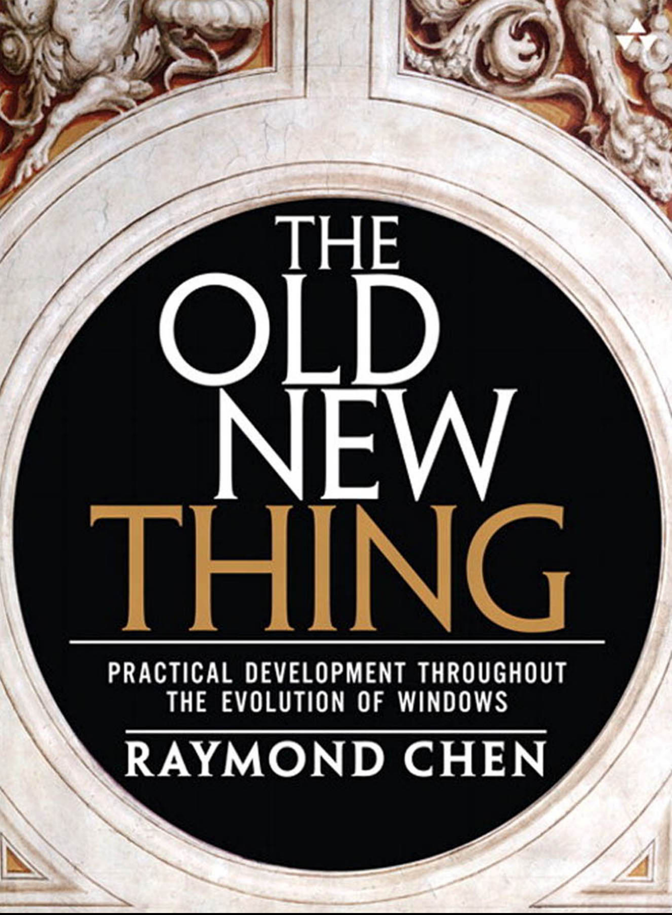


THE OLD NEW THING

PRACTICAL DEVELOPMENT THROUGHOUT THE EVOLUTION OF WINDOWS

RAYMOND CHEN

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Praise for *The Old New Thing*

"Raymond Chen is the original raconteur of Windows."

—Scott Hanselman, ComputerZen.com

"Raymond has been at Microsoft for many years and has seen many nuances of Windows that others could only ever hope to get a glimpse of. With this book, Raymond shares his knowledge, experience, and anecdotal stories, allowing all of us to get a better understanding of the operating system that affects millions of people every day. This book has something for everyone, is a casual read, and I highly recommend it!"

—Jeffrey Richter, Author/Consultant, Cofounder of Wintellect

"Very interesting read. Raymond tells the inside story of why Windows is the way it is."

—Eric Gunnerson, Program Manager, Microsoft Corporation

"Absolutely essential reading for understanding the history of Windows, its intricacies and quirks, and why they came about."

—Matt Pietrek, *MSDN Magazine*'s Under the Hood Columnist

"Raymond Chen has become something of a legend in the software industry, and in this book you'll discover why. From his high-level reminiscences on the design of the Windows Start button to his low-level discussions of GlobalAlloc that only your inner-geek could love, ***The Old New Thing*** is a captivating collection of anecdotes that will help you to truly appreciate the difficulty inherent in designing and writing quality software."

—Stephen Toub, Technical Editor, *MSDN Magazine*

This page intentionally left blank

THE OLD NEW THING

This page intentionally left blank

The Old New Thing

Practical Development Throughout the Evolution of Windows

Raymond Chen

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States please contact:

International Sales
international@pearsoned.com

Safari® BOOKS ONLINE ENABLED

This Book Is Safari Enabled
The Safari® Enabled icon on the cover of your favorite technology book means the book is available through Safari Bookshelf. When you buy this book, you get free access to the online edition for 45 days.

Safari Bookshelf is an electronic reference library that lets you easily search thousands of technical books, find code samples, download chapters, and access technical information whenever and wherever you need it.

To gain 45-day Safari Enabled access to this book:

- Go to http://www.awprofessional.com/safarienabled
- Complete the brief registration form
- Enter the coupon code

If you have difficulty registering on Safari Bookshelf or accessing the online edition, please e-mail customer-service@safaribooksonline.com.

Visit us on the Web: www.awprofessional.com

Library of Congress Cataloging-in-Publication Data

Chen, Raymond.
The old new thing. Practical development throughout the evolution of Windows / Raymond Chen.
p. cm.
Includes index.
ISBN 0-321-44030-7 (pbk. : alk. paper)
1. Microsoft Windows (Computer file) 2. Operating systems (Computers) 3. Computer software—Development. I. Title.

QA76.76.O63C45747 2007
005.4'46—dc22 2006028949

Copyright © 2007 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise.

For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax: (617) 671-3447

ISBN 0-321-44030-7

This product is printed digitally on demand.
First printing, December 2006

FOR MY FAMILY

This page intentionally left blank

Contents

Preface xxiii
Acknowledgments xxvii
About the Author xxix

CHAPTER ONE
Initial Forays into User Interface Design

Why do you have to click the Start button to shut down? 1
Why doesn't Windows have an "expert mode"? 2
The default answer to every dialog box is Cancel 3
The best setting is the one you don't even sense, but it's there, and it works the way you expect 6
In order to demonstrate our superior intellect, we will now ask you a question you cannot answer 7
Why doesn't Setup ask you if you want to keep newer versions of operating system files? 7
Thinking through a feature 9
When do you disable an option, and when do you remove it? 12
When do you put ... after a button or menu? 13
User interface design for vending machines 13

User interface design for interior door locks 15
The evolution of mascara in Windows UI 16

CHAPTER TWO

Selected Reminiscences on Windows 95

Why isn't my time zone highlighted on the world map? 19
Why didn't Windows 95 boot with more than 1GB of memory? 20
Why did Windows 95 have functions called BEAR, BUNNY, and PIGLET? 22
What about BOZOSLIVEHERE and TABTHETEXTOUTFORWIMPS? 23
What was in the Windows 95 Special Edition box? 25
Windows brings out the Rorschach test in everyone 25
The martial arts logon picture 26
Why a really large dictionary is not a good thing 27
An insight into the Windows 95 startup sound 27
It's a lot easier to write a column if you don't care about accuracy 28
Why does the System Properties page round the memory size? 29
Why does my hard drive light flash every few seconds? 29
The hunt for a faster syscall trap 30
One byte used to cost a dollar 31
Each product-support call costs a sale 32
Why isn't Tweak UI included on the Windows CD? 32
Turns out that you can't install Windows via xcopy 34
Buying an entire Egghead Software store 35
The history of the Windows PowerToys 35
How did Windows choose its final build numbers? 38
Why doesn't the build number increment for service packs? 39

CHAPTER THREE

The Secret Life of GetWindowText

How windows manage their text 41
Enter GetWindowText 42
What if I don't like these rules? 43
Can you give an example where this makes a difference? 44
Why are the rules for GetWindowText so weird? 44

CHAPTER FOUR

The Taskbar and Notification Area

Why do some people call the taskbar the "tray"? 47
Why does the taskbar default to the bottom of the screen? 49
Why doesn't the clock in the taskbar display seconds? 50
Why doesn't the taskbar show an analog clock? 51
When I dock my taskbar vertically, why does the word "Start" disappear? 51
Why don't notification icons get a message when the user clicks the "X" button? 52

CHAPTER FIVE

Puzzling Interface Issues

What are those little overlay icons? 53
Why are these unwanted files/folders opening when I log on? 54
What do the text label colors mean for files? 56
Why does my advanced options dialog say ON and OFF after every option? 57
What determines the order in which icons appear in the Alt+Tab list? 58
Why is the read-only property for folders so strange? 59

What's with those blank taskbar buttons that go away when I click on them? 59
What is the difference between Minimize All and Show Desktop? 60
What does boldface on a menu mean? 62
Where do those customized Web site icons come from? 62
Where did my task manager tabs and buttons go? 63
Will dragging a file result in a move or a copy? 64
Why does the Links folder keep re-creating itself? 65
Why are documents printed out of order when you multiselect and choose Print? 66
Raymond spends the day doing product support 67
Blow the dust out of the connector 68
How much is that gigabyte in the window? 69
Why can't I remove the "For test/evaluation purposes only" tag? 70

CHAPTER SIX

A History of the GlobalAlloc Function

The early years 71
Selectors 73
Transitioning to Win32 75
A peek at the implementation 76

CHAPTER SEVEN

Short Topics in Windows Programming

The scratch program 79
Getting a custom right-click menu for the caption icon 85
What's the difference between CreateMenu and CreatePopupMenu? 86
When does the window manager destroy menus automatically? 88
Painting only when your window is visible onscreen 89
Determining whether your window is covered 93

Using bitmap brushes for tiling effects 95
What is the DC brush good for? 98
Using ExtTextOut to draw solid rectangles 100
Using StretchBlt to draw solid rectangles 102
Displaying a string without those ugly boxes 103
Semaphores don't have owners 110
An auto-reset event is just a stupid semaphore 112

CHAPTER EIGHT
Window Management

Why do I get spurious WM_MOUSEMOVE messages? 115
Why is there no WM_MOUSEENTER message? 118
The white flash 118
What is the hollow brush for? 119
What's so special about the desktop window? 120
The correct order for disabling and enabling windows 121
A subtlety in restoring the previous window position 122
UI-modality versus code-modality 123
The WM_QUIT message and modality 126
The importance of setting the correct owner for modal UI 129
Interacting with a program that has gone modal 132
A timed MessageBox, the cheap version 133
The scratch window 135
The bonus window bytes at GWLP_USERDATA 136
A timed MessageBox, the better version 136
A timed context menu 138
Why does my window receive messages after it has been destroyed? 139

CHAPTER NINE

Reminiscences on Hardware

Hardware backward compatibility 141
The ghost CD-ROM drives 142
The Microsoft corporate network: 1.7 times worse than hell 143
When vendors insult themselves 144
Defrauding the WHQL driver certification process 145
A twenty-foot-long computer 146
The USB cart of death 147
New device detected: Boeing 747 147
There's an awful lot of overclocking out there 148

CHAPTER TEN

The Inner Workings of the Dialog Manager

On the dialog procedure 151
The evolution of dialog templates 163
Why dialog templates, anyway? 196
How dialogs are created 197
The modal dialog loop 204
Nested dialogs and DS_CONTROL 216
Why do we need a dialog loop, anyway? 224
Why do dialog editors start assigning control IDs with 100? 225
What happens inside DefDlgProc? 226
Never leave focus on a disabled control 228
What happens inside IsDialogMessage? 229
Why is the X button disabled on my message box? 237

CHAPTER ELEVEN

General Software Issues

Why daylight saving time is nonintuitive 239
Why do timestamps change when I copy files to a floppy? 241
Don't trust the return address 242
Writing a sort comparison function 243
You can read a contract from the other side 245
The battle between pragmatism and purity 249
Optimization is often counterintuitive 250
On a server, paging = death 253
Don't save anything you can recalculate 254
Performance gains at the cost of other components 255
Performances consequences of polling 257
The poor man's way of identifying memory leaks 258
A cache with a bad policy is another name for a memory leak 259

CHAPTER TWELVE

Digging into the Visual C++ Compiler

Do you know when your destructors run? 267
The layout of a COM object 272
Adjustor thunks 274
Pointers to member functions are very strange animals 276
What is __purecall? 280

CHAPTER THIRTEEN

Backward Compatibility

Sometimes an app just wants to crash 283
When programs grovel into undocumented structures 284
Why not just block the applications that rely on undocumented behavior? 286

Why 16-bit DOS and Windows are still with us 288
What's the deal with those reserved filenames such as NUL and CON? 290
Why is a drive letter permitted in front of UNC paths (sometimes)? 292
Do not underestimate the power of the game Deer Hunter 293
Sometimes the bug isn't apparent until late in the game 293
The long and sad story of the Shell Folders key 294
The importance of error code backward compatibility 297
Sure, we do that 298
When programs patch the operating system and mess up 299
The compatibility constraints of even your internal bookkeeping 300
Why does Windows keep your BIOS clock on local time? 301
Bad version number checks 302
The ways people mess up IUnknown::QueryInterface 303
When programs assume that the system will never change, Episode 1 305
When programs assume that the system will never change, Episode 2 306
The decoy Display Control Panel 308
The decoy visual style 309

CHAPTER FOURTEEN
Etymology and History

What do the letters W and L stand for in WPARAM and LPARAM? 311
Why was nine the maximum number of monitors in Windows 98? 312
Why is a registry file called a hive? 312
The management of memory for resources in 16-bit Windows 312

What is the difference between HINSTANCE and HMODULE? 313
What was the purpose of the hPrevInstance parameter to WinMain? 316
Why is the GlobalWire function called GlobalWire? 317
What was the difference between LocalAlloc and GlobalAlloc? 318
What was the point of the GMEM_SHARE flag? 320
Why do I sometimes see redundant casts before casting to LPARAM? 321
Why do the names of the registry functions randomly end in Ex? 322
What's the difference between SHGetMalloc, SHAlloc, CoGetMalloc, and CoTaskMemAlloc? 324
Why is Windows Error Reporting nicknamed Dr. Watson? 329
What happened to DirectX 4? 330
Why are HANDLE return values so inconsistent? 331
Why do text files end in Ctrl+Z? 333
Why is the line terminator CR+LF? 334
TEXT vs. _TEXT vs. _T, and UNICODE vs. _UNICODE 335
Why are dialog boxes initially created hidden? 335
When you change the insides, nobody notices 336
If FlushInstructionCache doesn't do anything, why do you have to call it? 337
If InitCommonControls doesn't do anything, why do you have to call it? 338
Why did InterlockedIncrement/Decrement only return the sign of the result? 339
Why does the function WSASetLastError exist? 340
Why are there broadcast-based mechanisms in Windows? 340
Where did windows minimize to before the taskbar was invented? 341
Why didn't the desktop window shrink to exclude the taskbar? 343

Why does the caret stop blinking when I tap the Alt key? 343
What is the deal with the ES_OEMCONVERT flag? 345
The story behind file system tunneling 346
Why do NTFS and Explorer disagree on filename sorting? 347
The Date/Time Control Panel is not a calendar 350
How did Windows 95 rebase DLLs? 351
What are SYSTEM_FONT and DEFAULT_GUI_FONT? 353
Why do up-down controls have the arrows backward? 354
A ticket to the Windows 95 launch 355

CHAPTER FIFTEEN
How Window Messages Are Delivered and Retrieved

Sent and posted messages 358
The life of a sent message 363
The life of a posted message 364
Generated posted messages 365
When does SendMessageCallback call you back? 368
What happens in SendMessageTimeout when a message times out? 369
Applying what you've learned to some message processing myths 370
How can you tell who sent or posted you a message? 371
You can't simulate keyboard input with PostMessage 371

CHAPTER SIXTEEN
International Programming

Case mapping on Unicode is hard 373
An anecdote about improper case mapping 374
Why you can't rotate text 375
What are these directories called 0409 and 1033? 379
Keep your eye on the code page 379
Why is the default 8-bit codepage called "ANSI"? 388

Why is the default console codepage called "OEM"? 388
Why is the OEM code page often called ANSI? 389
Logical but perhaps surprising consequences of converting between Unicode and ANSI 391

CHAPTER SEVENTEEN

Security

World-writable files 393
Hiding files from Explorer 394
Stealing passwords 395
Silent install of uncertified drivers 396
Your debugging code can be a security hole 397
Why shared sections are a security hole 398
Internet Explorer's Enhanced Security Configuration doesn't trust the intranet 402

CHAPTER EIGHTEEN

Windows 2000 and Windows XP

Why doesn't the new Start menu use Intellimenus in the All Programs list? 403
Why is there no programmatic access to the Start menu pin list? 404
Why does Windows XP Service Pack 2 sometimes forget my CD autoplay settings? 406
The unsafe device removal dialog 407
Two brief reminiscences on the Windows XP Comments? button 408
Why does Explorer eject the CD after you finish burning it? 408
Why does Windows setup lay down a new boot sector? 409
Psychic debugging: Why your expensive four-processor machine is ignoring three of its processors 410
Psychic debugging: Why your CPU usage is hovering at 50% 411

What's the deal with the DS_SHELLFONT flag? 412
Why does DS_SHELLFONT = DS_FIXEDSYS | DS_SETFONT? 413
What other effects does DS_SHELLFONT have on property sheet pages? 414

CHAPTER NINETEEN

Win32 Design Issues

Why does Win32 fail a module load if an import could not be resolved? 417
Why are structure sizes checked strictly? 418
Why do I have to return this goofy value for WM_DEVICECHANGE? 421
The arms race between programs and users 422
Why can't you trap TerminateProcess? 424
Why do some processes stay in Task Manager after they've been killed? 424
Understanding the consequences of WAIT_ABANDONED 425
Why can't I put hyperlinks in notification icon balloon tips? 427
Why can't I use the same tree item multiple times? 429
The kooky STRRET structure 429
Why can't you set UTF-8 as your ANSI code page? 431
When should you use a sunken client area? 432
Why is there no all-encompassing superset version of Windows? 433
Why is it even possible to disable the desktop, anyway? 433
What are the window and menu nesting limits? 435
What's the difference between HWND_TOP and HWND_TOPMOST? 435

CHAPTER TWENTY

Taxes

Hierarchical Storage Management 438

Geopolitics 439

Remote Desktop Connection and Painting 440

Fast User Switching and Terminal Services 443

Multiple users 444

Roaming user profiles 445

Redirected folders 447

My Documents vs. Application Data 450

Large address spaces 451

Power management and detecting battery power 455

Intermittent network connectivity 457

Anti-aliased fonts and ClearType 459

High DPI displays 462

Multiple monitors 467

The work area 470

Displaying your pop-up windows in the right place 471

Accessibility 472

CHAPTER TWENTY-ONE

Silliness

The much-misunderstood "nop" action 481

Don't let Marketing mess with your slides 482

Whimsical bug reports 482

Watch out for those sample URLs 483

No code is an island 484

But I have Visual Basic Professional 485

It's all about the translucent plastic 485

My first death threat 486

You can't escape those AOL CDs 487
Giving fair warning before plugging in your computer 487
Spider Solitaire unseats the reigning champion 488
There's something about Rat Poker 489
Be careful what you name your product group 490
The psychology of naming your internal distribution lists 490
Differences between managers and programmers 491
Using floppy disks as semaphore tokens 492
When a token changes its meaning midstream 492
Whimsical embarrassment as a gentle form of reprimand 493
Using a physical object as a reminder 494
The office disco party 495
The Halloween-themed lobby 495

Index 497

Preface

Much ink is devoted to describing the "how" of using and developing software for Windows, but few authors go into the "why." What might appear at first to be quirks often turn out to have entirely logical explanations, reflecting the history, evolution, and philosophy of the Microsoft Windows operating system. This book attempts to provide knowledge not so much in the form of telling what needs to be done (although there is certainly plenty of that, too) but rather by helping to understand why things came to be that way. Thus informed of the history and philosophy of Windows, you can become a more effective Windows programmer.

The emphasis here, then, is on the rationale behind Windows. It is not a reference or even a tutorial, but rather a "practical history," taking a conversational rather than didactic approach in an attempt to give you an appreciation for the philosophy of Windows through a series of brief, largely independent essays. You can therefore skip freely to topics of momentary interest (or technical expertise). Essays have been grouped into general themes, and there is the occasional sequential pedagogical treatment when a topic is explored in depth; even in those cases, however, the topic is confined to a single self-contained chapter.

Writer and commentator David Sedaris is often asked whether his stories are true. He responds that they are "true enough." Like David Sedaris's stories, the material in this book is also "true enough." The focus is on the big picture, not on the minutiae; on making a single point without getting distracted by nitpicking detail. Key details are highlighted, but unimportant ones are set aside, and potentially interesting digressions may be neglected if they do not serve the topic at hand.

The primary audience is technology-savvy readers with an interest in Windows history. About half of the essays require no programming background. Most of the remaining topics assume a basic background in software design and development, although nothing particularly advanced. Topics specifically related to Windows programming assume reader familiarity with Win32 user interface programming and COM. The table on page xxv provides a breakdown of the chapters for nonprogrammers and for general programmers who do not have an interest in Win32 specifically. Of course, you are welcome to skim chapters not explicitly marked as of interest to you. Perhaps you will find something interesting in them after all.

What will you get out of this book? As noted previously, the primary goal is to convey the philosophy and rationale behind what might at first appear to be an irrational design. You will also understand that when something can't be done in Windows, it's often for a good reason; and you will gain an appreciation of the lengths to which Windows goes to preserve backward compatibility (and why it's important that it do so). And if nothing else, you will be able to tell amusing stories about Windows history at cocktail parties (that is, cocktail parties thrown by other geeks).

Much of the short-essay material here has already appeared in one form or another on the author's Web site, The Old New Thing (http://blogs.msdn.com/oldnewthing/), but is substantially supplemented by new material better suited to book form.

Visit the Web page for this book (www.awprofessional.com/title/0321440307) to download two bonus chapters, "Tales of Application Compatibility" and "How to Ensure That Your Program Does Not Run Under Windows 95." Think of them if you like as the book version of a movie's unique and insightful deleted scenes. The Web page also contains the code samples from the book as well as errata.

Breakdown of Chapters by Audience

Chapter	Title	General Audience	General Programmer	Win32 Programmer
Chapter 1	Initial Forays into User Interface Design	x	x	x
Chapter 2	Selected Reminiscences on Windows 95	x	x	x
Chapter 3	The Secret Life of GetWindowText			x
Chapter 4	The Taskbar and Notification Area	x	x	x
Chapter 5	Puzzling Interface Issues	x	x	x
Chapter 6	A History of the GlobalLock Function			x
Chapter 7	Short Topics in Windows Programming			x
Chapter 8	Window Management			x
Chapter 9	Reminiscences on Hardware	x	x	x
Chapter 10	The Inner Workings of the Dialog Manager			x
Chapter 11	General Software Issues		x	x
Chapter 12	Digging into the Visual C++ Compiler		x	x
Chapter 13	Backward Compatibility	x	x	x
Chapter 14	Etymology and History	x	x	x
Chapter 15	How Window Messages Are Delivered and Retrieved			x
Chapter 16	International Programming	First half	x	x
Chapter 17	Security		x	x
Chapter 18	Reminiscences on Windows 2000 and Windows XP	First half	First half	x
Chapter 19	Win32 Design Issues		Part	x
Chapter 20	Taxes		x	x
Chapter 21	Silliness	x	x	x
*	Tales of Application Compatibility	x	x	x
*	How to Ensure That Your Program Doesn't Run Under Windows 95		x	x

* These bonus chapters can be downloaded from www.awprofessional.com/title/0321440307.

This page intentionally left blank

Acknowledgments

I want to begin by thanking Joan Murray at Addison-Wesley for believing in a book as unusual as this one. Without her support, this project would never have come together. Others at Addison-Wesley have also been of great help, including Tyrrell Albaugh, Patty Boyd, Keith Cline, Curt Johnson, and Chris Zahn. Ben Ryan deserves credit for suggesting to me back in the late 1990s that I should write a book on Win32 (sorry it took so long), and I blame Brad Abrams for flat-out telling me to start a Web log in 2003.

Additional thanks to Betsy Aoki, Jeff Davis, Henry Gabryjelski, Jeffery Galinovsky, Michael Grier, Mike Gunderloy, Eric Gunnerson, Chris Guzak, Johnson M. Hart, Francis Hogle, Aleš Holecek, Michael Kaplan, KC Lemson, Shelley McKinley, Rico Mariani, Joseph Newcomer, Adrian Oney, Larry Osterman, Matt Pietrek, Jeffrey Richter, Mike Schmidt, Jan Shanahan, Joel Spolsky, Stephen Toub, and Ed Wax for their assistance in various capacities throughout this entire project (either intentional or unwitting).

Finally, I must acknowledge all the people who visit my Web site, which serves as the title as well as the inspiration for this book. They're the ones who convinced me to give this book thing another try.

This page intentionally left blank

About the Author

Raymond Chen is a programmer in the Windows division at Microsoft. His Web site The Old New Thing deals with Windows history and Win32 programming. He also writes the Windows Confidential column for *TechNet Magazine*.

This page intentionally left blank

CHAPTER ONE

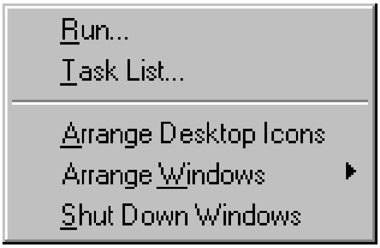
Initial Forays into User Interface Design

If you ask ten people for their thoughts on user interface design, you will get ten self-proclaimed expert opinions. Designing an interface for a single user grants you the luxury of just asking your customer what they want and doing it, but designing an interface for a large audience forces you to make tough decisions. Here are some stories on the subject of user interface design, starting with probably the most frequently asked question about the Windows 95 user interface.

Why do you have to click the Start button to shut down?

Back in the early days of what would eventually be named Windows 95, the taskbar didn't have a Start button. (Later, you'll learn that back in the early days of the project, the taskbar wasn't called the taskbar.)

Instead of the Start button, three buttons were displayed in the lower-left corner: the System button (icon: the Windows flag), the Find button (icon: an eyeball), and the Help button (icon: a question mark). Find and Help are self-explanatory. The System button gave you this menu:



Over time, the Find and Help buttons eventually joined the System button menu, and the System button menu itself gradually turned into the Windows 95 Start menu. Some menu options such as Arrange Windows (which led to options such as Cascade Windows and Tile Windows Horizontally) moved to other parts of the user interface; others such as Task List vanished completely.

One thing kept showing up during usability tests as a major hurdle: People turned on the computer and just sat there, unsure what to do next.

That's when someone got the idea of labeling the System menu Start. It says, "Psst. Click here." With this simple change, the usability results improved dramatically because, all of a sudden, people knew what to click when they wanted to do something.

So why is Shut down on the Start menu?

When we asked people to shut down their computers, they clicked the Start button. Because, after all, when you want to shut down, you have to start somewhere.

Why doesn't Windows have an "expert mode"?

We often get requests like this:

> *There should be a slider bar somewhere, say on the Performance tab, that ranges from Novice to Advanced. At the highest level, all the advanced settings are turned on. At the Novice level, all the settings for beginners are turned on. In between, we can gradually enable stuff.*

We've been trying to do something like this since even before Windows 95, and it doesn't work.

It doesn't work because those who might be whizzes at Excel will rate themselves as Advanced even though they can't tell a page file from a box of corn flakes. They're not stupid. They really are advanced users. Just not advanced at the skill we're asking them about.

And before you go mocking the beginners: Even so-called advanced users don't know everything. I know a lot about GUI programming, but I only know a little about disk partitioning, and I don't know squat about Active Directory. So am I an expert? When I need to format a hard drive, I don't want to face a dialog box filled with incomprehensible options. I just want to format the hard drive.

In the real world, people who are experts in one area are probably not experts in other areas. It's not something you can capture in a single number.

The default answer to every dialog box is Cancel

THE PROBLEM WITH displaying a dialog box is that people will take every opportunity to ignore it. One system administrator related a story in a *Network World* magazine online contest of a user who ignored a dozen virus security warnings and repeatedly tried to open an infected email attachment, complaining, "I keep trying to open it, but nothing happens." When the administrator asked why the user kept trying to open an attachment from a stranger, the answer was,"It might have been from a friend! They might have made up a new email address and didn't tell me!"[1] This story is a template for how users treat any unexpected dialog: They try to get rid of it.

We see this time and time again. If you are trying to accomplish task A, and in the process of doing it, an unexpected dialog box B appears, you aren't going to stop and read and consider B carefully. You're going to try to find the quickest path to getting rid of dialog B. For most people, this means minimizing it or clicking Cancel or just plain ignoring it.

[1] "Why Some People Shouldn't Be Allowed Near Computers," *Network World*, August 23, 2003, http://napps.networkworld.com/compendium/archive/003362.html.

This manifests itself in many ways, but the basic idea is, "That dialog box is scary. I'm afraid to answer the question because I might answer it incorrectly and lose all my data. So I'll try to find a way to get rid of it as quickly as possible."

Here are some specific examples, taken from conversations I have had with real customers who called the Microsoft customer support line:

- "How do I make this error message go away? It appears every time I start the computer."

  "What does this error message say?"

  "It says, 'Updates are ready to install.' I've just been clicking the X to make it go away, but it's really annoying."

- "Every time I start my computer, I get this message that says that updates are ready to install. What does it mean?"

  "It means that Microsoft has found a problem that may allow a computer virus to get into your machine, and it's asking for your permission to fix the problem. You should click on it so the problem can be fixed."

  "Oh, that's what it is? I thought it was a virus, so I just kept clicking 'No.'"

- "When I start the computer I get this big dialog that talks about automatic updates. I've just been hitting Cancel. How do I make it stop popping up?"

  "Did you read what the dialog said?"

  "No. I just want it to go away."

- "Sometimes I get the message saying that my program has crashed and would I like to send an error report to Microsoft. Should I do it?"

  "Yes, we study these error reports so we can see how we can fix the problem that caused the crash."

  "Oh, I've just been hitting Cancel because that's what I always do when I see an error message."

  "Did you read the error message?"

> "Why should I? It's just an error message. All it's going to say is 'Operation could not be performed because blah blah blah blah blah.' "

When most people buy a car, they don't expect to have to learn how an engine works and how to change spark plugs. They buy a car so that they can drive it to get from point A to point B. If the car makes a funny noise, they will ignore it as long as possible. Eventually, it may bother them to the point of taking it to a mechanic who will ask incredulously, "How long has it been doing this?" And the answer will be something like, "Oh, about a year."

The same goes for computers. People don't want to learn about gigabytes and dual-core processors and security zones. They just want to send email to their friends and surf the Web.

I myself have thrown out a recall notice because I thought it was junk mail. And computers are so filled with pop-up messages that any new pop-up message is treated as just another piece of junk mail to be thrown away.

Those who work at an information desk encounter this constantly. People ignore unexpected information. For example, even when a sign on a door says that "XYZ is closed today," you can bet that people will walk on in and ask, "Is XYZ open today?"

"No, it's closed today. Didn't you see the sign on the door?"

"Hmm, yeah, now that you mention it, there was a sign on the door, but I didn't read it."

Automobile manufacturers have learned to consolidate all their error messages into one message called "Check engine." Most people are conditioned to take the car in to a mechanic when the "Check engine" light goes on, and let the mechanic figure out what is wrong. Is it even possible to have a "Check engine" light for computers? Or would people just ignore that, too? How can a computer even tell whether a particular change in behavior is *normal* or *unintended?*

The best setting is the one you don't even sense, but it's there, and it works the way you expect

One solution that many people propose to the issue of "How should something be designed" is "Design it in every imaginable way, then let the end users pick the one they want with an option setting somewhere." This is a cop-out.

Computers need to be made simpler. This means fewer settings, not more. One way to reduce the number of settings is to make them implicit. You'll see more of this trend as researchers work on ways to make computers simpler, not more complicated.

Your toaster has a slider to set the darkness, which is remembered for your next piece of toast. There is no Settings dialog where you set the default darkness, but which you can override on a slice-by-slice basis.

Yes, this means that if you spent three weeks developing the perfect toaster slider position for Oroweat Honey Wheat Berry, and then you decide for a change of pace to have a slice of rye bread instead, you're going to have to move the slider and lose your old setting. People seem not to be particularly upset by this. The toaster works the way they expect.

Perhaps, you, the power-toaster-user, would want all toasters to let you save up to ten favorite darkness settings. But I suspect most people don't even sense that there are "missing options." If you started adding options to toasters, people would start wishing for the old days when toasters were simpler and easier to use.

"When I was a kid, you didn't have to log on to your toaster to establish your personal settings."

In order to demonstrate our superior intellect, we will now ask you a question you cannot answer

During the development of Windows 95, a placeholder dialog was added with the title "In order to demonstrate our superior intellect, we will now ask you a question you cannot answer." The dialog itself asked a technical question that you need a brain the size of a planet to answer. (Okay, your brain didn't need to be quite that big.)

Of course, there was no intention of shipping Windows 95 with such a dialog. The dialog was there only until other infrastructure became available, permitting the system to answer the question automatically.

But when I saw that dialog, I was enlightened. As programmers, we often find ourselves unsure what to do next, and we say, "Well, to play it safe, I'll just ask users what they want to do. I'm sure they'll make the right decision."

Except that they don't. As we saw earlier, the default answer to every dialog box is Cancel. If you ask the user a technical question, odds are that they're just going to stare at it blankly for a while, then try to cancel out of it. The lesson they've learned is this: Computers are hard to use.

So don't ask questions the user can't answer. It doesn't get you anywhere, and it just frustrates the user.

Why doesn't Setup ask you if you want to keep newer versions of operating system files?

Windows 95 Setup would notice that a file it was installing was older than the file already on the machine and would ask you whether you wanted to keep the existing (newer) file or overwrite it with the older version.

Asking the user this question at all turned out to have been a bad idea. It's one of those dialogs that asks users a question they have no idea how to answer.

Suppose you're installing Windows 95 and you get the file version conflict dialog box. "The file Windows is attempting to install is older than the one already on the system. Do you want to keep the newer file?" What do you do?

Well, if you're like most people, you say, "Um, I guess I'll keep the newer one," so you click Yes.

And then a few seconds later, you get the same prompt for some other file. And you click Yes again.

And then a few seconds later, you get the same prompt for yet another file. Now you're getting nervous. Why is the system asking you all these questions? Is it second-guessing your previous answers? Often when this happens, it's because you're doing something bad and the computer is giving you one more chance to change your mind before something horrible happens. Like in the movies when you have to type Yes five times before you can launch the nuclear weapons.

Maybe this is one of those times.

Now you start clicking No. Besides, it's always safer to say "No," isn't it?

After a few more dialogs (clicking No this time), Setup finally completes. The system reboots, and … it blue-screens.

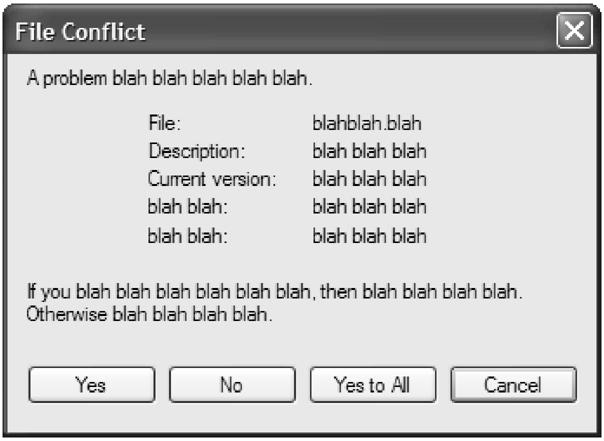
Why?

Because those five files were part of a matched set of files that together form your video driver. By saying "Yes" to some of them and "No" to others, you ended up with a mishmash of files that don't work together.

We learned our lesson. Setup doesn't ask this question any more. It always overwrites the files with the ones that come with the operating system. Sure, you may lose functionality, but at least you will be able to boot. Afterward, you can go to Windows Update and update that driver to the latest version.

Some have suggested that expanding the dialog with more explanatory text would solve the problem, but this misses the fact that people don't want to be bothered with these dialogs to begin with, as well as the fact that more information doesn't help anyway because the user doesn't have the background knowledge necessary to make an informed decision in the first place.

To a user, the dialog looks like this:



File Conflict

A problem blah blah blah blah blah.

File:	blahblah.blah
Description:	blah blah blah
Current version:	blah blah blah
blah blah:	blah blah blah
blah blah:	blah blah blah

If you blah blah blah blah blah blah, then blah blah blah blah.
Otherwise blah blah blah blah.

Making the dialog longer just increases the number of blahs. It's like trying to communicate with someone who doesn't speak your language by repeating yourself louder and more slowly. Users just want to surf the Web and send email to their grandchildren. Whatever you put in the dialog, they simply won't read it. Giving the dialog more buttons merely increases the paralysis factor.

Do you know the name of your printer driver? Or whether you should keep version 4.12.5.101 or downgrade it to 4.12.4.8? I sure don't.

Thinking through a feature

EVERYONE HAS A suggestion for a taskbar grouping feature. It's just a little bit of code; why not just do it?

Writing the code is the easy part.

Designing a feature is hard.

You have several audiences to consider. It's not just about the alpha geeks; you have to worry about the grandmothers, the office workers, the IT departments. They all have different needs. Sometimes a feature that pleases one group offends another.

So let's look at some of the issues surrounding the proposed feature of allowing users to selectively ungroup items in the taskbar.

One issue with selective grouping is deciding the scope of the feature. Suppose the user ungroups Internet Explorer, then closes all the Internet Explorer windows, and then opens two new Internet Explorer windows: Do the new ones group?

If so, you now have an invisible setting. How do you configure grouping for programs that aren't running? (How do you configure something that you can't see?)

Suppose you've figured that out. That's fine for the alpha geeks, but what about Grandma?

"The Internet is all disorganized."

"What do you mean?"

"My Internet windows are all disorganized."

"Can you explain a little more?"

"My taskbar used to be nice and organized, but now the Internet parts are disorganized and spread out all over the place. It used to be nice and neat. I don't know how it happened. I hate the Internet. It's always messing up my computer."

What is the user interface for selective ungrouping? Anything that is on a context menu will be executed accidentally by tens of thousands of people due to mouse twitching. Putting the regroup onto the context menu isn't necessarily good enough because those people don't even realize it was a context menu that did it. It was just a mouse twitch.

Mouse twitches cause all sorts of problems. Some people accidentally dock their taskbar vertically; others accidentally resize their taskbar to half the size of the screen. Do not underestimate the havoc that can be caused by mouse twitching.

Soon people will want to do arbitrary grouping. "I want to group this command prompt, that Notepad window, and this Calc window together."

What about selective ungrouping? "I have this group of ten windows, but I want to ungroup just two of them, leaving the other eight grouped together."

When you have selective/arbitrary grouping, how do you handle new windows? What group do they go into?

Remember: If you decide, "No, that's too much," thousands of people will be cursing you for not doing enough. Where do you draw the line? And also remember that each feature you add will cost you another feature somewhere else. Manpower isn't free.

But wait, the job has just begun. Next, you get to sit down and do the usability testing.

Soon you'll discover that everything you assumed to be true is completely wrong, and you have to go back to the drawing board. Eventually, you might conclude that you overdesigned the feature and you should go back to the simple on/off switch.

Wait, you're still not done. Now you have to bounce this feature off corporate IT managers. They will probably tear it to shreds, too. In particular, they're going to demand things such as remote administration and the capability to force the setting on or off across their entire company from a central location. (And woe unto you if you chose something more complicated than an on/off switch: Now you have to be able to deploy that complex setting across tens of thousands of computers, some of which may be connected to the corporate network via slow modems.)

Those are just some of the issues involved in designing a feature. Sometimes I think it's a miracle that features happen at all!

(Disclaimer: I'm not saying this is how the grouping feature actually came to be. I just used it as an illustration.)

Curiously, when I bring up this issue, the reaction of most people is not to consider the issue of trade-offs in feature design but rather to chip in with their vision of how the taskbar should work. "All I want is for the taskbar to do *X*. That other feature *Y* is useless." The value of *X* and *Y* changes from person to person; these people end up unwittingly proving my point rather than refuting it.

When do you disable an option, and when do you remove it?

When you're displaying a menu item or a dialog option, and the option is not available, you can either disable it or you can remove it. What is the rule for deciding which one to do?

Experiments have shown that if something is shown but disabled, users expect that they will be able to get it enabled if they tinker around enough.

Therefore, leave a menu item shown but disabled if there is something the user can do to cause the operation to become available. For example, in a media playback program, the option to stop playback is disabled if the media file is not playing. When it starts playing, however, the option becomes available again.

On the other hand, if the option is not available for a reason the user has no control over, remove it. Otherwise the user will go nuts looking for the magic way to enable it. For example, if a printer is not capable of printing color, don't show any of the color management options, because there's nothing the user can do with your program to make that printer a color printer.

By analogy, consider a text adventure game. The player tries something clever, such as "Take the torch from the wall," and the computer replies, "You can't do that, yet." This is the adventure game equivalent to graying out a menu item. The user is now going to go nuts trying to figure out what's happening: "Hmm, maybe I need a chair, or the torch is too hot, or I'm carrying too much stuff, or I have to find another character and ask him to do it for me."

If it turns out that the torch is simply not removable, what you've done is send the user down fruitless paths to accomplish something that simply can't be done. For an adventure game, this frustration is part of the fun. But for a computer program, frustration is not something people tend to enjoy.

Note that this isn't a hard-and-fast rule; it's just a guideline. Other considerations might override this principle. For example, you may believe that a consistent menu structure is more desirable because it is less confusing. (A media playback program, for example, might decide to leave the video-related options visible but grayed when playing a music file.)

When do you put … after a button or menu?

Save as… appears on some menus. You'll also find plenty of Customize… buttons. What is the rule for dots?

Many people believe that the rule for dots is this: "If it's going to display a dialog, you need dots." This is a misapprehension.

The rules are spelled out in the Windows User Interface Design Specifications and Guidelines (what a mouthful) in the section titled "Ellipses."

You should read the guidelines for the full story, but here's the short version: Use an ellipsis if the command requires additional information before it can be performed. Sometimes the dialog box is the command itself, such as About or Properties. Even though they display a dialog, the dialog *is the result,* as opposed to commands such as Print, where the dialog is *collecting additional information prior to the result.*

User interface design for vending machines

How hard can it be to design the user interface of a vending machine? You accept money, you have some buttons, users push the buttons, and they get their product and their change.

At least in the United States, many vending machines arrange their product in rows and columns. To select a product, you press the letter of the row and the number of the column. Could it be any simpler?

It turns out that subtleties lurk even in something this simple.

If the vending machine contains ten items per row, and you number them 1 through 10, a person who wants to buy product C10 has to push the buttons C and 10. But in our modern keyboard-based world, there is no 10 key. Instead, people press 1 followed by 0.

What happens if you type C + 1 + 0? After you type the 1, product C1 drops. Then the user realizes that there is no 0 key. And he bought the wrong product.

This is not a purely theoretical problem. I have seen this happen myself.

How would you fix this?

One solution is simply not to put so many items on a single row, considering that people have difficulty making decisions if given too many options. On the other hand, the vendor might not like that design; their goal might be to maximize the number of products.

Another solution is to change the labels so that the number of button presses needed always matches the number of characters in the label. In other words, no buttons with two characters on them (for example, a 10 button).

You could switch the rows and columns so that the products are labeled 1A through 1J across the top row and 9A through 9J across the bottom. This assumes you don't have more than nine rows, however. Some vending machines have many more selections on display, resulting in a very large number of rows.

If you have exactly ten items per row, you can call the tenth column 0. Notice, however that you also should remove rows I and O to avoid possible confusion with 1 and 0.

Some vending machines use numeric codes for all items rather than a letter and a digit. For example, if the cookies are product number 23, you punch 2 + 3. If you want the chewing gum (product code 71), you punch 7 + 1. What are some problems with having your products numbered from 1 to 99?

Here are a few problems. You may have come up with others:

- Products with codes 11, 22, 33, and so on may be selected accidentally. A faulty momentary switch might cause a single keypress to register as two, or a user may press the button twice by mistake or frustration.
- Product codes less than ten are ambiguous. Is a 3 a request for product number 3, or is the user just being slow at entering 32? Solving this by adding a leading zero will not work because people are in the habit of ignoring leading zeros.

- Product codes should not coincide with product prices. If there is a bag of cookies that costs 75 cents, users are likely to press 75 when they want the cookies, even though the product code for the cookies is 23.

User interface design for interior door locks

How hard can it be to design the user interface of an interior door lock?

Locking or unlocking the door from the inside is typically done with a latch that you turn. Often, the latch handle is in the shape of a bar that turns.

Now, there are two possible ways you can set up your lock. One is that a horizontal bar represents the locked position, and a vertical bar represents the unlocked position. The other is to have a horizontal bar represent the unlocked position and a vertical bar represent the locked position.

For some reason, it seems that most lock designers went for the latter interpretation. A horizontal bar means unlocked.

This is wrong.

Think about what the bar represents. When the deadbolt is locked, a horizontal bar extends from the door into the door jamb. Clearly, the horizontal bar position should reflect the horizontal position of the deadbolt. It also resonates with the old-fashioned way of locking a door by placing a wooden or metal bar horizontally across the face. (Does no one say "bar the door" any more?)

Car doors even followed this convention, back when car door locks were little knobs that popped up and down. The up position represented the removal of the imaginary deadbolt from the door/jamb interface. Pushing the button down was conceptually the same as sliding the deadbolt into the locked position.

But now, many car door locks don't use knobs. Instead, they use rocker switches. (Forward means lock. Or is it backward? What is the intuition there?) The visual indicator of the door lock is a red dot. But what does it mean? Red clearly means *danger,* so is it more dangerous to have a locked door or an unlocked door? I can never remember; I always have to tug on the door handle.

(Horizontally mounted power window switches have the same problem. Does pushing the switch forward raise the window or lower it?)

The evolution of mascara in Windows UI

THE LOOK OF the Windows user interface has gone through fashion cycles.

In the beginning, there was Windows 1.0, which looked very flat because screen resolutions were rather low in those days, and color depth was practically nonexistent. If you had 16 colors, you were doing pretty well. You couldn't afford to spend very many pixels on fluff such as borders, and shadows were out of the question because of lack of color depth.

The *flat look* continued in Windows 2.0, but Windows 3.0 added a hint of 3D, with a touch of beveling in push buttons.

Other people decided that the 3D look was the hot new thing, and libraries sprang up to add 3D shadow and outlining effects to nearly everything. The library CTL3D.DLL started out as just an Excel thing, but it grew in popularity until it became the standard way to make your dialog boxes *even more 3D.*

Come Windows 95, and even more of the system had a 3D look. For example, beveling appeared along the inside edge of the panes in the Explorer window. Furthermore, 3D-ness was turned on by default for all programs that marked themselves as designed for Windows 95. For programs that wanted to run on older versions of Windows as well, a new dialog style `DS_3DLOOK` was added, so that they could indicate that they wanted 3D-ization if available.

And if the 3D provided by Windows 95 by default wasn't enough, you could use CTL3D32.DLL to make your controls *even more 3D than ever before.* By this point, things started getting really ugly. Buttons on dialog boxes had so many heavy black outlines that it started to look like a really bad mascara job.

Fortunately, like many fashions that get out of hand, people realized that too much 3D is not a good thing. User interfaces got flatter. Instead of using 3D effects and bold outlines to separate items, subtler cues were used. Divider lines became more subdued and sometimes disappeared entirely.

Microsoft Office and Microsoft Money were two programs that embraced the *less-is-more* approach. The beveling is gone, and there are no 3D effects. Buttons are flat and unobtrusive. The task pane separates itself from the content pane by a simple gray line and a change in background shade. Even the toolbar has gone flat. Office 2000 also went largely flat, although some simple 3D effects linger (in the grooves and in the scrollbars, for example).

Windows XP jumped on the *flat-is-good* bandwagon and even got rid of the separator line between the tasks pane and the contents pane. The division is merely implied by the change in color. "Separation through juxtaposition" has become the new mantra.

Office XP and Outlook 2003 continue the trend and flatten nearly everything aside from the scrollbar elements. Blocks of color are used to separate elements onscreen, sometimes with the help of simple outlines.

So now the pendulum of fashion has swung away from 3D back toward flatness. Who knows how long this school of visual expression will hold the upper hand? Will 3D return with a vengeance when people tire of the starkness of the flat look?

This page intentionally left blank

CHAPTER THREE

The Secret Life of GetWindowText

The GetWindowText function is more complicated than you think. The documentation tries to explain its complexity with small words, which is great if you don't understand long words, but it also means that the full story becomes obscured.

Here's an attempt to give the full story.

How windows manage their text

There are two ways window classes can manage their text. They can do it manually or they can let the system do it. The default is to let the system do it.

If a window class lets the system manage its text, the system will do the following:

- Default handling of the `WM_NCCREATE` message takes the `lpWindowName` parameter passed to `CreateWindow/Ex` and saves the string in a "special place."
- Default handling of the `WM_GETTEXT` message retrieves the string from that special place.

- Default handling of the WM_SETTEXT message copies the string to that special place.

On the other hand, if a window class manages its window text manually, the system does not do any special handling, and it is the window class's responsibility to respond to the WM_GETTEXT/WM_SETTEXT messages and return/save the strings explicitly.

Frame windows typically let the system manage their window text. Custom controls typically manage their window text manually.

Enter GetWindowText

The GetWindowText function has a problem: Window text needs to be readily available without hanging. FindWindow needs to get window text to find a window. Task-switching applications need to get window text so that they can display the window title in the switcher window. It should not be possible for a hung application to clog up other applications. This is particularly true of the task-switcher scenario.

This argues *against* sending WM_GETTEXT messages, because the target window of the WM_GETTEXT might be hung. Instead, GetWindowText should use the "special place" because that cannot be affected by hung applications.

On the other hand, GetWindowText is used to retrieve text from controls on a dialog, and those controls frequently employ custom text management. This argues *for* sending WM_GETTEXT messages, because that is the only way to retrieve custom-managed text.

GetWindowText strikes a compromise:

- If you are trying to get the window text from a window in your own process, GetWindowText will send the WM_GETTEXT message.
- If you are trying to get the window from a window in another process, GetWindowText will use the string from the special place and not send a message.

According to the first rule, if you are trying to get text from a window in your own process, and the window is hung, `GetWindowText` will also hang. But because the window belongs to your process, it's your own fault, and you deserve to lose. Sending the `WM_GETTEXT` message ensures that text from windows that do custom text management (typically, custom controls) are properly retrieved.

According to the second rule, if you are trying to get text from a window in another process, `GetWindowText` will not send a message; it just retrieves the string from the special place. Because the most common reason for getting text from a window in another process is to get the title of the frame, and because frame windows typically do not do custom window text manipulation, this usually gets the right string.

The documentation simplifies this as "`GetWindowText` cannot retrieve text from a window from another application."

What if I don't like these rules?

IF THE SECOND rule bothers you because you need to get text from a custom control in another process, you can send the `WM_GETTEXT` message manually. Because you are not using `GetWindowText`, you are not subject to its rules.

Note, however, that if the target window is hung, your application will also hang because `SendMessage` will not return until the target window responds.

Note also that because `WM_GETTEXT` is in the system message range (0 to `WM_USER-1`), you do not need to take any special action to get your buffer transferred into the target process and to get the result transferred back to the calling process (a procedure known as *marshalling*). In fact, any special steps you take to this end are in error. The window manager does the marshalling for you.

Can you give an example where this makes a difference?

Consider this control:

```
SampleWndProc(...)
{
    case WM_GETTEXT:
        lstrcpyn((LPTSTR)lParam, TEXT("Booga!"), (int)wParam);
        return lstrlen((LPTSTR)lParam);
    case WM_GETTEXTLENGTH: return 7; // lstrlen("Booga!") + null
    ...
}
```

And application A, which does this:

```
hwnd = CreateWindow("Sample", "Frappy", ...);
```

Now consider process B, which gets the handle to the window created by application A (by whatever means):

```
TCHAR szBuf[80];
GetWindowText(hwnd, szBuf, 80);
```

This will return `szBuf = "Frappy"` because it is getting the window text from the special place. However

```
SendMessage(hwnd, WM_GETTEXT, 80, (LPARAM)szBuf);
```

will return `szBuf = "Booga!"`

Why are the rules for GetWindowText so weird?

Set the wayback machine to 1983. Your typical PC had an 8086 processor running at a whopping 4.7MHz, two 360K 5¼-inch floppy drives (or if you were really loaded, one floppy drive and a 10MB hard drive), and 256KB of memory

This was the world of Windows 1.0.

Windows 1.0 was a cooperatively multitasked system. No preemptive multitasking here. When your program got control, it had control for as long as it wanted it. Only when you called a function such as `PeekMessage` or `GetMessage` did you release control to other applications.

This was important because in the absence of a hardware memory manager, you really had to make sure that your memory didn't get ripped out from under you.

One important consequence of cooperative multitasking is that if your program is running, not only do you know that no other program is running, but you also know that *every window is responding to messages*. Why? Because if they are hung, they won't release control to you!

This means that it was *always* safe to send a message. You never had to worry about the possibility of sending a message to a hung window, because you knew that no windows were hung.

In this simpler world, `GetWindowText` was a straightforward function:

```
int WINAPI
GetWindowText(HWND hwnd, LPSTR pchBuf, int cch)
{
  // ah for the simpler days
  return SendMessage(hwnd, WM_GETTEXT, (WPARAM)cch, (LPARAM)pchBuf);
}
```

This worked for all windows, all the time. No special handling of windows in a different process.

It was the transition to Win32 and preemptive multitasking that forced the change in the rules, because for the first time, there was the possibility that (gasp) the window you were trying to communicate with was not responding to messages.

Now you have the backward-compatibility problem. As noted previously, many parts of the system and many programs rely on the capability to retrieve window text without hanging. So how do you make it possible to retrieve window text without hanging, while still enabling controls such as the edit control to do their own window text management?

The Win32 rules on `GetWindowText` are the result of this attempt to reconcile conflicting goals.

This page intentionally left blank

Index

Abstract functions, 281
Accelerators, 215–216
Accessibility, 57, 472–480
Active Accessibility feature, 480
AddRef method, 274
Address spaces, large, 451–455
Adjustor thunks, 274–275
Advanced Options dialog, 2–3, 57
All Programs list, 403–404
Alt key for blinking caret, 343–344
Alt+Esc hotkey, 58
Alt+Tab order, 58–59
Always on top, 58, 436
Analog clocks, 51
Animal-named functions, 22–23
Animations, stealing, 305
ANSI code page, 379–390
  converting with Unicode, 391–392
  UTF-8 as, 431–432
ANSI strings, 164
Anti-aliased fonts, 459–462
Anti-piracy holograms, 25–26
Anti-symmetry rule, 243
AOL CDs, 487
Application Compatibility Toolkit, 287–288
Application Data *vs.* My Documents, 450–451
Application Verifier, 288
Arithmetic library for Calc, 337
Arrows, up-down controls, 354–355
Auto-reset events, 112–114

Background operations in power management, 455–457
Backward compatibility, 283
  16-bit DOS and Windows, 288–290
  BIOS clock, 301–302
  bugs in, 293–294
  Deer Hunter, 293
  DirectX video driver, 298–299
  Display Control Panel, 308–309
  drive letters, 292–293
  error code, 297–298
  GetWindowText, 45–46
  hardware, 141–142
  intentional crashes, 283
  listview controls, 300–301
  operating system patches, 299–300
  Printers Control Panel, 306–307

Backward compatibility (Continued)
- QueryInterface, 303–305
- reserved filenames, 290–292
- Shell Folders key, 294–296
- undocumented behavior, 286–288
- undocumented functions, 23–24
- undocumented resources, 305
- undocumented structures, 284–286
- version numbers, 302–303
- visual style, 309

Balloon tips, 4, 427–428
Base addresses, DLL, 351–353
Based pointers, 401
Battery power detection, 455–457
BEAR function, 22–23
Behavior, undocumented, 286–288
Beta release numbers, 38–39
Betamax format, 249
BIOS clock, 301–302
BitBlt function, 442
Bitmap brushes, 95–98
Bitmaps on high DPI displays, 464–465
Black clock icon, 54
Blank taskbar buttons, 59–60
Blaster worm, 67
Blinking caret, 343–344
Blue-screen crashes, 396–397
Blue swirling arrows icon, 53
BN_CLICKED notification, 236
Boeing 747, modems on, 147
Boldface on menus, 62
Boot sectors, Windows setup for, 409–410
Booting in Windows 95, 20–22
Border wars, 20
BOZOSLIVEHERE function, 24
Brady Bunch, The, 312
Brickell, Edie, 28
BROADCAST_QUERY_DENY value, 421
Broadcasting messages, 340–341
Brushes
- DC, 98–100
- hollow, 119
- for tiling effects, 95–98

BS_* button styles, 232, 234
"Bug Bunny," 494
Bug reports, whimsical, 482–483
Bugs
- debugging as security hole, 397–398
- identifying, 293–294

Build numbers
- service pack, 39–40
- Windows 95, 38–39

BUNNY function, 22–23

CabView PowerToy, 36
Caches, memory leaks from, 259–266
Calc tool, unnoticed changes to, 336–337
Callback functions, 23–24
Cancel button
- as default, 3–5
- and X buttons, 237

Caption icon, 85–86
Caret, blinking, 343–344
Carriage return (CR) in line terminators, 334–335
Cars
- check engine lights, 5
- door locks, 15–16
- electric, 249

Case mapping, 373–375
Casts to LPARAM, 321–322
CComPtr template, 267–269
CD AutoPlay PowerToy, 36
CD-ROM ghost drives, 142–143
CDs
- AOL, 487
- autoplay settings, 406–407
- ejecting, 408–409

Certification process for drivers, 145–146, 397
Changes
- nonlocal effects of, 484–485
- timestamp, 241–242
- unnoticed, 336–337

Check engine lights, 5
CheckDlgRecursion macro, 156
Class brush, 118–119, 96–97

Classic templates
  16-bit, 164–172
  32-bit, 172–181
ClearType technique, 459–462
Client areas
  in frame windows, 199–200
  sunken, 432
Clocks
  BIOS, 301–302
  black, 54
  on taskbar, 50–51
  painting, 89–93
Clocks chips, overclocking, 148–150
CloseHandle function, 425
CoCreateGuid function, 298–299
Code bloat, 31
Code-modality *vs.* UI-modality, 123–126
Code pages, 379–388
  ANSI, 388
  OEM, 388–390
Code segments, 72
CoGetMalloc function, 324–329
Color displays, monochrome text on, 459–462
Color, icon label, 56–57
COLOR_WINDOW, 119
Columnists, 28, 34–35, 76
COM objects
  layout, 272–274
  teardown, 268
COMCTL32.DLL file, 338
Command Prompt Here PowerToy, 36
Comments? button, 408
Commutative diagrams, 187–188
comp.unix.wizards problem, 490–491
Compaction process, 72
Compatibility. *See* Backward compatibility
Compatibility subsystems, 289
Compilers
  code pages with, 383
  Visual C++. See Visual C++ compiler
CON filename, 290–292
Connectivity, network, 457–459
Connectors, blowing dust out of, 68–69
Content Watson, 329
Context menus
  boldface on, 62
  for caption icon, 85–86
  timed, 138–139
Contracts, interface, 245–248
Control Panel applications, 245–248
  and Tweak UI, 33
  and Windows 95 installation, 34
Controls
  for dialogs, 202–204
  focus on, 227–229
  IDs for, 225–227
Converting
  DLUs to pixels, 199, 202–203
  nonmodal dialog boxes to modal, 207–211
  between Unicode and ANSI, 391–392
Cooperative multitasking, 45
Coordinated universal time (UTC)
  and BIOS clock, 301–302
  and daylight saving time, 239–240
Copying
  timestamps changes in, 241–242
  when dragging files, 64–65
Costs, product-support, 32
CoTaskMemAlloc function, 324–329
CoUninitialize function, 268–269
Countries in geopolitics, 20, 439
Covered windows, determining, 93–95
CP/M operating system, 333–334
CPU meter, 63
CPU usage, 411–412
CR (carriage return) in line terminators, 334–335
Crashes
  intentional, 283
  from uncertified drivers, 396–397
CreateActCtx function, 248
CreateDialog function, 197, 222
CreateDialogIndirectParam function, 197–198, 206
CreateDialogParam function, 197–198, 213, 221

CreateEvent function, 112
CreateFile function, 331
CreateFileMapping function, 398
CreateMenu function, 86–88
CreateMutex function, 332
CreatePatternBrush function, 96–97
CreatePopupMenu function, 86–88
CreateProcess function, 249
CreateStdAccessibleObject function, 478
CreateThread function, 332
CreateWindowEx function, 166, 201, 203
Critical sections, 110–111
CSIDL_APPDATA directory, 451
CSIDL_LOCAL_APPDATA directory, 451
CSIDL_MYDOCUMENTS directory, 450
CSRSS.EXE program, 423
CTL3D.DLL library, 16
Ctrl+Z, 333–334
Custom right-click menus, 85–86
Customers, death threats from, 486–487
Customized Web site icons, 62–63

DAD (Desktop Applications Division), 490
Data tampering attacks, 393
Date/Time Control Panel, 350–351
Davis, Jeff, 493
Daylight saving time, 239–241
DBCS (double byte character sets), 431
DC brush, 98–100
DDE (Dynamic Data Exchange) messages, 371
ddeexec key, 66
Death threats, 486–487
Deaths from improper case mapping, 374–375
Debugging
  as security hole, 397–398
  psychic, 410–412
  vtables in, 401
Decoys, 306–309
Deer Hunter game, 293
DEFAULT_GUI_FONT font, 353–354
Defaults
  dialog box answers, 3–5, 151–159
  dialog box IDs, 232–233
  menu commands, 62
  selectors, 319
  shell font, 412–415
  Taskbar position, 49–50
DefDlgProc function, 152–159, 226–227
DefDlgProcEx macro, 156
Defect tracking system, 289
DeferWindowPos function, 435–436
Defragmenting undefragmentable structures, 284
Defrauding WHQL, 145–146
DefWindowProc function, 85
  for accessibility, 479
  for default colors, 99
Delay-load feature, 418
DeleteObject function, 96–97
Deleting
  Device Manager, 65–66
  options, 12
DeliverIncomingSentMessages pseudo-function, 360, 368, 370
DeliverMessage pseudo-function, 359–360
Denial-of-service attacks, 393
Descriptors, 73–75
DeskMenu PowerToy, 36
Desktop
  disabling, 433–434
  Remote Desktop Connection, 440–443
  taskbar on, 343
  window, 120–121
Desktop Applications Division (DAD), 490
Desktop composition, 95, 466–467
desktop.ini file
  opening at logon, 55–56
  for folders, 59
Desktop Window Manager (DWM), 95, 466–467
Destroyed windows, messages for, 139–140
Destroying
  menus, 88–89
  modal dialogs, 121–122
DestroyWindow function, 203
Destructors, 267–272

Dev O'Day hat, 494–495
Device Manager, deleting, 65–66
Dialog boxes, 151
  accelerators in, 215–216
  alternative designs, 154–163
  controls for, 202–204
    focus on, 227–229
    IDs for, 225–227
  converting nonmodal to modal, 207–211
  creating, 197–198
  default answers to, 3–5, 151–159
  default IDs for, 232–233
  DefDlgProc for, 226–227
  destroying, 121–122
  frame windows for, 198–202
  hidden, 335–336
  IsDialogMessage for, 229–236
  loops, 204
    basic, 204–207
    need for, 224–225
    structure, 207
    subtleties in, 211–214
  navigation in, 214–215
  nested, 216–224
  procedures, 151–163
  resizing, 222–224
  with unanswerable questions, 7, 8–9
  unexpected, 3–5, 489
Dialog templates, 163–164
  16-bit
    classic, 164–172
    extended, 181–187
  32-bit
    classic, 172–181
    extended, 187–194
  purpose, 195
  summary, 195, 196
Dialog units (DLUs), 165
  converting to pixels, 199
  for dialog templates, 414–415
  in high DPI displays, 462
DialogBox function, 129, 208
DialogBoxIndirectParam function, 205
DIALOGEX resource, 187, 194, 412, 415
Dictionaries, spell-checking, 27
Direct Annotation feature, 480
DirectX
  DirectX 4, 330–331
  video driver interface, 298–299
Disabled controls
  focus on, 228
  X buttons, 237
Disabling
  desktops, 433–434
  vs. removing, 12
  windows, 120–122
Discardable resources, 313
Discarding code, 72
Disco parties, 495
Disk quotas, 394
DispatchMessage function,
    130–131, 214, 224–225, 365
Display Control Panel, 308–309
Displaying
  pop-up windows, 471–472
  strings, 103–110
Displays
  high DPI, 462–467
  monochrome text on, 459–462
Distribution list names, 490–491
DLGC_* flags, 232, 233–236
DLLs. See Dynamic link libraries (DLLs)
DLUs (dialog units), 165
  converting to pixels, 199, 202–203
  for dialog templates, 414–415
  in high DPI displays, 462
DM_GETDEFID message, 229, 232–233
DM_SETDEFID message, 227–229, 232–233
Documents, printing order, 66–67
DoesDriverSupport function, 298
DoModal function, 157, 208–212
Dongle rumor, 28
Door locks, 15–16
DOS
  backward compatibility for, 288–290
  code page in, 389

Double byte character sets (DBCS), 431
Downward-pointing blue arrow icon, 53
DPI in high DPI displays, 462–467
Dr. Watson feature, 329–330
Dragging files, 64–65
DragQueryFile function, 76
Drawing
  with Remote Desktop Connection, 440–443
  solid rectangles
    with ExtTextOut, 100–102
    with StretchBlt, 102–103
DRAWITEMSTRUCT structure, 462
DrawSolidRect function, 100–102, 300
Drive letters in UNC paths, 292–293
Driver Verifier, 145, 147
Drivers
  Direct X, 298–299
  uncertified, 70, 396–397
  versions, 8
  WHQL certification process, 145–146
  In Visual Basic, 485
Drives in dragging files, 64–65
DropTarget key, 67
DS_3DLOOK style, 16
DS_ABSALIGN style, 200
DS_CONTROL style, 198, 216–224
DS_FIXEDSYS style, 199, 413–414
DS_NOFAILCREATE style, 203
DS_SETFONT style
  for 16-bit templates
    classic, 165, 167, 172
    extended, 181
  for 32-bit templates
    classic, 173, 176, 180
    extended, 191
  and DS_SHELLFONT, 413–414
  for frame windows, 199
DS_SHELLFONT style, 412–415
Dual-booting, BIOS clock in, 301–302
Dust in connectors, 68–69
DWLP_MSGRESULT, 152–153, 156
DWM (Desktop Window Manager), 466–467
Dynamic Data Exchange (DDE) messages, 371
Dynamic link libraries (DLLs)
  16-bit Windows resources, 314
  calling functions in, 338
  in large address spaces, 451–453
  rebasing, 254, 351–353
  with Visual C++ compiler, 268

Easter egg music, 28
EditWndProc function, 24
Egghead Software, 35
8-bit characters, 390
Ejecting CDs, 408–409
Electric cars, 249
Ellipses (...) on menus, 13
Embarrassment, whimsical, 493–494
"Enable Dubious Optimizations" switch, 146
EnableWindow function, 206, 211
Enabling windows, 121–122
Encoding integers as pointers, 452
EndDialog function, 129, 210–212
Eno, Brian, 27–28
EnumDisplayMonitors function, 468
Error code backward compatibility, 297–298
Error reporting, Dr. Watson,
    283–284, 329–330
ES_OEMCONVERT style, 345–346
ETO_OPAQUE flag, 92, 101
Event, auto-reset, 112–113
"Ex" suffix for registry function
    names, 322–324
ExitThread function, 271
Expert mode, requests for, 2–3
Explore From Here PowerToy, 36
Explorer
  CD ejection by, 408
  filename sorting in, 347–350
  hiding files from, 394–395
  parsing, 285
Extended templates
  16-bit, 181–187
  32-bit, 187–194

ExtTextOut function
  patched version, 299
  for solid rectangles, 100–102

Face-saving aspect in product support, 68–69
Fast task switching, 58
Fast User Switching, 443–444
FAT file system, times and dates in, 241–242
favicon.ico icon, 62–63
Features, thinking through, 9–11
FILE_ATTRIBUTE_HIDDEN flag, 394–395
FILE_ATTRIBUTE_OFFLINE flag, 54, 438–439
FILE_ATTRIBUTE_SYSTEM flag, 395
File system tunneling, 346–347
Filenames
  reserved, 290–292
  sorting, 347–350
Files
  Ctrl-Z in, 333–334
  dragging, 64–65
  hiding, 394–395
  Hierarchical Storage Management, 438–439
  text label colors for, 56–57
  timestamp changes in, 241–242
  unwanted, opened at logon, 54–56
  world-writable, 393–394
FILETIME structure, 239–240
FileTimeToLocalFileTime function, 239–240
Filling shapes, 98
FillRect function, 100
FindFirstFile function, 347, 391
FindResource function, 312
FindWindow function, 42
Fixed memory, 72
Flashing hard drive lights, 29–30
Flat look, 16–17
FlexiCD PowerToy, 36
Floating-point library for Calc, 337
Floppy disks as semaphore tokens, 492
Floppy drives
  file copies to, 241–242
  USB, 485–486
FlushInstructionCache function, 337
Focus in dialog boxes, 227–229
Folders
  opened at logon, 54–56
  read-only property for, 59
  redirected, 447–450
Fonts
  and ClearType, 459–462
  default shell, 412–415
  for dialogs, 202
  on high DPI displays, 462
  linking, 104–110
  support for, 103–104
"For test/evaluation purposes only," 70
Frame windows, 198–202
FreeLibraryAndExitThread function, 271
FreeResource function, 313
Function pointer tables, 273
Functions
  memory management, 324–329
  registry, 322–324
  return addresses of, 242–243
  strangely named, 22–24
  virtual, 280–282

Games
  Deer Hunter, 293
  Rat Poker, 489–490
  Spider Solitaire, 488
Garbage collector, 282
Generated posted messages, 365–367
Geopolitics, 20, 439
GetAsyncKeyState function, 372
GetClientRect function, 95
GetClipboardData function, 78
GetClipBox function, 93–94
GetClipRgn function, 95
GetCurrentProcess function, 333
GetDC function, 93

GetDesktopWindow function, 120, 132
GetDisplayNameOf method, 429
GetDlgCtrlID function, 189
GetFileAttributes function, 54
GetInstanceData function, 316
GetKeyState function, 372
GetMenuDefaultItem function, 62
GetMessage function, 359, 361, 363–364
  for cooperative multitasking, 45, 340
  information on, 357
  in message loops, 211–214
  for mouse move messages, 116
  WM_QUIT with, 127
GetModuleHandleEx function, 242, 271
GetMonitorInfo function, 469–470
GetNextDlgTabItem function, 221
GetProcAddress function, 418
GetQueueState function, 372
GetRegionData function, 95
GetStockObject function, 98, 100, 353–354
GetStrCodePages method, 104–106
GetSubMenu function, 87
GetSysColorBrush function, 98
GetSystemMetrics function, 468
GetVersion function, 302–303
GetWindowContextHelpId function, 181
GetWindowLong function, 24, 136, 152
GetWindowLongPtr function, 24, 136, 152
GetWindowPlacement function, 50, 122, 471
GetWindowText function, 41–42
  operation, 42–43
  rules in, 43–46
Ghost CD-ROM drives, 142–143
Gigabytes, 69–70
Glasser, Danny, 356
GlobalAlloc function, 71
  16-bit, 71–75
  32-bit 75–78
  vs. LocalAlloc, 318–320
GlobalFlags function, 76
GlobalLock function, 72–73, 77
GlobalReAlloc function, 73, 75–76
GlobalWire function, 317
GMEM_FIXED flag, 77
GMEM_MOVEABLE flag, 73, 77–78
GMEM_SHARE flag, 76, 320–321
Grier, Michael, 431
Group Policy, 33
Grouping on taskbar, 9–11
GUID generation, 298–299
GWLP_USERDATA constant, 136

Halloween-themed lobby, 495–496
Hand, palm up icon, 53
Handle table in hardware, 74
HANDLE_WM_CONTEXTMENU macro, 86
Handles
  memory, 77
  return values, 331–333
Hangs from error code compatibility, 297
Hard drive flashing lights, 29–30
Hardware, 141
  backward compatibility, 141–142
  ghost CD-ROM drives, 142–143
  overclocking, 148–150
  Plug and Play tests, 146
  USB Cart of Death, 147
  vendor misspellings, 144–145
  WHQL tests, 143–146
HDROP handle, 76
Height of dialog controls, 202
Hell test, 143–146
HGLOBAL handle, 72, 77–78, 319
Hidden attribute, 55–56
Hidden dialog boxes, 335–336
Hidden variables, 251
Hiding files, 394–395
Hierarchical Storage Management, 438–439
High DPI displays, 462–467
HINSTANCE handle *vs.* HMODULE, 313–316
Historical topics, 311
  blinking caret, 343–344
  broadcast-based mechanisms, 340–341
  changes unnoticed, 336–337

Ctrl+Z, 333–334
Date/Time Control Panel, 350–351
dialog boxes, 335–336
DirectX 4, 330–331
DLL rebase, 254, 351–353
Dr. Watson, 329–330
ES_OEMCONVERT flag, 345–346
file system tunneling, 346–347
filename sorting, 347–350
FlushInstructionCache, 337
GlobalWire, 317
GMEM_SHARE flag, 320–321
HANDLE return values, 331–333
HINSTANCE *vs.* HMODULE, 313–316
InitCommonControls, 338
InterlockedIncrement and InterlockedDecrement, 339–340
line terminators, 334–335
LocalAlloc *vs.* GlobalAlloc, 318–320
LPARAM redundant casts, 321–322
memory management functions, 324–329
monitors, 312
registry file names, 312
registry functions, 322–324
resource management, 312–313
spinners, 354–355
SYSTEM_FONT and DEFAULT_GUI_FONT, 353–354
taskbar on desktop, 343
text macros, 335
Windows 95 launch, 355–356
windows minimized, 341–343
WinMain hPrevInstance parameter, 316–317
WPARAM and LPARAM, 311–312
WSASetLastError, 340
"Hives," derivation of, 312
HKEY_CURRENT_USER hive, 444
HKEY_LOCAL_MACHINE hive, 444, 446
HLT instruction, 141–142
HMODULE handle *vs.* HINSTANCE, 313–316
HMONITOR handle, 468
Holistic view of performance, 255–256
Hollow brush, 119
Holograms, anti-piracy, 25–26
Howard, Michael, 67
hPrevInstance parameter, 316–317
HRESULTs, 453–454
HWND_TOP handle *vs.* HWND_TOPMOST, 435–436
Hyperlinks in notification icon balloon tips, 427–428

IAccessible interface, 477–480
Icons
Alt+Tab order for, 58–59
caption, 85–86
notification, 427–428
overlay, 53–54, 256
Web site, 62–63
IDCANCEL button, 236–237
Idea kernel, 104
IDNO button, 237
IDs
dialog box controls, 225–227
dialog boxes, 232–233
IDYES button, 237
IMLangFontLink2 interface, 104–105
Imports, optional, 418
InitCommonControls function, 83, 338
InitCommonControlsEx function, 203, 338
Inkblot test, 25–26
Input messages for dialogs, 207
Input simulation, 371–372
Installation
boot sectors for, 409–410
Windows 95, 34–35
Intellimenus, 403–404
Intellivision, 249
Intentional crashes, 283
Inter-thread sent messages, 363
Interface contracts, 245–248
InterlockedDecrement function, 339–340
InterlockedIncrement function, 339–340
Intermittent network connectivity, 457–459

Internal distribution list names, 490–491
International programming, 373
  0409 and 1033 directories, 379
  case mapping, 373–375
  code pages, 379–388
    ANSI, 388
    OEM, 388–390
  converting between Unicode and ANSI, 391–392
  grammar, 379
  rotating text, 375–379
Internet Explorer security, 402
Intranet security, 402
INVALID_HANDLE_VALUE, 331–333
Invalid instruction exceptions, 30–31
Invalidation in painting, 91
IsDialogMessage function, 207
  in message loops, 125, 224, 365
  for navigation, 214–215, 226, 229–233
  operation, 235–236
  WM_GETDLGCODE with, 233–235
Itanium computers, power surges from, 487–488
ITaskbarList interface, 60
IUnknown interface, 273–274, 303–305

Juxtaposition, separation through, 17

Kaplan, Michael, 373
Kernel Toys, 36–37
Keyboard input simulation, 371–372
Killed processes in Task Manager, 424–425
Killing programs, 422–423
Kilo prefix, 69
Klondike Solitaire, 488

L in LPARAM, 311–312
Label colors for files, 56–57
Languages in keyboard input simulation, 372
Large address spaces, 451–455
LARGEADDRESSAWARE flag, 451
LastWriteTime function, 240
Leno, Jay, 19, 355
Less-is-more interface approach, 17
LF (line feed) in line terminators, 334–335
Line-of-business (LOB) applications, 289–290
Line terminators, 334–335
Linking fonts, 104–110
Links folder, re-created, 65–66
Listview controls, 300–301
Load failures, module, 417–418
LoadMenu function, 86
LoadResource function, 313
LOB (line-of-business) applications, 289–290
LocalAlloc function, 76, 318–320
LocalFileTimeToFileTime function, 240
LocalInit function, 318
Localization, dialog templates for, 195, 196
Locked memory, 72–73
Locks, car door, 15–16
Logon, picture, 26–27
Logon, unwanted files/folders opened at, 54–56
Loops, dialog, 204
  basic, 204–207
  need for, 224–225
  structure, 207
  subtleties in, 211–214
LPARAM
  redundant casts to, 321–322
  vs. WPARAM, 311–312
LresultFromObject function, 479
Luna visual style, 309

M3 Beta, 294–295
MAKEINTRESOURCE macro, 453–454
Mallard visual style, 309
Managers *vs.* programmers, 491–492
MapFont method, 104–105
Mapping
  case, 373–375
  Unicode and ANSI, 391
Maps, geopolitical issues in, 20, 439
Mariani, Rico, 266

Marketing, problems caused by, 482
Marshalling, 43
Martial arts logon picture, 26–27
Mascara, 16–17
Matched set of files in setup, 8
Maximum number of monitors, 312
MBCS (multi-byte character set), 431
Mega prefix, 69
Member functions, pointers to, 276–280
Memory and memory management
  16-bit Windows resources, 312–313
  for COM objects, 273–274
  dialog templates for, 195
  exception handling in, 74–75
  fixed memory, 72
  functions for, 324–329
  GlobalAlloc for. *See* GlobalAlloc function
  leaks
    from caches, 259–266
    identifying, 258–259
    on servers, 253–254
  reports, 29, 69–70
  shared, 398–402, 426
  in Windows 95, 20–22
Menus
  boldface on, 62
  for capture icon, 85–86
  creating, 86–88
  destroying, 88–89
  disabling *vs.* removing, 12
  ellipses on, 13
  nesting limits, 435
  Start
    Intellimenus on, 403–404
    pin list, 404–406
  timed, 138–139
Message boxes
  timed, 133–134, 136–138
  X buttons disabled in, 237
MessageBeep function, 91, 130
MessageBox function, 132, 224, 382–384
Messages
  dialog loop
    basic, 204–207
    need for, 224–225
    structure, 207
    subtleties in, 211–214
  window. *See* Window messages
MFU (most frequently used) programs list, 403–404
Milk carton bug report, 482–483
Minimize All feature, 60–61
Minimized windows, 341–343
Modal dialog boxes
  converting nonmodal to, 207–211
  destroying, 121–122
  loops, 204
    basic, 204–207
    need for, 224–225
    structure, 207
    subtleties in, 211–214
Modality
  dialog loops for, 224
  disabling owner, 120–122, 206, 211
  interacting with, 132
  UI *vs.* code, 123–126
  UI owners, 129–132
  and WM_QUIT messages, 126–129, 206, 211, 213
Modems on Boeing 747, 147
Module load failures, 417–418
MonitorFromWindow function, 468
MONITORINFO structure, 470
Monitors
  maximum number of, 312
  multiple, 308–309, 467–470
Monochrome text on color displays, 459–462
Most frequently used (MFU) programs list, 403–404
Mouse twitches, taskbar changes from, 10
Movable memory, 72–73
MoveToEx function, 105
Moving in dragging files, 64–65
MS-DOS
  backward compatibility for, 288–290
  code page in, 389

MsgWaitForMultipleObjects function, 359, 372
Multi-byte character set (MBCS), 431
Multi-select documents printing order, 66–67
MultiLanguage object, 106–107
Multilingual User Interface, 305
Multiple monitors, 308–309, 312, 467–470
Multiple users, 444–445
Multitasking
- broadcast-based mechanisms with, 341
- in Windows 1.0, 45

Mutexes, 110–111
My Documents *vs.* Application Data, 450–451
Myths, message processing, 370–371

Naked baby hologram, 25–26
Names
- internal distribution lists, 490–491
- function, strangely named, 22–24
- product groups, 490
- registry functions, 322–324

Navigation
- in dialog boxes, 214–215
- Tab key for, 229–232

Near pointers, 318
Negative coordinates for monitors, 467–468
Nesting
- dialog boxes, 216–224
- menus, 435

NetBEUI protocol, 144
Network card packet stress test, 143–144
Network connectivity, 457–459
Newline character, 335
NIF_PARSELINKS flag, 427–428
NMHDR structure, 371
"No action required" action items, 481
Nonlocal effects of changes, 484–485
Nonmodal dialog boxes, converting to modal, 207–211
Notepad
- copies of, 315
- unnoticed changes to, 336–337

Notification icons
- hyperlinks in, 427–428
- for X button clicks, 52

NotifyAddrChange function, 458–459
novtable optimization, 281–282
NTFS system
- alternate data stream, 394
- filename sorting in, 347–350
- times and dates in, 241–242

NUL filename, 290–292
NULL pointers, 304–305
Null-terminated ANSI strings, 164

ODA_FOCUS flag, 462
OEM code page, 379–388
Office disco parties, 495
Office redecoration, 495–496
Offline files, 53–54, 438–439, 450
OLE Chicken, 326–327
Operating systems
- program patches to, 299–300
- setup file versions, 7–9
- size, 31

Optimization, 250–253
Optional imports, 418
Options, disabling *vs.* removing, 12
Order
- Alt+Tab, 58–59
- dialog box tabs, 221
- disabling and enabling windows, 121–122
- printing, 66–67

Orr, Brian, 28
Overclocking, 148–150
Overlay icons, 53–54, 256
Owner-draw and overprinting, 462
Owners
- modal UI, 129–132
- semaphore, 110–112

Page boundaries, 352
Paging
- performance with, 254
- on servers, 253–254

Painting
  with Remote Desktop Connection, 440–443
  visible windows only, 89–93
PAINTSTRUCT structure, 81–82
Parsing Explorer view data structures, 285
Passwords, stealing, 395–396
Patches to operating systems, 299–300
PathMakeSystemFolder function, 59
Paths, drive letters in front of, 292–293
PBT_APMBATTERYLOW notification, 457
PBT_APMPOWERSTATUSCHANGE notification, 457
PeekMessage function, 357
  for cooperative multitasking, 45, 340
  vs. GetMessage, 213–214
  for sent and posted messages, 359–361, 364, 366–367
  for timed message boxes, 134
  WM_QUIT with, 127
Pens, DC, 100
Performance
  holistic view of, 255–256
  paging, 254
  polling, 257
PIGLET function, 22–23
Pin list, 404–406
Plug and Play tests, 146
Plympton, Bill, 28
PM_NOREMOVE flag, 367
Pointers
  to member functions, 276–280
  near, 318
  NULL, 304–305
Political issues, 20, 439
Polling performance, 257
Pop-up windows display, 471–472
Position
  dialog controls, 202
  pop-up windows, 471–472
  Taskbar, 49–50
  window, restoring, 122–123
Posted messages, 358–362
  generated, 365–367
  life of, 364–365
PostMessage function, 358, 370–372
PostQuitMessage function, 127–129
PostThreadMessage function, 358
Power management, 437, 455–457
Power outage, 487–488
PowerPoint presentations, 491–492
PowerToys, 35–37
  calculator, 337
Pragmatism *vs.* purity, 249–250
Preemptive multitasking, 341
Prefixes, memory, 69
PRINTDLG structure, 78
Printers Control Panel compatibility, 306–307
Printing order, 66–67
printit function, 399
PRN filename, 291
Processors
  affinity, 410–411
  and CPU usage, 411–412
  hidden variables in, 251
Product group names, 490
Product support
  costs, 32
  for developers, 67–69
Profiles, roaming user
  handling, 445–447
  redirected folders for, 447–450
Programmatic access to Start menu pin list, 404–406
Programmers *vs.* managers, 491–492
Programming, 79
  bitmap brushes, 95–98
  CreateMenu *vs.* CreatePopupMenu, 86–88
  DC brushes in, 98–100
  displaying strings, 103–110
  international. See International programming
  menu destruction in, 88–89
  painting in, 89–93
  right-click menus for caption icon, 85–86

Programming (Continued)
  scratch program, 79–84
  semaphores in, 110–114
  solid rectangles
    with ExtTextOut, 100–102
    with StretchBlt, 102–103
  window coverage determination, 93–95
Programs
  supporting data for, 451
  unkillable, 422–423
Property sheet pages, 414–415
Protected mode, 73
Psychic debugging, 410–412
Pure virtual functions, 281
__purecall symbol, 280–282
Purity *vs.* pragmatism, 249–250
Puzzle Collection, 489–490

Quarks, 347
QueryInterface method, 274–275, 303–305
Questions, unanswerable, 7, 8–9
Queued messages, 358
QuickRes PowerToy, 36

Rat Poker, 489–490
Read-only folder property, 59
ReadProcessMemory function, 316
Rebasing DLLs, 254, 351–353
Recalculating *vs.* saving, 254
Rectangles, solid
  ExtTextOut for, 100–102
  StretchBlt for, 102–103
Red dot on car door locks, 15
Red Moon Desert, 26
Redirected folders, 447–450
Redundant casts to LPARAM, 321–322
Reflexivity rule, 243
Regions in geopolitics, 439
Registering dialog classes, 160
Registry
  for blocked applications, 286
  for document printing order, 66–67
  function names for, 322–324
  "hive" name for, 312
  Run key, 54–55
  Shell Folders key, 294–295
regsvr32 program, 451
Relative pointers, 318
Release method, 274
ReleaseSemaphore function, 110–112
Remote Desktop Connection, 440–443
Removing options *vs.* disabling, 12
ReplyMessage function, 363
Reprimands, whimsical embarrassment for, 493–494
Reserved filenames, 290–292
Resizing dialog boxes, 222–224
Resources
  in 16-bit Windows, 312–313
  undocumented, 305
Restoring window position, 122–123
Return address prediction, 251–253
Return addresses, 242–243
Return values
  dialog procedures, 151–153
  HANDLE, 331–333
_ReturnAddress intrinsic, 242–243
Right-click menus
  boldface on, 62
  for capture icon, 85–86
  timed, 138–139
Roaming user profiles
  handling, 445–447
  redirected folders for, 447–450
Rorschach test, 25–26
Rotating text, 375–379
Round Clock PowerToy, 36
Rumors columns, 28–29
Run registry key, 54–55

Sample URLs, 483–484
Saving *vs.* recalculating, 254
SBCS (single byte character sets), 431
Schedule Chicken, 327
Scratch program, 79–84
Scratch windows, 135

ScratchAccessible class, 477–480
Screen reader, 57
ScriptStringAnalyse function, 110
Secondary monitors, 467–470
Seconds display on taskbar clock, 50–51
Sections, shared, 398–402
Security, 393
  debugging, 397–398
  file hiding, 394–395
  intranet, 402
  passwords, 395–396
  shared sections, 398–402
  uncertified drivers, 396–397
  world-writable files, 393–394
Selectors, 73–75, 318–319, 321–322
Selvin, Joel, 27
Semaphores
  auto-reset events as, 112–114
  floppy disks as, 492
  owners for, 110–112
SendInput function, 358, 372
SendMessage function, 358, 361–363
SendMessageCallback function, 362, 368–369
SendMessageTimeout function, 362, 369
SendNotifyMessage function, 370–371
Sent messages
  generated, 358–362
  life of, 363–364
Separation through juxtaposition, 17
Servers, paging on, 253–254
Service packs
  build numbers for, 39–40
  and CD autoplay settings, 406–407
Services, 394
SetDCBrushColor function, 98
SetDialogFocus function, 204, 227
SetDlgMsgResult macro, 156
SetFileAPIsToOEM function, 390
SetFileAttributes function, 54
SetFocus function, 227–228
SetLastError function, 340
setlocale directive, 383, 385
SetMenuDefaultItem function, 62
SetProcessAffinityMask function, 411
SetProcessDPIAware function, 467
SetTextAlign function, 105–106
SetTimer function
  for clock painting, 89
  for timed message boxes, 134
  for timed context menus, 139
Settings, 3–4, 6
Setup
  for boot sectors, 409–410
  operating system file versions in, 7–9
SetWindow Placement function, 122
SetWindowContextHelpId function, 188, 203
SetWindowFont macro, 202–203
SetWindowLongPtr function, 136, 152–153, 201
SetWindowPlacement function, 471
SetWindowPos function, 50, 471
SHAlloc function, 324–329
Shared memory, 398–402, 426
Shared resources, 394
Shared sections, 398–402
Shell Folders key, 294–296
Shell verbs, 66–67
Shell32.dll file, 305
ShellExecute function, 315
Sherlock tool, 329–330
SHGetFolderPath function, 296
SHGetMalloc function, 324–329
SHGetSpecialFolderLocation function, 295
SHLoadOLE function, 329
Shortcut Target Menu PowerToy, 36
Show Desktop feature, 60–61
SHSetLocalizedName function, 56
Shutdown, Start button for, 1–2
Sibling windows, 436
Silent driver installs, 396–397
Simulation, keyboard input, 371–372
Single byte character sets (SBCS), 431
16-bit DOS and Windows
  backward compatibility for, 288–290
  resource memory management, 312–313

16-bit templates
  classic, 164–172
  extended, 181–187
Size
  dialog boxes, 222–224
  dialog controls, 202
  operating system, 31
  structures, 418–421
sizeof operator, 279–280
Small arrow icon, 53
Smuggling integers inside pointers, 453
Software issues, 239
  daylight saving time, 239–241
  interface contracts, 245–248
  memory leaks
    from caches, 259–266
    identifying, 258–259
  optimization, 250–253
  performance
    holistic view, 255–256
    polling, 257
  pragmatism *vs.* purity, 249–250
  return addresses, 242–243
  saving *vs.* recalculating, 254
  server paging, 253–254
  sort comparisons, 243–245
  tests, 35
  timestamp changes, 241–242
Solid rectangles, drawing
  with ExtTextOut, 100–102
  with StretchBlt, 102–103
Solitaire, Spider, 488
Sorting
  filenames, 347–350
  rules for, 243–245
Sorting It All Out, 373
Special Edition box, 25
Spell checkers
  complaints about, 26–27
  dictionaries, 27
SPI_SETFASTTASKSWITCH setting, 58
Spider Solitaire, 488
Spinner controls, 354–355
SS_CENTERIMAGE style, 102, 466
SS_NOPREFIX style, 236
SS_REALSIZECONTROL style, 465
Stacks
  in backward compatibility, 285–286
  return address, 251–253
  structures allocated on, 420
Start button, 1–2, 51
Start menu
  Intellimenus on, 403–404
  pin list, 404–406
Startup sound, 27–28
Stealing
  animations, 305
  passwords, 395–396
Stealth overclocked computers, 149
STGMEDIUM structure, 78, 319
Strangely named functions, 22–24
Stress testing, 258
StretchBlt function
  for solid rectangles, 102–103
  working with, 465
Strings
  displaying, 103–110
  null-terminated, 164
STRRET structure, 429–430
Structures
  size checks, 418–421
  undocumented, 284–286
Substitution principle, 244
Subsystems, compatibility, 289
Sunken client areas, 432
__super keyword, 158
Superset versions of Windows, 433
Supporting data for programs, 451
Syscall traps, 30–31
System attribute, 55–56
System button, 1–2
SYSTEM font, 104, 353–354
System menu, 344
System policies, 33, 405–406
System Properties memory report, 29
SystemParametersInfo function, 354

systray.exe program, 48
_T macro, 335
TA_UPDATECP mode, 105
Tab key for navigation, 229–232
Tab order in dialog boxes, 221
Tables of function pointers, 273
TABTHETEXTOUTFORWIMPS function, 24
Tag bits, 452
Task Manager, killed processes in, 424–425
Taskbar
  blank buttons on, 59–60
  clocks on, 50–51
  default position, 49–50
  on desktop, 343
  grouping on, 9–11
  pre-history, 342
  Start indicator on, 51
  in tiny footprint mode, 63–64
  vs. tray, 47–49
  in work area, 470
Taxes
  accessibility, 472–480
  anti-aliased fonts and ClearType, 459–462
  Fast User Switching and terminal services, 443–444
  geopolitics, 439
  Hierarchical Storage Management, 438–439
  high DPI displays, 462–467
  intermittent network connectivity, 457–459
  large address spaces, 451–455
  multiple monitors, 467–470
  multiple users, 444–445
  My Documents *vs.* Application Data, 450–451
  pop-up windows, 471–472
  power management, 437, 455–457
  redirected folders, 447–450
  Remote Desktop Connection and painting, 440–443
  roaming user profiles, 445–447
  work area, 470–471
TCP/IP protocol, 144
Teletypewriters, 334
%TEMP% directory, 451
Templates, dialog. See Dialog templates
Temporary Internet Files directory, 451
1033 directory, 379
Terminal services, 443–444
TerminateProcess function, 424
Terminators, line, 334–335
Text
  GetWindowText for, 41–42
    operation, 42–43
    rules in, 43–46
  managing, 41–42
  monochrome, 459–462
  multilingual, 103–110
  rotating, 375–379
Text files, Ctrl+Z in, 333–334
Text label colors for files, 56–57
TEXT macro, 335
_TEXT macro, 335
TextOut function
  font-linked-enabled version, 104–110
  with Remote Desktop Connection, 442
this pointer, 276–278
Threads and messages, 368
Threats, death, 486–487
3D shadow and outlining, 16–17
32-bit templates
  classic, 172–181
  extended, 187–194
Thunks, adjustor, 274–275
Tickets to Windows 95 launch, 355–356
Tiling effects, brushes for, 95–98
Time bomb, 70
Time zones, 19–20, 240
Timed context menus, 138–139
Timed message boxes, 133–134, 136–138
Timeouts for window messages, 369
Timers in painting, 89–92
Timestamp changes, 241–242
Tiny footprint mode, 63–64
Toasters, 6
Tokens, changing meanings of, 492–493

Topmost windows, 58, 436
Transitivity rule, 243–244
TranslateAccelerator function, 216
Translucent plastic floppy drives, 485–486
Traps
  syscall, 30–31
  TerminateProcess function, 424
Tray *vs.* taskbar, 47–49
  See also Notification icons
Tree view control, 429
Tunneling, file system, 346–347
Tweak UI, 32–33, 37
TYPE command, 380
Typeahead, 335–336, 343–344
Typographical errors, 27, 326

UAE (Unrecoverable Application Error), 74
Ugly boxes in string display, 103–110
UI-modality *vs.* code-modality, 123–126
UMA (Unified Memory Architecture) machine, 29
Unanswerable questions in dialogs, 7, 8–9
UNC paths, 292–293
Uncertified drivers, 70, 396–397
Undocumented behavior, 286–288
Undocumented resources, 305
Undocumented structures, 284–286
Unexpected dialogs, 3–5, 489
ungetch function, 431–432
Ungrouping on taskbar, 9–11
Unicode
  case mapping in, 373–375
  and code pages, 380, 383–384, 390
  converting with ANSI, 391–392
UNICODE macro, 335
_UNICODE macro, 335
Unified Memory Architecture (UMA) machine, 29
Uniscribe library, 109–110
Unkillable programs, 422–423
UnlockResource function, 313
Unnoticed changes, 336–337
Unrecoverable Application Error (UAE), 74
Unsafe device removal dialog, 407–408
Unwanted files/folders opened at logon, 54–56
Up-down controls, 354–355
URLs, sample, 483–484
Usability sessions, 67
USB Cart of Death, 147
USB devices
  floppy drives, 485–486
  removing, 407–408
User interface, 1
  Advanced Options dialog, 57
  best settings, 6
  boldface on menus, 62
  dialog box default answers, 3–5
  document printing order, 66–67
  dragging files, 64–65
  ellipses on menus, 13
  evolution of, 16–17
  expert mode, 2–3
  "For test/evaluation purposes only," 70
  icon order, 58–59
  interior door locks, 15–16
  Links folder, 65–66
  memory reporting in, 69–70
  Minimize All *vs.* Show Desktop, 60–61
  operating system file versions, 7–9
  options, disabling *vs.* removing, 12
  overlay icons, 53–54, 256
  product support for, 67–69
  read-only property for folders, 59
  Start button, 1–2
  taskbar
    blank buttons on, 59–60
    grouping on, 9–11
  text label colors for files, 56–57
  tiny footprint mode, 63–64
  unwanted files/folders, 54–56
  vending machines, 13–15
  Web site icons, 62–63
User profiles, roaming
  handling, 445–447
  redirected folders for, 447–450
User Shell Folders key, 296

User switching, 443–444
%USERPROFILE% directory, 446
Users
  death threats from, 486–487
  multiple, 444–445
  vs. programs, 422–423
UTC (coordinated universal time)
  and BIOS clock, 301–302
  and daylight saving time, 239–240
UTF-8 characters, 431–432

Vending machine user interface, 13–15
Vendors, misspellings by, 144–145
Verbs, shell, 66–67
Versions
  checking, 302–303
  operating system files, 7–9
  Windows superset, 433
Vertical taskbar, 51
Vertical text, 376–379
Video displays, high DPI, 462–467
Video drivers
  certification process, 145
  DirectX, 298–299
Virtual functions, 280–282
Virtual memory, 71–72
Visible windows, painting, 89–93
Visual C++ compiler, 267
  adjustor thunks in, 274–275
  COM object layout in, 272–274
  destructors in, 267–272
  pointers to member functions in, 276–280
  __purecall in, 280–282
Visual Studio compiler, 383
Visual style backward compatibility, 309
vtables, 273, 281–282, 401

W in WPARAM, 311–312
WAIT_ABANDONED status code, 425–427
WaitForSingleObject function, 111, 333
WaitMessage function, 213, 359
Waking receivers, 359
Wallpaper, Red Moon Desert, 26
Web site icons, 62–63
Whimsical bug reports, 482–483
Whimsical embarrassment, 493–494
White windows, 118–119
WHQL (Windows Hardware Quality Labs), 143–146
WideCharToMultiByte function, 385
Width of dialog controls, 202
Win32 design issues, 417
  desktop disabling, 433–434
  HWND_TOP vs. HWND_TOPMOST, 435–436
  hyperlinks in notification icon balloon tips, 427–428
  menu nesting limits, 435
  module load failures, 417–418
  programs vs. users, 422–423
  STRRET structure, 429–430
  structure size checks, 418–421
  sunken client areas, 432
  Task Manager and killed processes, 424–425
  TerminateProcess, 424
  transitioning to, 75–76
  tree items, 429
  UTF-8 code page, 431–432
  WAIT_ABANDONED, 425–427
  Windows superset versions, 433
  WM_DEVICECHANGE, 421–422
Window management, 115
  coverage determination, 93–95
  desktop window, 120–121
  disabling and enabling windows, 121–122
  GWLP_USERDATA in, 136
  hollow brush, 119
  menu destruction, 88–89
  messages for destroyed windows, 139–140
  minimized, 341–343
  modal program interaction, 132
  modal UI owners, 129–132
  restoring position, 122–123
  scratch windows, 135
  timed context menu, 138–139

Window management (Cotinued)
timed message boxes, 133–134, 136–138
UI-modality vs. code-modality, 123–126
white windows, 118–119
WM_MOUSEENTER messages, 118
WM_MOUSEMOVE messages, 115–118
WM_QUIT messages, 126–129, 206, 211, 213
Window messages, 357
keyboard input simulation, 371–372
myths, 370–371
sender and poster identification, 371
SendMessageCallback function, 368–369
SendMessageTimeout function, 369
sent and posted, 358–362
generated, 365–367
life of, 363–365
WINDOWPLACEMENT structure, 471
Windows operating systems
broadcast-based mechanisms in, 340–341
superset versions of, 433
Windows 2000, 403
boot sectors, 409–410
CPU usage, 411–412
DS_SHELLFONT style, 412–415
processor affinity, 410–411
Start menu Intellimenus, 403–404
unsafe device removal dialog, 407–408
Windows 95
anti-piracy hologram, 25–26
booting in, 20–22
build numbers, 38–40
code bloat, 31
hard drive light flashes, 29–30
hardware, 141–143
installing, 34–35
launch, 19, 355–356
M3 beta release, 294
martial arts logon picture, 26–27
memory size report, 29
PowerToys, 35–37
product-support call costs, 32
Rumors columns, 28–29
service packs, 39–40
software tests, 35
Special Edition box, 25
startup sound, 27–28
strangely named functions, 22–24
syscall traps, 30–31
time zones, 19–20
Tweak UI, 32–33
Windows Hardware Quality Labs (WHQL), 143–146
Windows Presentation Foundation, 225
Windows XP, 403
CD autoplay settings, 406–407
CD ejection, 408–409
Comments? button, 408
Start menu pin list, 404–406
unsafe device removal dialog, 407–408
windowsx.h header file, 80
WinExec function, 315
WINLOGON.EXE program, 423
WinMain function
hPrevInstance parameter, 316–317
in scratch program, 84
Winsock functions, 340
WINVER setting, 419
Wissink, Kathy, 388
WM_ACTIVATE message, 230, 232
WM_CANCELMODE message, 138–139
WM_COMMAND message, 216, 364
WM_CONTEXTMENU message, 85–86
WM_CREATE message, 81, 83
WM_CTLCOLOR messages, 98, 99, 119
WM_DESTROY message, 81, 83
WM_DEVICECHANGE message, 421–422
WM_DRAWITEM message, 462
WM_ERASEBKGND message, 462
WM_FONTCHANGE message, 340
WM_GETDLGCODE message, 232–235
WM_GETOBJECT message, 478–479
WM_GETTEXT message, 41–44
WM_INITDIALOG message, 156, 161, 219
WM_KEYDOWN message, 224
WM_MOUSEENTER message, 118

WM_MOUSEMOVE message, 115–118, 366–367
WM_NCCREATE message, 41
WM_NEXTDLGCTL message, 227–228
WM_NOTIFY message, 222, 371
WM_NULL message, 212
WM_PAINT message, 81–82, 89–91, 254, 366
WM_POWERBROADCAST message, 456–457
WM_PRINTCLIENT message, 82, 91
WM_QUERYENDSESSION message, 421
WM_QUIT message
  generated on the fly, 366
  in message loops, 206, 211, 213–214
  and modality, 126–129
  for timed message boxes, 133–134
WM_SETCURSOR message, 118, 158
WM_SETFOCUS message, 232
WM_SETFONT message, 202
WM_SETTEXT message, 42
WM_SIZE message, 81, 83
WM_TABSTOP message, 229
WM_TIMER message, 365–367
WM_USER message, 229
WM_WTSSESSIONCHANGE message, 443–444
Work area, 343, 470–471
World-writable files, 393–394
WPARAM, 311–312
wprintf function, 383–385
WriteConsole function, 385
WS_CAPTION style, 198
WS_CHILD style, 198, 220
WS_EX_APPWINDOW style, 59
WS_EX_CLIENTEDGE style, 432
WS_EX_CONTROLPARENT style, 198, 220–221
WS_EX_NOPARENTNOTIFY style, 203
WS_GROUP style, 229, 236
WS_SYSMENU style, 198
WS_TABSTOP style, 229, 236
WS_VISIBLE style, 220
WSASetLastError function, 340

X buttons
  disabled, 237
  in notification balloons, 52
XADD instruction, 339
xcopy
  death threat, 486–487
  for Windows 95 installation, 34–35

Z-order, 58, 121, 436
0409 directory, 379

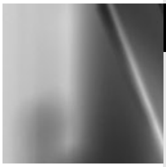
informIT

www.informit.com

YOUR GUIDE TO IT REFERENCE

Articles

Keep your edge with thousands of free articles, in-depth features, interviews, and IT reference recommendations – all written by experts you know and trust.



Online Books

Answers in an instant from **InformIT Online Book's** 600+ fully searchable on line books. For a limited time, you can get your first 14 days **free**.

POWERED BY
Safari®
TECH BOOKS ONLINE®

Catalog

Review online sample chapters, author biographies and customer rankings and choose exactly the right book from a selection of over 5,000 titles.

Wouldn’t it be great

if the world’s leading technical publishers joined forces to deliver their best tech books in a common digital reference platform?

They have. Introducing
InformIT Online Books
powered by Safari.

- **Specific answers to specific questions.**
InformIT Online Books’ powerful search engine gives you relevance-ranked results in a matter of seconds.

- **Immediate results.**
With InformIT Online Books, you can select the book you want and view the chapter or section you need immediately.

- **Cut, paste and annotate.**
Paste code to save time and eliminate typographical errors. Make notes on the material you find useful and choose whether or not to share them with your work group.

- **Customized for your enterprise.**
Customize a library for you, your department or your entire organization. You only pay for what you need.

Get your first 14 days FREE!

For a limited time, InformIT Online Books is offering its members a 10 book subscription risk-free for 14 days. Visit **http://www.informit.com/onlinebooks** for details.

POWERED BY Safari® TECH BOOKS ONLINE

informIT Online Books

informit.com/onlinebooks

Register Your Book

at www.awprofessional.com/register

You may be eligible to receive:

- Advance notice of forthcoming editions of the book
- Related book recommendations
- Chapter excerpts and supplements of forthcoming titles
- Information about special contests and promotions throughout the year
- Notices and reminders about author appearances, tradeshows, and online chats with special guests

Contact us

If you are interested in writing a book or reviewing manuscripts prior to publication, please write to us at:

Editorial Department
Addison-Wesley Professional
75 Arlington Street, Suite 300
Boston, MA 02116 USA
Email: AWPro@aw.com

Visit us on the Web: http://www.awprofessional.com