

**Financial Applications Using  
Excel Add-in Development in C/C++**

**Second Edition of Excel  
Add-in Development in C/C++**

**Steve Dalton**



**John Wiley & Sons, Ltd**



Financial Applications using  
Excel Add-in Development in C/C++

For other titles in the Wiley Finance Series  
please see [www.wiley.com/finance](http://www.wiley.com/finance)

# Financial Applications Using Excel Add-in Development in C/C++

Second Edition of Excel  
Add-in Development in C/C++

**Steve Dalton**



John Wiley & Sons, Ltd

Copyright © 2007

John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester,  
West Sussex PO19 8SQ, England

Telephone (+44) 1243 779777

Email (for orders and customer service enquiries): [cs-books@wiley.co.uk](mailto:cs-books@wiley.co.uk)

Visit our Home Page on [www.wileyeurope.com](http://www.wileyeurope.com) or [www.wiley.com](http://www.wiley.com)

All Rights Reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except under the terms of the Copyright, Designs and Patents Act 1988 or under the terms of a licence issued by the Copyright Licensing Agency Ltd, 90 Tottenham Court Road, London W1T 4LP, UK, without the permission in writing of the Publisher. Requests to the Publisher should be addressed to the Permissions Department, John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England, or emailed to [permreq@wiley.co.uk](mailto:permreq@wiley.co.uk), or faxed to (+44) 1243 770620.

Designations used by companies to distinguish their products are often claimed as trademarks. All brand names and product names used in this book are trade names, service marks, trademarks or registered trademarks of their respective owners. The Publisher is not associated with any product or vendor mentioned in this book.

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold on the understanding that the Publisher is not engaged in rendering professional services. If professional advice or other expert assistance is required, the services of a competent professional should be sought.

### ***Other Wiley Editorial Offices***

John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA

Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA

Wiley-VCH Verlag GmbH, Boschstr. 12, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 42 McDougall Street, Milton, Queensland 4064, Australia

John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01, Jin Xing Distripark, Singapore 129809

John Wiley & Sons Canada Ltd, 6045 Freemont Blvd, Mississauga, ONT, L5R 4J3, Canada

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

### ***Library of Congress Cataloging-in-Publication Data***

Dalton, Steve.

Financial applications using Excel add-in development in C/C++ / Steve Dalton.—2nd ed.  
p. cm.

Earlier ed. published under title: Excel add-in development in C/C++ : applications in finance.  
Chichester : Wiley, c2004.

Includes bibliographical references and index.

ISBN 978-0-470-02797-4 (cloth/cd : alk. paper)

1. Microsoft Excel (Computer file) 2. Business—Computer programs. 3. C (Computer program language)  
4. C++ (Computer program language) 5. Computer software—Development. I. Dalton, Steve.  
Excel add-in development in C/C++. II. Title.

HF5548 .4.M523D35 2007

005 .54—dc22

2006036080

### ***British Library Cataloguing in Publication Data***

A catalogue record for this book is available from the British Library

ISBN 978-0-0470-02797-4 (HB)

Typeset in 10/12pt Times by Laserwords Private Limited, Chennai, India

Printed and bound in Great Britain by Antony Rowe Ltd, Chippenham, Wiltshire

This book is printed on acid-free paper responsibly manufactured from sustainable forestry in which at least two trees are planted for each one used for paper production.

# Contents

<b>Preface to Second Edition</b>	<b>xvii</b>
<b>Preface to First Edition</b>	<b>xix</b>
<b>Acknowledgements for the First Edition</b>	<b>xxi</b>
<b>Acknowledgements for the Second Edition</b>	<b>xxiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Typographical and code conventions used in this book	1
1.2 What tools and resources are required to write add-ins	2
1.2.1 VBA macros and add-ins	3
1.2.2 C/C++ DLL add-ins	4
1.2.3 C/C++ DLLs that can access the C API and XLL add-ins	4
1.2.4 C/C++/C# .NET add-ins	4
1.3 To which versions of Excel does this book apply?	5
1.4 The future of Excel: Excel 2007 (Version 12)	5
1.4.1 Summary of key workbook changes	5
1.4.2 Aspects of Excel 2007 not covered in this book	6
1.4.3 Excel 2007 file formats	6
1.4.4 Compatibility between Excel 2007 and earlier versions	6
1.5 About add-ins	7
1.6 Why is this book needed?	8
1.7 How this book is organised	9
1.8 Scope and limitations	10
<b>2 Excel Functionality</b>	<b>11</b>
2.1 Overview of Excel data organisation	11
2.2 A1 versus R1C1 cell references	12
2.3 Cell contents	13
2.4 Worksheet data types and limits	13
2.5 Excel input evaluation	15
2.6 Data type conversion	16
2.6.1 The unary = operator	16
2.6.2 The unary – operator (negation)	16

---

2.6.3	Number-arithmetic binary operators: + - */^	17
2.6.4	Percentage operator: %	17
2.6.5	String concatenation operator: &	17
2.6.6	Boolean binary operators: =, < , > , <= , >= , < >	17
2.6.7	Conversion of single-cell references	18
2.6.8	Conversion of multi-cell range references	18
2.6.9	Conversion of defined range names	19
2.6.10	Explicit type conversion functions: N(), T(), TEXT(), VALUE()	20
2.6.11	Worksheet function argument type conversion	20
2.6.12	Operator evaluation precedence	22
2.7	Strings	23
2.7.1	Length-prepend versus null-terminated strings	23
2.7.2	Byte strings versus Unicode strings	23
2.7.3	Unmanaged versus managed strings	24
2.7.4	Summary of string types used in Excel	25
2.7.5	Converting one string type to another	26
2.7.6	Hybrid length-counted null-terminated strings	27
2.8	Excel Terminology: Active and current	27
2.9	Commands versus functions in Excel	28
2.10	Types of worksheet function	29
2.10.1	Function purpose and return type	29
2.10.2	Array formulae – The Ctrl-Shift-Enter keystroke	30
2.10.3	Required, optional and missing arguments and variable argument lists	31
2.11	Complex functions and commands	31
2.11.1	Data Tables	31
2.11.2	Goal Seek and Solver Add-in	32
2.12	Excel recalculation logic	33
2.12.1	Marking dependents for recalculation	33
2.12.2	Triggering functions to be called by Excel – the trigger argument	34
2.12.3	Volatile functions	35
2.12.4	Cross-worksheet dependencies – Excel 97/2000 versus 2002 and later versions	36
2.12.5	User-defined functions (VB Macros) and add-in functions	38
2.12.6	Data Table recalculation	40
2.12.7	Conditional formatting	40
2.12.8	Argument evaluation: IF(), OR(), AND(), CHOOSE(). . .	41
2.12.9	Controlling Excel recalculation programmatically	42
2.12.10	Forcing Excel to recalculate a workbook or other object	44
2.12.11	Using functions in name definitions	45
2.12.12	Multi-threaded recalculation	45
2.13	The Add-in Manager	46
2.14	Loading and unloading add-ins	46
2.14.1	Add-in information	47
2.15	Paste function dialog	47



2.15.1	Function category	47
2.15.2	Function name, argument list and description	48
2.15.3	Argument construction dialog	48
2.16	Good spreadsheet design and practice	49
2.16.1	Filename, sheet title and name, version and revision history	49
2.16.2	Magic numbers	49
2.16.3	Data organisation and design guidelines	50
2.16.4	Formula repetition	51
2.16.5	Efficient lookups: MATCH(), INDEX() and OFFSET() versus VLOOKUP()	51
2.17	Problems with very large spreadsheets	54
2.18	Conclusion	54
<b>3</b>	<b>Using VBA</b>	<b>55</b>
3.1	Opening the VB editor	55
3.2	Using VBA to create new commands	56
3.2.1	Recording VBA macro commands	57
3.3	Assigning VBA command macros to control objects in a worksheet	58
3.4	Using VBA to trap Excel events	59
3.5	Using VBA to create new functions	61
3.5.1	Function scope	61
3.5.2	Declaring VBA functions as volatile	62
3.6	Using VBA as an interface to external DLL add-ins	62
3.6.1	Declaring DLL functions in VB	62
3.6.2	Call-by-reference versus call-by-value	63
3.6.3	Converting argument and return data types between VBA and C/C++	64
3.6.4	VBA data types and limits	64
3.6.5	VB/OLE Currency type	66
3.6.6	VB/OLE Bstr Strings	66
3.6.7	Passing strings to C/C++ functions from VBA	68
3.6.8	Returning strings to VBA from a DLL	70
3.6.9	Variant data type	71
3.6.10	Variant types supported by VBA	72
3.6.11	Variant types that Excel can pass to VBA functions	74
3.6.12	User-defined data types in VB	76
3.6.13	VB object data type	78
3.6.14	Calling XLM functions and commands from VBA: Application.ExecuteExcel4Macro()	79
3.6.15	Calling user-defined functions and commands from VBA: Application.Run()	79
3.7	Excel ranges, VB arrays, SafeArrays, array Variants	80
3.7.1	Declaring VB arrays and passing them back to Excel	81
3.7.2	Passing arrays and ranges from Excel to VBA to C/C++	83
3.7.3	Converting array Variants to and from C/C++ types	84
3.7.4	Passing VB arrays to and from C/C++	86

---

3.8	Commands versus functions in VBA	86
3.9	Creating VB add-ins (XLA files)	87
3.10	VBA versus C/C++: some basic questions	88
<b>4</b>	<b>Creating a 32-bit Windows (Win32) DLL Using Visual C++ 6.0 or Visual Studio .NET</b>	<b>89</b>
4.1	Windows library basics	89
4.2	DLL basics	89
4.3	DLL memory and multiple DLL instances	90
4.4	Multi-threading	90
4.5	Compiled function names	91
4.5.1	Name decoration	91
4.5.2	The extern "C" declaration	92
4.6	Function calling conventions: <code>__cdecl</code> , <code>__stdcall</code> , <code>__fastcall</code>	93
4.7	Exporting DLL function names	94
4.7.1	The <code>__declspec(dllexport)</code> keyword	95
4.7.2	Definition (*.DEF) files	95
4.7.3	Using a preprocessor linker directive	97
4.8	What you need to start developing add-ins in C/C++	97
4.9	Creating a DLL using Visual C++ 6.0	98
4.9.1	Creating the empty DLL project	98
4.9.2	Adding code to the project	100
4.9.3	Compiling and debugging the DLL	101
4.10	Creating a DLL using Visual C++ .NET 2003	103
4.10.1	Creating the empty DLL project	103
4.10.2	Adding code to the project	106
4.10.3	Compiling and debugging the DLL	106
4.11	Accessing DLL functions from VB	108
4.12	Accessing DLL functions from excel	110
<b>5</b>	<b>Turning DLLs into XLLs: The Add-in Manager Interface</b>	<b>111</b>
5.1	The <code>xlcall32</code> library and the C API functions	111
5.2	What does the Add-in manager do?	114
5.2.1	Loading and unloading installed add-ins	114
5.2.2	Active and inactive add-ins	114
5.2.3	Deleted add-ins and loading of inactivate add-ins	114
5.3	Creating an XLL: The <code>xlAuto</code> interface functions	115
5.4	When and in what order does Excel call the XLL interface functions?	116
5.5	XLL functions called by the Add-in Manager and Excel	117
5.5.1	<code>xlAutoOpen</code>	117
5.5.2	<code>xlAutoClose</code>	118
5.5.3	<code>xlAutoAdd</code>	118
5.5.4	<code>xlAutoRemove</code>	119
5.5.5	<code>xlAddInManagerInfo</code> ( <code>xlAddInManagerInfo12</code> )	120

5.5.6	xlAutoRegister (xlAutoRegister12)	122
5.5.7	xlAutoFree (xlAutoFree12)	123
<b>6</b>	<b>Passing Data Between Excel and the DLL</b>	<b>127</b>
6.1	Handling Excel's internal data structures: C or C++?	127
6.2	How Excel exchanges worksheet data with DLL add-in functions	128
6.2.1	Native C/C++ data types	128
6.2.2	Excel floating-point array structures: xl4_array, xl12_array	129
6.2.3	The xloper/xloper12 structures	135
6.2.4	The xlref/xlref12 structures	141
6.2.5	The xlmref/xlmref12 structures	142
6.2.6	The oper/oper12 structures	143
6.3	Defining constant xlopers/xloper12s	144
6.4	A C++ class wrapper for the xloper/xloper12 - cpp_xloper	146
6.5	Converting between xloper/xloper12s and C/C++ data types	154
6.6	Converting between xloper/xloper12 types	154
6.7	Converting between xlopers and variants	155
6.8	Converting between xlopers and xloper12s	159
6.9	Detailed Discussion of xloper types	163
6.9.1	Freeing xloper memory	164
6.9.2	Worksheet (floating point) number: xltypeNum	166
6.9.3	Length-counted string: xltypeStr	168
6.9.4	Excel Boolean: xltypeBool	174
6.9.5	Worksheet error value: xltypeErr	177
6.9.6	Excel internal integer: xltypeInt	178
6.9.7	Array (mixed type): xltypeMulti	180
6.9.8	Worksheet cell/range reference: xltypeRef and xltypeSRef	191
6.9.9	Empty worksheet cell: xltypeNil	196
6.9.10	Worksheet binary name: xltypeBigData	198
6.10	Initialising xloper/xloper12s	198
6.11	Missing arguments	201
<b>7</b>	<b>Memory Management</b>	<b>203</b>
7.1	Excel stack space limitations	203
7.2	Static add-in memory and multiple Excel instances	204
7.3	Getting Excel to free memory allocated by Excel	205
7.3.1	Freeing xloper memory within the DLL call	205
7.3.2	Freeing Excel-allocated xloper memory returned by the DLL function	206
7.3.3	Hiding xloper memory management within a C++ class	208
7.4	Getting Excel to call back the DLL to free DLL-allocated memory	208
7.5	Returning data by modifying arguments in place	211

7.6	Making add-in functions thread safe	212
7.6.1	Multi-threaded recalculations (MTR) in Excel 2007 (version 12)	212
7.6.2	Which of Excel's built-in functions are thread-safe	213
7.6.3	Allocating thread-local memory	214
7.6.4	Excel's sequencing of calls to <code>xlAutoFree</code> in a multi-threaded system	218
7.6.5	Using critical sections with memory shared between threads	219
<b>8</b>	<b>Accessing Excel Functionality using the C API</b>	<b>223</b>
8.1	The Excel 4 macro language (XLM)	223
8.1.1	Commands, worksheet functions and macro sheet functions	224
8.1.2	Commands that optionally display dialogs – the <code>xlPrompt</code> bit	225
8.1.3	Accessing XLM functions from the worksheet using defined names	225
8.2	The <code>Excel4()</code> , <code>Excel12()</code> C API functions	226
8.2.1	Introduction	226
8.2.2	<code>Excel4()</code> , <code>Excel12()</code> return values	227
8.2.3	Calling Excel worksheet functions in the DLL using <code>Excel4()</code> , <code>Excel12()</code>	229
8.2.4	Calling macro sheet functions from the DLL using <code>Excel4()</code> , <code>Excel12()</code>	231
8.2.5	Calling macro sheet commands from the DLL using <code>Excel4()</code> / <code>Excel12()</code>	233
8.3	The <code>Excel4v()</code> / <code>Excel12v()</code> C API functions	233
8.4	What C API functions can the DLL call and when?	236
8.5	Wrapping the C API	238
8.6	Registering and un-registering DLL (XLL) functions	244
8.6.1	The <code>xlRegister</code> function	245
8.6.2	Specifying which category the function should be listed under	248
8.6.3	Specifying argument and return types	249
8.6.4	Giving functions macro sheet function permissions	252
8.6.5	Specifying functions as volatile	253
8.6.6	Specifying functions as thread-safe (Excel 2007 only)	253
8.6.7	Returning values by modifying arguments in place	253
8.6.8	The Paste Function dialog (Function Wizard)	254
8.6.9	Function help parameter to <code>xlRegister</code>	256
8.6.10	Argument help parameters to <code>xlRegister</code>	256
8.6.11	Managing the data needed to register exported functions	256
8.6.12	Registering functions with dual interfaces for Excel 2007 and earlier versions	263
8.6.13	A class based approach to managing registration data	266
8.6.14	Getting and using the function's register ID	269
8.6.15	Un-registering a DLL function	270
8.7	Registering and un-registering DLL (XLL) commands	271

8.7.1	Accessing XLL commands	273
8.7.2	Breaking execution of an XLL command	274
8.8	Functions defined for the C API only	274
8.8.1	Freeing Excel-allocated memory within the DLL: <code>xlFree</code>	274
8.8.2	Getting the available stack space: <code>xlStack</code>	275
8.8.3	Converting one <code>xloper</code> / <code>xloper12</code> type to another: <code>xlCoerce</code>	276
8.8.4	Setting cell values from a command: <code>xlSet</code>	278
8.8.5	Getting the internal ID of a named sheet: <code>xlSheetId</code>	279
8.8.6	Getting a sheet name from its internal ID: <code>xlSheetNm</code>	281
8.8.7	Yielding processor time and checking for user breaks: <code>xlAbort</code>	282
8.8.8	Getting Excel's instance handle: <code>xlGetInst</code>	283
8.8.9	Getting the handle of the top-level Excel window: <code>xlGetHwnd</code>	283
8.8.10	Getting the path and file name of the DLL: <code>xlGetName</code>	284
8.9	Working with binary names	285
8.9.1	The <code>xltypeBigData</code> <code>xloper</code>	286
8.9.2	Basic operations with binary names	286
8.9.3	Creating, deleting and overwriting binary names	287
8.9.4	Retrieving binary name data	287
8.9.5	Example worksheet functions	288
8.10	Workspace information commands and functions	289
8.10.1	Setting the application title: <code>xlAppTitle</code>	290
8.10.2	Setting the document window title: <code>xlWindowTitle</code>	290
8.10.3	Getting a reference to the active cell: <code>xlActiveCell</code>	291
8.10.4	Getting a list of all open Excel documents: <code>xlDocuments</code>	291
8.10.5	Information about a cell or a range of cells: <code>xlGetCell</code>	291
8.10.6	Sheet or workbook information: <code>xlGetDocument</code>	293
8.10.7	Getting the formula of a cell: <code>xlGetFormula</code>	297
8.10.8	Getting a cell's comment: <code>xlGetNote</code>	297
8.10.9	Information about a window: <code>xlGetWindow</code>	298
8.10.10	Information about a workbook: <code>xlGetWorkbook</code>	301
8.10.11	Information about the workspace: <code>xlGetWorkspace</code>	303
8.10.12	Information about the selected range or object: <code>xlSelection</code>	309
8.10.13	Getting names of open Excel windows: <code>xlWindows</code>	310
8.10.14	Converting a range reference: <code>xlFormulaConvert</code>	311
8.10.15	Converting text to a reference: <code>xlTextref</code>	312
8.10.16	Converting a reference to text: <code>xlRefText</code>	312
8.10.17	Information about the calling cell or object: <code>xlCaller</code>	313
8.10.18	Information about the calling function type	315
8.11	Working with Excel names	316
8.11.1	Specifying worksheet names and name scope	316
8.11.2	Basic operations with Excel names	318
8.11.3	Defining a name on a worksheet: <code>xlDefineName</code>	318

---

8.11.4	Defining and deleting a name in the DLL: <code>xlFSetName</code>	319
8.11.5	Deleting a worksheet name: <code>xlCDeleteName</code>	321
8.11.6	Getting the definition of a named range: <code>xlFGetName</code>	322
8.11.7	Getting the defined name of a range of cells: <code>xlFGetDef</code>	324
8.11.8	Getting a list of named ranges: <code>xlFNames</code>	325
8.12	Working with Excel menus	326
8.12.1	Menu bars and ID numbers and menu and command specifiers	327
8.12.2	Short-cut (context) menu groups	328
8.12.3	Getting information about a menu bar: <code>xlFGetBar</code>	330
8.12.4	Creating a new menu bar or restoring a default bar: <code>xlFAddBar</code>	332
8.12.5	Adding a menu or sub-menu: <code>xlFAddMenu</code>	332
8.12.6	Adding a command to a menu: <code>xlFAddCommand</code>	335
8.12.7	Displaying a custom menu bar: <code>xlFShowBar</code>	338
8.12.8	Adding/removing a check mark on a menu command: <code>xlFCheckCommand</code>	338
8.12.9	Enabling/disabling a custom command or menu: <code>xlFEnableCommand</code>	339
8.12.10	Changing a menu command name: <code>xlFRenameCommand</code>	341
8.12.11	Deleting a command from a menu: <code>xlFDeleteCommand</code>	342
8.12.12	Deleting a custom menu: <code>xlFDeleteMenu</code>	343
8.12.13	Deleting a custom menu bar: <code>xlFDeleteBar</code>	343
8.13	Working with toolbars	344
8.13.1	Getting information about a toolbar: <code>xlFGetToolbar</code>	345
8.13.2	Getting information about a tool button on a toolbar: <code>xlFGetTool</code>	345
8.13.3	Creating a new toolbar: <code>xlFAddToolbar</code>	346
8.13.4	Adding buttons to a toolbar: <code>xlCAddTool</code>	347
8.13.5	Assigning/removing a command on a tool: <code>xlCAssignToTool</code>	347
8.13.6	Enabling/disabling a button on a toolbar: <code>xlFEnableTool</code>	348
8.13.7	Moving/copying a command between toolbars: <code>xlCMoveTool</code>	348
8.13.8	Showing a toolbar button as pressed: <code>xlFPressTool</code>	349
8.13.9	Displaying or hiding a toolbar: <code>xlCShowToolbar</code>	349
8.13.10	Resetting a built-in toolbar: <code>xlFResetToolbar</code>	350
8.13.11	Deleting a button from a toolbar: <code>xlCDeleteTool</code>	350
8.13.12	Deleting a custom toolbar: <code>xlFDeleteToolbar</code>	351
8.14	Working with custom dialog boxes	351
8.14.1	Displaying an alert dialog box: <code>xlCAlert</code>	351
8.14.2	Displaying a custom dialog box: <code>xlFDialogBox</code>	352
8.14.3	Restricting user input to dialog boxes: <code>xlCDisableInput</code>	356
8.15	Trapping events with the C API	356
8.15.1	Trapping a DDE data update event: <code>xlCOnData</code>	357
8.15.2	Trapping a double-click event: <code>xlCOnDoubleClick</code>	357

8.15.3	Trapping a worksheet data entry event: <code>xlOnEntry</code>	358
8.15.4	Trapping a keyboard event: <code>xlOnKey</code>	358
8.15.5	Trapping a recalculation event: <code>xlOnRecalc</code>	360
8.15.6	Trapping a window selection event: <code>xlOnWindow</code>	360
8.15.7	Trapping a system clock event: <code>xlOnTime</code>	361
8.16	Miscellaneous commands and functions	361
8.16.1	Disabling screen updating during command execution: <code>xlEcho</code>	361
8.16.2	Displaying text in the status bar: <code>xlMessage</code>	361
8.16.3	Evaluating a cell formula: <code>xlEvaluate</code>	362
8.16.4	Calling user-defined functions from an XLL or DLL: <code>xlUDF</code>	363
8.16.5	Calling user-defined commands from an XLL or DLL: <code>xlRun</code>	363
8.17	The <code>XLCallVer()</code> C API function	364
<b>9</b>	<b>Miscellaneous Topics</b>	<b>365</b>
9.1	Timing function execution in VBA and C/C++	365
9.2	Relative performance of VBA, C/C++: Tests and results	369
9.2.1	Conclusion of test results	372
9.3	Relative performance of C API versus VBA calling from a worksheet cell	372
9.4	Detecting when a worksheet function is called from an Excel dialog	373
9.4.1	Detecting when a worksheet function is called from the Paste Function dialog (Function Wizard)	374
9.4.2	Detecting when a worksheet function is called from the Search and Replace dialog	375
9.4.3	Detecting when a worksheet function is called from either the Search and Replace or Paste Function dialogs	375
9.5	Accessing Excel functionality using COM/OLE automation using C++	376
9.5.1	Initialising and un-initialising COM	377
9.5.2	Getting Excel to recalculate worksheets using COM	379
9.5.3	Calling user-defined commands using COM	380
9.5.4	Calling user-defined functions using COM	382
9.5.5	Calling XLM functions using COM	383
9.5.6	Calling worksheet functions using COM	383
9.6	Maintaining large data structures within the DLL	385
9.7	A C++ Excel name class example, <code>xlName</code>	387
9.8	Keeping track of the calling cell of a DLL function	389
9.8.1	Generating a unique name	390
9.8.2	Obtaining the internal name of the calling cell	393
9.8.3	Naming the calling cell	394
9.8.4	Internal XLL name housekeeping	396
9.9	Passing references to Excel worksheet functions	398
9.9.1	Data references	398
9.9.2	Function references	398