Morin

Have you ever noticed how you can get a very rich context menu if you install a lot of software and right-click on HTML files in Windows Explorer. When I right-click on HTML files, I have seven options; Open, Open with Word, Edit, New, Print, Edit in HoTMetaL and Add to Zip.



Then if I select Send To, I'm presented with even more options; 3½ floppy, Backup folder, Desktop, Mail Recipient, My Briefcase, Notepad, Wordpad and Zip Drive. Why is there so many options when you right-click? Because they are so easy to create. Every software developer wants to add as many features to there products as possible. Write simple shell extension, is the fastest way to that end. In this article, I'll show you a few tricks I've mastered.

Let's start with the Send To menu. Adding new items to the Send To menu is trivial. In a users Profile folder, usually Winnt/Profiles/{username}, you'll find a folder titled SendTo. Anything in this folder will also appear in the Sent To menu when you right-click on a file in Windows Explorer. What I've got in the habit of doing is putting a shortcut to various folders and editors. This allows me to quickly copy files to a folder and open files with editors that are not typically associated with that file type.



The next easy task in creating shell extensions, is add items to the context menu of file types. A neat little trick is to add a shortcut to the command-line shell in the directory context menu. This is done by making a small change in the Windows Registry. Simply add a new subkey in HKEY_CLASSES_ROOT\Directory\shell. Here's .REG file that adds a CmdHere item to the context menu for folders.

```
REGEDIT4

[HKEY_CLASSES_ROOT\Directory\shell\CmdHere]
@="CMD &Prompt Here"

[HKEY_CLASSES_ROOT\Directory\shell\CmdHere\command]
@="C:\WINNT\System32\cmd.exe /k cd \"%1\""
```

I've extended this simple technique to provide for automatic registration of COM components and Service module. The next listen adds two commands to the context menu of DLLs and three to the context menu of EXEs.

```
REGEDIT4

[HKEY_CLASSES_ROOT\.dll]
@="dllfile"

[HKEY_CLASSES_ROOT\dllfile]
@="Application Extension"

[HKEY_CLASSES_ROOT\dllfile\shell\Self Register\command]
@="RegSrv32.EXE %1"

[HKEY_CLASSES_ROOT\dllfile\shell\Self Unregister\command]
@="RegSrv32.EXE -u %1"

[HKEY_CLASSES_ROOT\.exe]
@="exefile"

[HKEY_CLASSES_ROOT\exefile]
@="Application"

[HKEY_CLASSES_ROOT\exefile\shell\Self Register\command]
@="%1 /regserver"

[HKEY_CLASSES_ROOT\exefile\shell\Register Service\command]
@="%1 /service"

[HKEY_CLASSES_ROOT\exefile\shell\Self Unregister\command]
@="%1 /unregserver"
```

Note the indirection in this listing. The HKEY_CLASSES_ROOT\.exe folder is redirected to the HKEY_CLASSES_ROOT\exefile folder. This layer of abstraction is very standard and I recommend you follow it whenever possible.

Up to now I've presented you with very trivial shell extensions. The next extension is a little bit more complex. Now that I had a quick way to register COM DLLs and EXEs, I thought it would be great to have a context menu item that would register type libraries (TLBs). At the time, there wasn't even a command-line tool that did this, never mind a shell extension for one. So first I set out to create the command-line tool.

```
#include <windows.h>
#include <iostream>

int main(int argc, char * argv[])
{
    if (argc != 2)
    {
```

```

        std::cout << "Usage:\tRegTLB [-u] filename\n"
                  "\twhere filename is name of the TLB file\n"
                  "\twhere optional parameter -u unregisters "
                  "the file\n" << std::end;

        return 0;
    }

    ::CoInitialize(NULL);

    ITypeLib * typelib;
    HRESULT hresult = ::LoadTypeLibEx(AnsiToWide(lpCmdLine).c_str(),
                                     REGKIND_REGISTER, &typelib);
    if (FAILED(hresult))
    {
        std::stringstream ss;
        std::cout << "File = " << lpCmdLine << ". HRESULT = "
                  << hresult << std::endl;
        return 0;
    }
    typelib->Release();
    ::CoUninitialize();

    return 0;
}

```

Creating this small utility was trivial because the LoadTypeLibEx Win32 API function did most of what was required.

Creating the shell extension for this utility was also trivial. Following the template already outlined for EXEs and DLLs, I pointed a Register command to my new little utility. Note that the RegTLB utility must be in the path for this to work, as I did not specify a full path.

```

REGEDIT4

[HKEY_CLASSES_ROOT\.tlb]
@="tlbfile"

[HKEY_CLASSES_ROOT\tlbfile]
@="Type Library"

[HKEY_CLASSES_ROOT\tlbfile\shell\Register\command]
@="RegTLB.EXE %1"

```

The last shell extension I show you is a little bit more complex. In this final example, I wanted the ability to register and unregister Microsoft new RGS files. The RGS files are a much more powerful scripting language than the old styled REG files that I've been creating in this article. The Visual C++ Wizards also create many of these RGS files when you create new ATL projects and COM objects. Unfortunately, there was not utility that registered and unregistered RGS files, so I had to create one.

```

#include <iostream>
#include <windows.h>

int main(int argc, char * argv[])
{
    if (argc != 2 &&
        (argc != 3 || std::string("-u") == argv[1]) )
    {
        std::cout << "Usage:\tRegRGS [-u] filename\n"
                  "\twhere filename is name of the RGS file\n"
                  "\twhere optional parameter -u unregisters "
                  "the file\n" << std::end;

        return 0;
    }

    ::CoInitialize(NULL);
}

```

```
IRegistrar * registrar;
::CoCreateInstance(CLSID_Registrar, NULL, CLSCTX_ALL,
    IID_IRegistrar, (void **)&registrar);

if (argc == 2)
{
    registrar->FileRegister(AnsiToWide(argv[1]).c_str());
}
else
{
    registrar->FileUnregister(AnsiToWide(argv[1]).c_str());
}
registrar->Release();
::CoUninitialize();

return 0;
}
```

After doing a little research, I found that the IRegistrar interface and class, that is part of ATL has two methods that will do the job.

And finally the REG file to register our shell extensions.

```
REGEDIT4

[HKEY_CLASSES_ROOT\.rgs]
@="rgsfile"

[HKEY_CLASSES_ROOT\rgsfile]
@="RGS Extension"

[HKEY_CLASSES_ROOT\rgsfile\shell\Register\command]
@="RegRGS.EXE %1"

[HKEY_CLASSES_ROOT\dllfile\shell\Unregister\command]
@="RegRGS.EXE -u %1"
```

My next project was going to be to rewrite some of the previous REG files in RGS, but I doubt I'll ever spend that time. As you can see though, creating these shell extensions is very simple. So next time you create a new file type, you'll know how to create shell extensions for your editor and your viewer.

What I will do next is create a small utility to automatically FTP files to my webserver. I run a complex hockey pool that requires three people to maintain. The statistician of our pool wants the ability to add files to the site. Unfortunately, he's an accountant, not a programmer, so telling him about the FTP command-line utility or even an FTP GUI client is not really acceptable. What I'm going to do is create a small FTP client that simply logs into my site and uploads the file passed as the command-line parameter. I can then create a shortcut in his Send To folder and he can upload file by right-clicking on the file and selecting Sent To | KBcafe.COM. It should be cool. Stay tuned.

Later this month, I will return to the IP series and write a small article on how to NNTP. NNTP is the protocol your news reader uses to talk to a news server. See you then.