Modern XLL Development

XLLs remain one of the best ways of creating fast, powerful and easily distributable Add ins for Office. Leveraging the power of C and C++, they have the most direct interface with Excel via the C api. They are however more obscure, difficult to develop and most resources seem to be about ten ears old or more as of writing (2015/01/13)

What are XLLs?

XLLs are DLLs which implement a specific interface which is required in order by Excel in order for the Add in to be loaded. Via the C API provided in the Excel SDK, DLLs can call Excel functionality.

DLL is short for **Dynamic Link Library** which is a way of storing code so that it can be easily distributed and shared by othee applications. Functions and variables defined in DLLs can be exported by the DLL and reused by other applications. It encourages code modularity and reuse.

XLLs themselves are now quite an old concept. The **Excel Macro Language** (XLM) used to be the primary way to access Excel functionality. Since then, the Excel C API has been been mapped to from XLM and the documentation for XLM can sometimes still be useful for controlling Excel via XLLs. This does mean that the C API refers mainly to older versions of Excel and some of the new functionality is not exposed via it.

Therefore, XLLs tend to be used mainly for **User Defined Functions** (UDFs) because of the performance C and C++ can offer when doing more expensive calculations.

The crux of the matter, however, is that the age of the C API shows. There isn't anything I can do about it, but much of the documentation is old or makes assumptions about prior knowledge. I will do the best I can to provide an introduction to the practical application the Excel SDK for a modern Add in.

Getting Started

There is, of course, exisiting documentation of the SDK and API and it is not my aim to replace those elements. However, much of the resources are old, and gloss over technicalities in the application process. It is here that this guide will attempt to fill the gaps. Thus, I will attempt to supply the readeer with as many of the sources that helped me as I can. There is of course the entry point for the SDK, which is a good place to start but there are also tutorials and buried forum threads which contain useful tidbits for any new XLL developer.

After some background reading (or maybe before so it's done by the time you finish!), you will need your working environment, be it an IDE or a text editor

and the commandline. I am using **Visual Studio Professional 2013** (VS). I'm certain the community edition would also work. The key elements are a text editor and C/C++ compiler. I will go no further.

Secondly, download the 2013 XLL SDK (it's not too large). On inspection, you'll find many of the files are straight from the 2007/2010 versions of the SDK.

The next step to building your first XLL is a combination of an adaptation of the instructions found here and a painful struggle with the VS environment.

Essentially, become familiar with using C++ in the VS environment and research the Excel Object Model (it's the same as in VBA and XLM, just translated).

It is possible to follow the tutorials written by Andrew Whitechapel and to build the projects in the "Samples" folder, it's just necessary to avoid some gotchas, update the conventions and workaround some defficiencies.

For instance, `GENERIC.H` needs the addition of the line `#include <windows.h>` in order to compile. `GENERIC.C` itself needs to include `GENERIC.H` with `"..."` instad of `<...>`.

1. Download and install VS2013 (most other should work too)
2. Download and install Excel XLL API SDK installer (msi file) in any convenient location.
3. In we are going to build frmwrk32.lib which is used to build other xll projects. Open `.../Excel2013XLLSDK/SAMPLES/FRAMEWRK/Framework.sln` in VS. VS might automatically make the project compatible with the version you are using and show the results in an html file. You can safely ignore this as far as I know.However, the project is not ready to build.
4. In the VS toolbar, select `PROJECT/Framework Properties`. This will open a menu. Navigate to `Configuration Properties/VC++ Directories` in the left pane and edit `Include Directories` to include `.../Excel2013XLLSDK/INCLUDE` and `.../Excel2013XLLSDK/SRC`
5. In `FRAMEWRK.C` replace chevrons `<>` with `""` in the following lines. This is so that the compiler looks in the local directory for the files. `#include <framewrk.h>  #include <memorymanager.h>`
6. It should now compile. Try it and there should be a new file `frmwrk32.lib` in `.../Excel2013XLLSDK/LIB`
7. Now we can create a minimal version of Andrew Whitechapel's project. Open a new instance of Visual Studio and follow Andrew's instructions, watching out for the following pitfalls
8. After naming your project and pressing `OK`, in the wizard press `Next >`and select `DLL` under Application type. There is no need to tick `Export Symbols` under Additional options even though the xll will be exporting symbols.
9. Follow the instructions. If you can't find the correct menu items visual studio, you may need to:
    a. Restart VS in compatability mode.

b. Have some C++ files in the project in order to enable the `C/C+ +` submenu option in the Property pages. It can be worth following the Configuration isntrcutions and then deleting the files.
c. Rather than changing `Linker ¦ General ¦ Output file` settings, I changed `Configuration Properties ¦ General ¦ Target Extension` from .dll to .xll. I don't know if this is better of worse. It looks equivalent at worst and a more robust method otherwise.
d. Make sure to put `xlcall32.lib` and `frmwrk32.lib` on seprate lines! I didn't and it screwed me up.
e. The final step is optional. It throws a warning about a .pdb file because Excel doesn't care about your debugging, but you can still set breakpoints and can ignore the warning.

10. Compile and it seems to work! I changed the names of