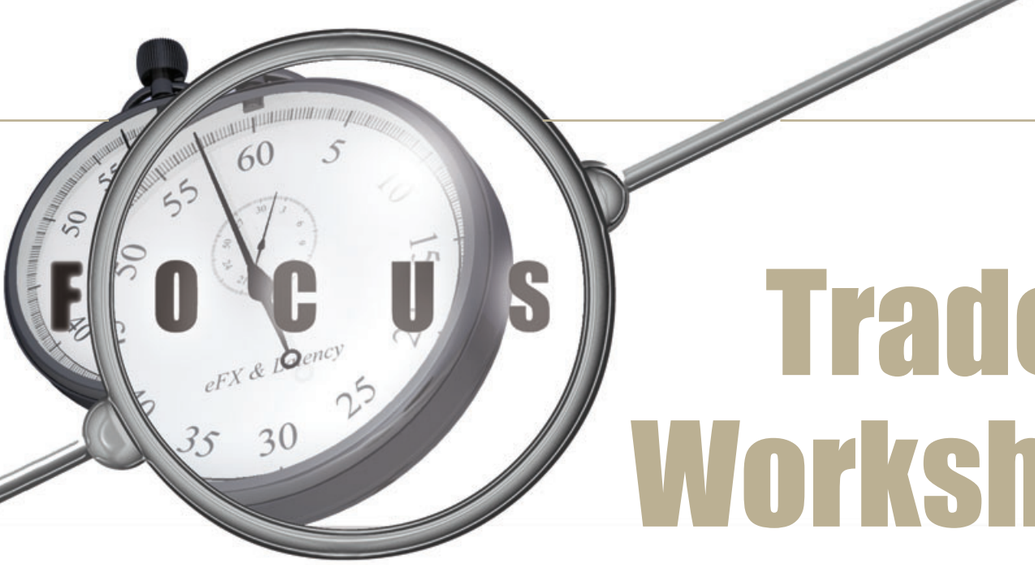


Traders Workshop

Latency has been an increasingly hot topic in FX of late, with several data vendors recently announcing lower latency versions of their feeds. However, any gain in delivery speed is all too easily negated by the latency of the applications that process the data. Andy Webb outlines one way of addressing this latency issue for one of the most ubiquitous applications in financial markets - Microsoft Excel[1].

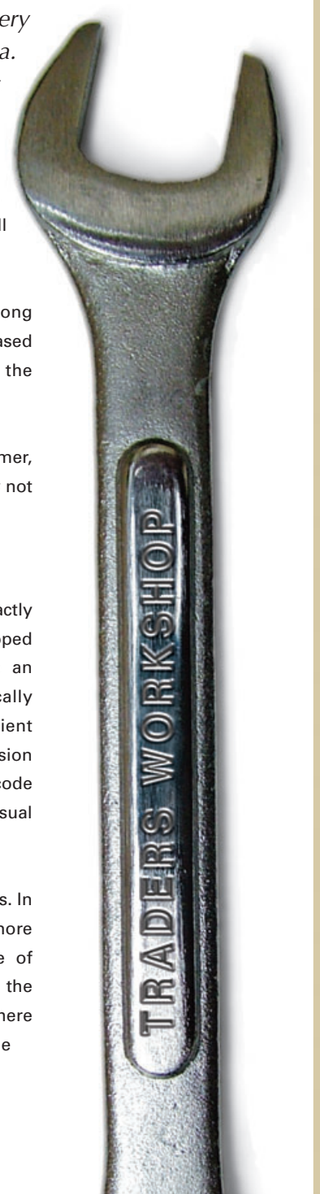
In finance, as in so many other areas of human endeavour, ease of use usually comes with a compromise price tag attached. In some cases that compromise is restricted functionality or flexibility - in others it is speed. Excel is a good example of the latter case. While it allows those with minimal programming expertise to achieve productivity, its performance can be less than ideal for production use in highly active markets such as FX.

An obvious example of this problem is exotic FX options, where many traders and quants use Excel for prototyping and testing pricing models. While performance issues are no great matter (other than tedium and productivity) at the prototyping stage, it is a very different matter when the same spreadsheet platform is also used in a production environment. Particularly where a spreadsheet contains a matrix of option prices, strikes and expiries that have to recalculate in real time, its calculation latency can become a matter of competitive disadvantage. In theory such a spreadsheet should not of course be used in a production environment, but there are a number of reasons why it may in practice:

- Time to market – in many cases the builder of the original spreadsheet is unlikely to have lower level (e.g. C++) programming skills. To convert the spreadsheet business logic to a more efficient form such as a Dynamic Link Library (DLL) or Excel C++ add-in (XLL), they will have to explain the methodology to a programmer. This can take a considerable time and will probably be a tediously iterative process.

- Errors – unless the programmer has strong quantitative understanding there an increased risk of errors being introduced during the translation process.

- Resources – a suitably qualified programmer, who could convert the spreadsheet, is simply not available.



Automation and process

It was in response to a client confronting exactly this type of situation that Savvysoft developed TurboExcel[2]. The objective was to produce an application that was capable of automatically converting a spreadsheet to a more efficient format, such as a DLL or XLL file. The conversion process had to be able to handle both the code worksheet cells and any associated VBA (Visual Basic for Applications) code.

This represents two entirely separate challenges. In some respects, translating VBA to C++ is the more straightforward of the two, as the sequence of translation can follow the same sequence as the original VBA with a line-by-line conversion. There are one or two important nuances – for example C++ syntax can differ dramatically from VBA, which necessitates a quite intelligent parser. Some additional helper functions are also necessary in order to catch such errors as divide by zero. Spreadsheet conversion is a more daunting task for an application like TurboExcel (as it is for human programmers given the unenviable task of converting a spreadsheet to C++), as it needs to figure out the correct order to write out the C++ code based on the implicit relationships among the cells.

Another major task is the need to replicate all built-in Excel worksheet and Analysis ToolPak functions, as well as VBA's rich object model. This is necessary because if the original source code is translated into a C++ DLL that is called from an application other than Excel then the original native Excel functions will not be available.

From the user's perspective, the translation of VBA requires little more than inputting the names of the subroutines or functions to be converted and the desired name of the output DLL or add-in. However, since spreadsheets don't have nice function declarations like VBA, the user who wants to convert spreadsheets to C++ is required to define the range of input cells that drive the calculation to the final output cell(s), and their associated data types.

Speed

Compiled C++ code obviously runs far faster than VBA or code entered directly into worksheet cells. This speed advantage becomes increasingly apparent as the code complexity increases. This is particularly significant for FX option traders who are most likely to be suffering extensive recalculation times if they are using Excel/VBA in a production environment. Using a DLL or XLL instead should therefore significantly enhance their response times and edge.

However, other types of FX traders can also benefit from compiled VBA/worksheet formulae. Proprietary traders using complex algorithms for mechanical/automated spot trading at high trade frequencies in short time frames are very much at the mercy of execution speed. A few milliseconds can make the difference between profit and loss. Again, using compiled versions of trading system logic can recoup those milliseconds.

The speed improvements possible from using compiled Excel/VBA code vary enormously, but it is not unusual to see calculation times reduced by a factor of three hundred or so. In certain cases the gains can be far greater. For example, if Excel is being used as COM server and receiving calculation calls from other applications, then compiling the calculation logic and calling the resultant DLL directly will result in speed gains of several thousand times.

The bigger picture

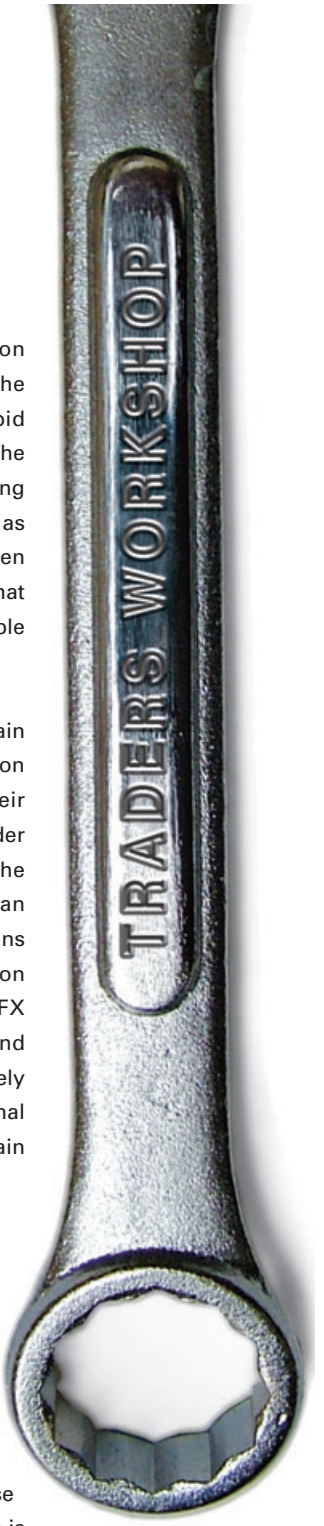
The immediate speed benefits of automated code translation and compilation are self-evident, but there is a broader potential gain in terms of workflow enhancement. Consider the FX trader who has to interact with multiple systems. The initial step in their workflow might be to calculate a figure in a spreadsheet and then re-key that into another application, take the output from that application and re-key to another spreadsheet, recalculate that second spreadsheet and take a trading decision based upon the output.

That entire process may take several minutes to complete. One alternative might be to run the initial worksheet through an automated translator/compiler such as TurboExcel to produce a DLL that also incorporates a call to the external application. Then translate/compile the second spreadsheet as an XLL or DLL incorporating a call to the DLL produced by compiling the first worksheet. This will obviously result in faster code execution, but also in far faster workflow and reduced operational risks (no re-keying errors).

Automated spreadsheet translation/compilation can also be used to enhance the functionality of core FX systems. For example, an FX option desk's main trading application may well not be able to interact directly with proprietary models running in a spreadsheet. (If it can it will probably be by using Excel as a COM server – an inherently inefficient and unreliable approach). On the other hand, such an application is far more likely to be able to make function calls directly to a DLL.

[1]Microsoft Excel is a registered trademark of the Microsoft Corporation in the United States and/or other countries.
[2] http://www.turboexcel.com

Traders Workshop



Development environment and enterprise gains

Automated translation/ compilation technology effectively turns the spreadsheet into a user-friendly rapid application development environment. The whole notion of structured programming also becomes immediately available, as spreadsheet functionality can be broken out into a series of discrete DLL/XLLs that can be reused and recombined for multiple purposes.

FX traders or quants effectively also gain instant C++ expertise without any effort on their part. While this enhances their individual productivity, it also has wider implications. The fact that the translation/compilation is done in an automated and consistent manner means that an extensible and robust function library becomes available to the FX enterprise as a whole. Maintaining and enhancing that library becomes a relatively trivial and error-free matter, as the original builders of models will not have to explain their changes to a programmer.

Finally, in a market environment with increasing emphasis upon control and transparency, automated code translation/compilation has a number of advantages. For example, if only the DLL or XLL is distributed, the risks of intellectual property theft are greatly reduced – a DLL is far harder to reverse engineer than Excel password protection is to crack. At the same time, an auditor is far more likely to understand and sign off on a model where the business logic is readable from a spreadsheet, yet where the production deployment (a DLL or XLL) is inherently secure and the process of translation is consistent.

An application such as TurboExcel does just one thing: converts Excel spreadsheets and VBA to C++ DLLs and add-ins. But the implications of this, and its myriad uses, are much more far-reaching in terms of latency reduction and productivity enhancement.