1 Introduction

1.1 TYPOGRAPHICAL AND CODE CONVENTIONS USED IN THIS BOOK

To distinguish between the text of the book, Visual Basic code, C/C++ code, and Excel worksheet functions, formulae and cell references, the following fonts are used throughout:

Excel functions and formulae
Windows application menus and control button text
`Visual Basic code`
`C/C++ code`
`Directory paths, file names and file masks`

Passages of source code appear as boxed text in the appropriate font.

The spelling and grammar used throughout this book are British Isles English, with the occasional US variation such as *dialog*.

Examples of non-VB code are mostly in C++-flavoured C. That is, C written in C++ source modules so that some of the useful C++ features can be used including:

- the declaration of automatic variables anywhere in a function body;
- the use of the `bool` data type with associated `true` and `false` values;
- the use of call-by-reference arguments;
- C++ style comments.

C functions and variables are written in lower case with underscores to improve readability, for example, `c_thing`. In the few places C++ classes are used, class and instance names and member functions and variables are written in proper case, and in general, without underscores, for example, `CppThing`. Class member variables are prefixed with '`m_`' to clarify class body code. Beyond this, no coding standard or variable naming convention is applied. Names of XLL functions, as registered with Excel, are generally in proper case with no underlines, to distinguish them from Excel's own uppercase function names, for example, `MyAddInFunction`.

Where function names appear in the book text, they appear in the appropriate font with trailing parentheses but, in general, without their arguments. For example, a C/C++ function is written as `c_function()` or `CppFunction()` and an Excel worksheet function is written as Excel_Function(). VB functions may be written as `VB_Function()`, or simply `VB_Function` where the function takes no arguments, consistent with VB syntax.

Code examples mostly rely on the Standard C Library functions rather than, say, the C++ Standard Template Library or other C++ language artefacts. Memory allocation and release use `malloc()`, `calloc()` and `free()`, rather than `new` and `delete` or the Win32 global memory functions. (There are a few exceptions to this.) This is not because the choice of the C functions is considered better, but because it is a simple common denominator. It is assumed that any competent programmer can alter the examples given to suit their own preferences. String manipulation is generally done with the standard C library functions such as `strchr()`, rather than the C++ `String` class. (There is some discussion of `BSTR` strings and the functions that handle them, where the topic is interoperability of C/C++ DLLs and VB.)

The standard C library `sprintf()` function is used for formatted output to string buffers, despite the fact that it is not type-safe and risks buffer overrun. (The book avoids the use of any other standard input/output routines.)

The object oriented features of C++ have mostly been restricted to two classes. The first is the `cpp_xloper`, which wraps the basic Excel storage unit (the `xloper`) and greatly simplifies the use of the C API. The second is the `xlName` which simplifies the use of named ranges. (Strictly speaking, defined names can refer to more than just ranges of cells.) There are, of course, many places where an add-in programmer might find object-abstraction useful, or the functionality of the classes provided in this book lacking; the choice of how to code your add-in is entirely yours.

C++ *throw and catch* exception handling are not used or discussed, although it is expected that any competent C++ programmer might, quite rightly, want to use these. Their omission is intended to keep the Excel-related points as the main focus.

Many other C++ features are avoided in order to make the code examples accessible to those with little C++ experience: namespaces, class inheritance and friends, streams and templates. These are all things that an experienced C++ programmer will be able to include in their own code with no problem, and are not needed in order to address the issues of interfacing with Excel.

The C++ terms *member variable* and *member function*, and their VB analogues *property* and *method*, are generally used in the appropriate context, except where readability is improved.

Throughout the book, where information is Excel version-specific, the version to which it applies is sometimes denoted as follows: [v11−] for versions up to and including 11 (Excel 2003); [v12+] for versions 12 (Excel 2007) and later; and so on. (See section 1.3 below).

1.2 WHAT TOOLS AND RESOURCES ARE REQUIRED TO WRITE ADD-INS

Licensed copies of a 32-bit version of Excel and a 32-bit Windows OS are both assumed. (16-bit systems are not covered by this book). In addition, and depending on how and what you want to develop, other software tools may be required, and are described in this section. Table 1.1 summarises the resources needed for the various levels of capability, starting with the simplest.

Table 1.1 Resources required for add-in development

What you want to develop	Required resources	Where to get them
VBA macros and add-ins	VBA (for Excel)	Supplied with Excel
Win32 DLLs whose functions can be accessed via VB	VBA A compiler capable of building a Win32 DLL from the chosen source language (which does not have to be C or C++)	Supplied with Excel Various commercial and shareware/freeware sources
C/C++ Win32 DLLs whose functions can be accessed via VB and that can control Excel using OLE/COM Automation	VBA A C/C++ compiler capable of building Win32 DLLs, and that has the necessary library and header file resources for OLE COM Automation	Supplied with Excel Various commercial and shareware/freeware sources. Microsoft IDEs provide these resources. (See below for details.)
C/C++ Win32 DLLs that can access the Excel C API whose functions can be accessed directly by Excel without the use of VBA.	A C/C++ compiler capable of building Win32 DLLs. The C API library and header files. A copy of the XLM (Excel 4 macro language) help file. (Not strictly required but a very useful resource.)	Various commercial and shareware/freeware sources. Downloadable free from Microsoft at the time of writing. (See below for details.) Static library also shipped with Excel.
.NET add-ins and controllers.	Excel 2002 or later. A C/C++/C# compiler capable of building .NET components for Microsoft Office applications.	

At the time of writing, a good starting point for locating Microsoft downloads is www.microsoft.com/downloads/search.asp.

1.2.1 VBA macros and add-ins

VBA is supplied and installed as part of all 32-bit versions of Excel. If you only want to write add-ins in VB, then that's all you need. The fact that you are reading this book already suggests you want to do more than just use VBA.

1.2.2 C/C++ DLL add-ins

It is, of course, possible to create Win32 DLLs using a variety of languages other than C and C++. You may, for example, be far more comfortable with Pascal. Provided that you can create standard DLLs you can access the exposed functions in Excel via VB. If this is all you want to be able to do, then all you need is a compiler for your chosen language that can build DLLs.

Chapter 4 *Creating a 32-bit Windows (Win32) DLL using Visual C++ 6.0 or Visual Studio .NET*, page 89, contains step-by-step examples of the use of Microsoft's *Visual Studio C++ version 6.0 Standard Edition* and *Visual Studio C++ .NET 2003* integrated development environments (IDEs). The examples demonstrate compiler and project settings and show how to debug the DLL from within Excel. No prior knowledge of these IDEs is required. (Standard Win32 DLLs are among the simplest things to create.) Other IDEs, or even simple command-line compilers, could be used, although it is beyond the scope of this book to provide examples or comparisons.

1.2.3 C/C++ DLLs that can access the C API and XLL add-ins

If you want your DLL to be able to access the C API, then you need a C or C++ compiler, as well as the C API library and header file. The C API functions and the definitions of the data types that Excel uses are contained in the library and header files `xlcall32.lib` and `xlcall.h`. The pre-Excel 2007 versions of these files[1] are contained in a sample project, downloadable from Microsoft at the time of writing, free of charge, at download.microsoft.com/download/platformsdk/sample27/1/NT4/EN-US/Frmwrk32.exe. It is also possible to link Excel's library in its DLL form, `xlcall32.dll`, in your DLL project, removing the need to obtain the static `.lib` version. This file is created as part of a standard Excel installation. Another approach is to create the `.lib` file from the `.dll` file, as discussed in section 5.1.

An XLL add-in is a DLL that exports a set of interface functions to help Excel load and manage the add-in directly. These functions, in turn, need to be able to access Excel's functionality via the C API, if only to be able to register the exported functions and commands. Only when registered can they be accessed directly from the worksheet (if functions) or via menus and toolbars (if commands). The C API is based on the XLM (Excel 4 macro language). This book provides guidance on the most relevant C API functions in Chapter 8. However, for a full description of all the C API's XLM equivalents you should ideally have a copy of the XLM help file, `Macrofun.hlp`. This is, at the time of writing, downloadable in the form of a self-extracting executable from Microsoft at download.microsoft.com/download/excel97win/utility4/1/WIN98/EN-US/Macrofun.exe.

1.2.4 C/C++/C# .NET add-ins

This book does not cover .NET and C#. These technologies are an important part of Microsoft's vision for the future. The resources required to apply these technologies are Visual Studio .NET and a .NET-compatible version of Excel, i.e., Excel 2002 and later.

[1] At the time of writing, Microsoft plan to release an updated Framework project, although details of where and how this can be obtained are not known.

The principle purpose of this book is to bring the power of compiled C and C++ to Excel users, rather than to be a manual for implementing these technologies.

1.3 TO WHICH VERSIONS OF EXCEL DOES THIS BOOK APPLY?

Table 1.2 shows the marketing names and the underlying version numbers to which this book applies. Excel screenshots in this book (worksheets, dialogs, etc.) are mostly Excel 2000. Most of the interface differences between versions 2000 and 2003 are quite superficial. In contrast, the interface changes introduced in Excel 2007 are significant. The workbooks on the CD ROM are provided in both Excel 2000 and Excel 2007 format. (Contact ccppaddin@eigensys.com if you require 97 format files.)

Table 1.2 Excel version numbers

Product name	Version number
Excel 97 (SR-1, SR-2)	8
Excel 2000	9
Excel 2002	10
Excel 2003	11
Excel 2007	12

In some places, particularly in code examples, where information is Excel version-specific, the version to which it applies is denoted as follows: *v11–* for versions up to and including Excel 2003; *v12+* for versions Excel 2007 and later; and so on.

1.4 THE FUTURE OF EXCEL: EXCEL 2007 (VERSION 12)

At the time of writing, Excel 2007 (version 12) had only been released in beta. Whilst every effort has been made to ensure that what is written about it in this book is accurate, it is possible that the way some things work might be changed between beta and final release.

1.4.1 Summary of key workbook changes

The Excel team at Microsoft have made significant changes in many areas that are outside the scope of this book. As far as the subject matter of this book is concerned, however, the key changes are these:

- The size of the worksheet grid is expanded from 256 (2^8) to 16,384 (2^{14}) columns and from 65,536 (2^{16}) to 1,048,576 (2^{20}) rows, so from 2^{24} to 2^{34} cells – over 1,000 times as many.
- The maximum number of arguments a function can take is increased from 30 to 255.
- The level of function nesting in Excel worksheet formulae is increased from 7 to 64. (The author has some reservations about this being a good thing.)

- Multi-threaded workbook recalculation is supported on single- and multi-processor machines.
- The C API, XLL add-ins are still fully supported and are, for the first time in a very long while, upgraded to take advantage of some of the new features. In particular the Excel 2007 C API supports:
  - UNICODE strings up to 32Kbytes in length (in addition to byte-strings up to 255 bytes in length);
  - Larger grids;
  - More function arguments;
  - Multi-threaded recalculation;
  - Direct access to new worksheet functions.
- The user interface changes quite dramatically, providing applications developers and ordinary users with a much richer set of tools to control the appearance and behaviour of their workbooks, albeit at the expense of some familiarity.
- There are significant changes to the conditional-formatting capabilities. (See section 2.12.7 on page 40).
- Management of defined names is made much easier with improved interfaces.
- There are many new worksheet functions that should enable simplification of the more cumbersome data management, error handling and lookup tasks, e.g., IFERROR().
- The Analysis Toolpak worksheet functions are fully integrated into Excel and are also available directly via the C API.

Note that VBA and Automation add-ins will still not be able to take advantage of multi-threaded recalculation.

1.4.2 Aspects of Excel 2007 not covered in this book

Outside the scope of this book are the other changes that Excel 2007 introduces, in particular the radically different user interface through which built-in or custom commands are made available. Customising the new UI presents very different problems and issues than it did in previous versions, and where this book discusses these matters it does so only in relation to earlier versions of Excel.

1.4.3 Excel 2007 file formats

While still supporting the older file binary file formats (BIFF5 and BIFF8) and version 11 XML formats, Excel 2007 introduces a number of new formats and extensions:

- .XLSX – the XML-based default for code-less workbooks;
- .XLSM – the XML-based format for workbooks with VBA or XLM code;
- .XLSB – the new binary format (BIFF12);
- .XLAM – the XML-based add-in format (analogous to the .XLM of previous versions).

1.4.4 Compatibility between Excel 2007 and earlier versions

As stated above, Excel 2007 supports earlier versions' file formats for backwards compatibility, and contains a Compatibility Checker, which can be configured to run whenever a binary format file is saved, to check for elements not supported in earlier versions. VBA is

still supported in Excel 2007 and the object model is largely unchanged so that most VBA code in Excel 2003 and earlier workbooks should be expected to run without problems.

Compiled add-ins that are simply DLL's accessed via VBA (see section 4.11 *Accessing DLL functions from VB* on page 108) should run identically provided that they are not calling back into Excel via the C API or COM, in which case there are some cross-version compatibility issues covered in later parts of this book. XLL add-ins compiled with the old Excel SDK will work with Excel 2007 but again there are some compatibility issues, particularly where older add-ins customise the UI or call, say, Analysis Toolpak functions using `xlUDF`. VBA and compiled add-in code should therefore be modified to be version-sensing and -specific where these compatibility issues arise. XLL add-ins that rely on availability of Excel 2007 data types and C API, so that they can take advantage of larger grids and Unicode strings for example, will not be compatible with earlier versions of Excel. Sections 8.6.12 *Registering functions with dual interfaces for Excel 2007 and earlier versions* on page 263 and 9.13.3 *Making add-in behaviour Excel version-sensitive and backwards-compatible* on page 432 describe how to create XLLs that will run happily with Excel versions 11− and 12+.

1.5 ABOUT ADD-INS

An add-in is simply a code resource that can be attached to a standard application to enhance its functionality. Excel is supplied with a number of add-ins that can be installed according to the user's preference and need. Some provide specialist functions not needed by the average user, such as the Analysis ToolPak (sic) (whose functions are integrated into Excel in Excel 2007), and some that provide complex additional functionality such as the Solver add-in.

Add-ins come in two main flavours: interpreted macros and compiled code resources. Version 4 of Excel introduced macro sheets which could contain macros written in the Excel macro language (XLM). These comprised columns of instructions and calculations that either led to a result being returned to the caller, if functions, or that performed some action such as formatting a cell, if commands. Macro sheets could be part of a workbook or saved and loaded separately so as to be accessible to any workbook. Despite their flexibility they were relatively slow and did not promote sensible structured coding. In fact they encouraged the exact opposite given that, for example, they could modify themselves whilst executing.

Version 5 introduced Visual Basic worksheets. This enabled coding of functions and commands as before but promoted better coding practices and made implementation of algorithms from other languages easier. Excel 97 replaced these VB sheets with Visual Basic for Applications and the Visual Basic Editor (VBE) – a comprehensive IDE complete with context-sensitive object-oriented help, pre-compiler, debugger and so on.

Macros, be they XLM or VB, are interpreted. When run, the interpreter reads each line one-by-one, makes sense of it while checking for errors in syntax, compiles it and only then executes the instructions. Despite the fact that VBA does some of this work in advance, this is a slow process. The VBA approach avoids the need for tools to create fully pre-compiled code making the creation of add-ins possible for the non-expert programmer. VBA makes Excel application objects accessible and is therefore the obvious choice for a host of user-defined commands and functions where speed of development rather than speed of execution is the prime concern. Until Excel 2007, Microsoft had not updated the

C API since the release of Excel 97 and only support XLM for backwards compatibility. Even within Excel 2007 most of the new functionality and objects added since Excel 97 are only available to applications that can access Excel's COM-exposed objects. This is not too serious as the type of functionality added is that which it is most appropriate to access from VBA (or VB), rather than via the C API, anyway.

The other main flavour of add-in is the pre-compiled code resource which has none of the execution overhead of interpreted languages and is therefore extremely fast by comparison. The cost is the need to use, and so understand, another development language and another compiler or IDE. In essence, this is no harder than using VBA and the VB editor. The additional requirement is to know what Excel expects from and provides to anything calling itself an Excel add-in. In other words, you need to understand the Excel interface. The two interfaces that have been available over recent years are the C API and COM (the Common Object Model also known as Automation). COM provides access to Excel's exposed objects, their methods and properties. VBA itself is a COM Automation application. Section 9.5 *Accessing Excel functionality using COM/OLE automation using C++*, on page 376, discusses some very basic COM concepts.

VBA macros can be saved as Excel add-ins with very little effort but the resulting code is still slower than, say, compiled C add-ins. (Some performance comparisons are given in section 9.2 *Relative performance of VB, C/C++: Tests and results* on page 369). Despite the rapid development and flexibility of VBA, it lacks some of the key language concepts present in C and C++, in particular, pointers. These are sometimes critical to the efficient implementation of certain algorithms. One example of where this is especially true is with the manipulation of strings.

The advent of .NET changes a number of things. For example, VB code resources can be compiled and the functions contained made accessible directly from a worksheet, at least in Excel 2002 and later. C, C++ and C# resources can similarly be accessed directly from a worksheet without the need to use the C API.

1.6 WHY IS THIS BOOK NEEDED?

For anyone who decides that VBA just isn't up to the task for their application or who wants to decide the best way to make an existing C or C++ code resource available within Excel, just the task of weighing up all the options can at first seem daunting. At the publication of the first edition of this book, there were *no* published texts written specifically to help someone make this decision and then follow it through with practical step-by-step guidance. There are a number of commercial products that enable developers to access the power of Excel via the C API indirectly, through some sort of managed environment and set of classes. These are beyond the scope of this book, but do make sense for certain kinds of project.

The Excel C API is documented in Microsoft's *Excel 97 Developer's Kit* (1997, Microsoft Press), out of print at the time of writing. *This* book tries to complement that text as far as possible, providing information and guidance that it lacks. Where they overlap, this book tries to present information in a way that makes the subject as easy as possible to grasp. The *Developer's Kit* is a revision of an earlier version written for the 16-bit Excel 95, and contains much that was only relevant to developers making a transition from 16-bit to 32-bit. It provides a very comprehensive reference to the Microsoft BIFF (binary interchange file format) which is, however, of little use to most add-in writers.

Writing Win32 DLLs is fairly straightforward, but it is easy to get the impression that it is highly technical and complex. This is partly because available literature and articles often contain much that is no longer current (say relating to 16-bit versions of Windows), or because they concentrate heavily on 16- to 32-bit transition issues, or are simply badly written. Having said that, there are a few complexities and these need to be understood by anyone whose add-ins need to be robust and reliable. Overcoming the complexities to speed up the creation of fast-execution add-ins in C and C++ is what *this* book is all about.

1.7 HOW THIS BOOK IS ORGANISED

The book is organised into the following chapters:

Chapter 2 *Excel Functionality*
Basic things that you need to know about Excel, data types, terminology, recalculation logic and so on. Knowing these things is an important prerequisite to understanding subsequent chapters.

Chapter 3 *Using VBA*
Basic things about using VBA: creating commands and functions; accessing DLL functions via VB; VB data types; arrays and user-defined data structures, and how to pass them to DLLs and return them to Excel.

Chapter 4 *Creating a 32-bit Windows (Win32) DLL Using Visual C++ 6.0*
How to create a simple Win32 DLL, in VC or VC++ .NET, and export the functions so they can be accessed by VB, for example. Lays the foundation for the creation of XLLs – DLLs whose functions can be accessed directly by Excel.

Chapter 5 *Turning DLLs into XLLs: The Add-in Manager Interface*
How to turn a DLL into an add-in that Excel can load using the add-in manager: an XLL. The functions that Excel needs to find in the DLL. How to make DLL functions accessible directly from the worksheet.

Chapter 6 *Passing Data between Excel and the DLL*
The data structures used by the Excel C API. Converting between these data structures and C/C++ data types. Getting data from and returning data to Excel.

Chapter 7 *Memory Management*
Stack limitations and how to avoid memory leaks and crashes. Communication between Excel and the DLL regarding responsibility for memory release.

Chapter 8 *Accessing Excel Functionality Using the C API*
The C interface equivalent of the XLM macro language and how to use it in a DLL. Information about some of the more useful functions and their parameters. Working with named ranges, menus, toolbars and C API dialogs. Trapping events within a DLL.

Chapter 9 *Miscellaneous Topics*
Timing function execution speed. A brief look at how to access Excel's objects and their methods and properties using IDispatch and COM. Keeping track of cells. Multi-tasking,

multi-threading and asynchronous calls into a DLL add-in. Setting up timed calls to commands. Add-in design. Performance optimisation.

Chapter 10 *Example Add-ins and Financial Applications*
Examples that show how the previous chapters can be applied to financial applications such as, for example, Monte Carlo simulation, a stochastic volatility model, and constant maturity swap (CMS) derivative pricing.

1.8 SCOPE AND LIMITATIONS

The early chapters are intended to give just enough Excel and VBA background for the later chapters. There are literally dozens of books about Excel and VBA ranging from those whose titles are intended to coerce even the most timid out of the shadows, to those with titles designed to make them a must-buy for MBA students, such as '*Essential Power Excel Tips For Captains Of Industry And Entrepreneurs*'. (At the time of writing, this was a fictitious book title.) There are, of course, many well-written and comprehensive reference books on Excel and VBA. There are also a number of good specialist books for people who need to know how best to use Excel for a specific discipline, such as statistical analysis, for example.

The book is primarily focused on writing add-in worksheet functions. The reasons for this are gone into in later sections, such as section 2.9 *Commands versus functions in Excel* on page 27. One reason is that commands often rely on the creation of user-defined dialogs, which is a task far better suited to VBA. Even if the functionality that your command needs is already written in C/C++ code in a DLL, it can still easily be accessed from VB. Another reason is that, in general, commands do not have the same speed of execution requirements as worksheet functions – one of the main reasons for using a C/C++ DLL for functions.

Commands are covered to a certain extent, nevertheless. They can be a useful part of a well planned interface to a DLL. Knowing how to create and access them without the use of VBA is therefore important. Knowing how to create menus and menu items is important if you want DLL commands to be accessed in a seamless way. Chapter 8 *Accessing Excel Functionality Using the C API* on page 223 covers these topics.

The Excel COM interface is largely beyond the scope of this book, mainly to keep the book focused on the writing of high performance worksheet functions which COM does not help with. The other main reason is that if you need functionality that COM provides and the C API does not, for example, access to certain Excel objects, you are probably better off using VBA. That said, there are examples given in Chapter 9 of the use of COM from an XLL or DLL.

This book is not intended to be industry-specific or profession-specific except in the final chapter where applications of particular interest in certain areas of finance are discussed. It should be noted that the book is not intended to be a finance text book and deliberately avoids laborious explanations of things that finance professionals will know perfectly well. Nor are examples intended to necessarily cover all of what is a very broad field. It is hoped that readers will see enough parallel with their own field to be able to apply earlier sections of the book to their own problems without too much consternation. There are two new key sections in this second edition that contain applications with a little analytical background as well as a discussion of how they can be implemented in Excel. These are the stochastic volatility model SABR, and constant maturity swap (CMS) derivative pricing.