

How-to set up a Project with libXLL

1. Add new project with type "Win32 Dynamic Link Library". Name it e.g. "myExample" and choose "empty library" as we will do everything step by step. If you are already working with COM, or intend to so, start with a COM/ATL project, which already contains most of the settings below.
2. Add a new textfile e.g. "myExample.idl". This is going to be your "interface definition file", where you define (and hopefully document!) all the functions that you are going to export to Excel and VBA.
3. Add the xlw directory to your project's include path. Right-click on the "myExample" project, choose settings, choose "all configurations"
 - go to the "C/C++" tab, choose "preprocessor" and add the directory to the "additional include dirs"
 - go to the "Midl" tab and add the directory to the "additional include dirs"
4. While you are in the "Midl" tab, uncheck the "MkTypLib compatibility" option and change the output file name to e.g. "myExample.tlb" so it is not written to the build directories.
5. We also add the xlw library path to our project. Go to the "Linker" tab. Choose "inputs" and add e.g. ".\lib\Win32\Visual Studio" to the additional library path.
6. Copy the content of the original `xllExample.idl` to your `myExample.idl` file. Make sure it compiles without problems.
7. Now, and that is important everytime that you copy some code from any IDL example, change these strange hex numbers in the code. These globally unique identifiers are called UUIDs, and there is a tool in your VC6 distribution called `UUIDGEN.EXE` to make your own (its in `<root>\Common\Tools\UUIDGEN.EXE`). You may also want to modify the library, module and dll names as well as the associated descriptions. The `dllname()` should match your project name, eg. "myExample.dll"
8. Now we implement the functions. Add a new "C++ source file" e.g. "myExample.cpp" to your project. Start by copying the content of the original `xllExample.cpp`. Check if it compiles.

9. Now we must make sure that the typelib we created will be included in our .DLL as a resource. Add a new "resource script" e.g. "myExample.rc" to your project. Change to the resource view, right-click on your resources and choose "resource includes". Add

```
1 TYPELIB "myExample.tlb"
```

to the compile-time directives.

10. A final task remains. The functions exported from your DLL are not "visible" from outside. You can either solve this by adding a .DEF file to your project (e.g. "myExample.def") that contains a single "EXPORTS" line followed by one line with the name of every function you want to export. Or you can use the supplied `IDL2DEF.EXE` to generate it on-the-fly: Right-click your project, select "settings", choose "all configurations" and go to the pre-linker build tab. Enter a name e.g. "Generating .DEF file" for this step and enter e.g. the following as a single command line below:

```
type myExample.idl | ..\idl2def\idl2def > myExample.def
```

Build once and add the resulting .def file to your project.

11. To test your Add-In, copy it from the Debug or Release directory to `\WINNT\System32` and open Excel. We will start by testing it from Visual Basic. Press Alt-F11 to open the Editor, select the empty Project on the left, and click on "Extras", "References". Browse to `\WINNT\System32` and select your .DLL. There should be no error when you close the dialog. Press F2 to enter the object catalogue, and select your library (e.g. XLL_EXAMPLE). You can now browse your functions and constants. Right click and "show hidden" if anything is missing ;-). Now run

```
Debug.Print xllCirc(1)
```

The output should be quite familiar.

12. Now we try the same from the worksheet. Close the VB editor and go to "Extras", "Add-In manager". Browse to `\WINNT\System32\`, select `myExample.dll` and it should become available in your add-in list. Go to an empty worksheet cell and start the function wizard. Your exported functions should be available in their respective categories.