Part of the five-volume
Microsoft® Win32® Developer's Reference Library

Microsoft®

The essential reference to Win32®
technologies and APIs

David Iseminger
Series Editor
www.iseminger.com

Microsoft® Windows® Shell

BASED ON
msdn™ library

Microsoft®

The essential reference to Win32® technologies and APIs

David Iseminger
Series Editor

Microsoft® Windows® Shell

BASED ON
msdn™ library

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Cataloging-in-Publication Data
Iseminger, David, 1969-
Microsoft Win32 Developer's Reference Library / David Iseminger.
p. cm.
ISBN 0-7356-0816-4
1. Microsoft Win32. 2. Operating systems (Computers) I. Title.
QA76.76.O63 I74 1999
005.26'8--dc21 99-045609
CIP

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 WCWC 4 3 2 1 0 9

Distributed in Canada by Penguin Books Canada Limited.

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

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at mspress.microsoft.com.

The example companies, organizations, products, people, and events depicted herein are fictitious. No association with any real company, organization, product, person, or event is intended or should be inferred.

Acquisitions Editor: Ben Ryan
Project Editor: Wendy Zucker

Part No. 097-0002310

Acknowledgements

Acknowledgements are often tricky things; generally, the day after books are printed you think of someone who absolutely should have been recognized, whom you now have rudely omitted. You'd think authors would keep an ongoing list. Oh well, here goes:

First, thanks to Ben Ryan at Microsoft Press for sharing my enthusiasm about the series idea, and for keeping up with the myriad of issues that cropped up, and for managing the business details associated with publishing this series. Thanks also to Steve Guty at Microsoft Press for seeing certain publishing issues through the wringer.

Wendy Zucker kept in step with the difficult and tight schedule at Microsoft Press, and orchestrated things in the way only project editors can endure. John Pierce was also instrumental in seeing the publishing process through completion; many thanks to both of them. The cool Win32 cover art was created by Greg Hickman—thanks for the excellent work; I'm a firm believer that artwork and packaging are integral to the success of a project. Marketing acknowledgements go out to Jocelyn Paul, for her coordination efforts with MSDN and her other unsung victories.

On the SDK side of things, thanks to Morgan Seeley for introducing me to the editor at Microsoft Press, and thereby routing this series to the right place. Throughout the process, Julie Solon provided lots of Win32 feedback and helped gather feedback from others, all of which was quite helpful in compiling the right collection of technologies…thanks to Julie for the help on that. Guy Smith pointed me to the information I needed for Volumes 4 and 5, and was always very responsive.

On the developer side of things, thanks go out to Lars Opstad and Paramesh Vaidyanathan for their help and openness, respectively, with letting me provide the common coding errors found in Chapter 5 of each of these volumes. Thanks on my behalf, and on behalf of anyone who finds that information useful (I'm sure that includes a bunch of people!).

Thanks are also in order for artist-guru David Deyo for transforming my functional "circled i" logo into a 3D piece of art, as well as for his work on the Iseminger.com site. You can see more of his artwork through links found at www.iseminger.com.

Last, but certainly not least, thanks to Margot Hutchison for doing all the things great agents do best.

Contents

Chapter 1: Introduction 1

How the Win32 Library Is Structured 2

How the Win32 Library Is Designed 3

Chapter 2: What's In This Volume? 5

Chapter 3: Using Microsoft Reference Resources 7

The Microsoft Developer Network (MSDN) 8

Comparing MSDN and MSDN Online 8

MSDN Subscriptions 10

MSDN Library Subscription 11

MSDN Professional Subscription 11

MSDN Universal Subscription 11

Purchasing an MSDN Subscription 12

Using MSDN 13

Navigating MSDN 14

Quick Tips 16

Using MSDN Online 17

Navigating MSDN Online 19

MSDN Online Features 20

MSDN Online Registered Users 25

The Windows Programming Reference Series 26

Chapter 4: Finding the Developer Resources You Need 27

Developer Support 27

Online Resources 29

Learning Products 30

Conferences 32

Other Resources 33

Chapter 5: Getting the Most out of Win32 Library Technologies: Part 5 35

Memory Abuse 35

Allocation failures 36

Uninitialized memory 36

Avoiding leaks 37

Don't use freed resources 38

Resource attacks 39

Miscalculations 39
Division by zero 40
Signed versus unsigned variables 40
Floating-point variables 41
Solution Summary 42

Part 2 Introduction 43

Commonly Asked Shell Questions 43
Shell and Common Controls Versions 45
DLL Version Numbers 45
Using DllGetVersion to Determine the Version Number 46
Using DllGetVersion 46
Project Versions 48

Chapter 6: Shell Programmer's Guide 49

Programming the Shell 49
Integrating an Application with the Shell 49
The Shell Namespace 50
Introduction 50
Identifying Namespace Objects 51
Getting a Folder's ID 54
The SHBrowseForFolder Dialog Box 54
Special Folders and CSIDLs 54
An Example of How to Use CSIDLs and SHBrowseForFolder 56
Getting Information About the Contents of a Folder 57
Using the IShellFolder Interface 57
Enumerating the Contents of a Folder 58
Determining Display Names and Other Properties 58
Getting a Pointer to a Subfolder's IShellFolder Interface 59
Determining an Object's Parent Folder 59
Navigating the Namespace 61
An Example of Namespace Navigation 61
Launching Applications 63
Using ShellExecute and ShellExecuteEx 63
An Example of How to Use ShellExecuteEx 64
Managing the File System 66
Per-User File Management 66
The My Documents and My Pictures Folders 67
Connected Files 68
Moving, Copying, Renaming, and Deleting Files 68

An Example of Managing Files with SHFileOperation 70
Adding Files to the Shell's List of Recent Documents 72
Managing Printers 72
Printer Management 72
Printing Files with ShellExecuteEx 72
Transferring Shell Objects with Drag-Drop and the Clipboard 73
How Drag-Drop Works with Shell Objects 73
Clipboard Data Transfers 74
Drag-Drop Data Transfers 75
The Shell Data Object 76
How Data Objects Work 77
How a Source Creates a Data Object 79
How a Target Handles a Data Object 84
Using the Drag-Drop Helper Object 88
Shell Clipboard Formats 89
Formats for Transferring File System Objects 90
Formats for Transferring Virtual Objects 94
Formats for Communication Between Source and Target 94
Handling Shell Data Transfer Scenarios 97
General Guidelines 98
Copying File Names from the Clipboard to an Application 99
Copying the Contents of a Dropped File into an Application 100
Handling Optimized Move Operations 102
Handling Delete-on-Paste Operations 104
Transfering Data to and from Virtual Folders 105
Dropping Files on the Recycle Bin 106
Creating and Importing Scrap Files 107
Dragging and Dropping Shell Objects Asynchronously 109
Extending the Shell 110
Creating a File Association 110
Defining a File Class 111
Defining Attributes for a File Class 112
Excluding an Application from the Open With Dialog Box 113
Customizing Icons 114
Assigning a Custom Icon to a File Class 114
Assigning a Custom Icon and Label to a Drive Letter 116
Extending Context Menus 116
Context Menus for File System Objects 116
Verbs 118

Extending the Context Menu for a File Class 119
Extending the New Submenu 122
Customizing Folders with Desktop.ini 124
Using Desktop.ini Files 124
Creating a Desktop.ini File 125
Creating an AutoPlay-Enabled CD-ROM Application 128
Creating an AutoPlay-Enabled Application 129
Creating an Autorun.inf File 129
Tips for Writing AutoPlay Startup Applications 130
Autorun.inf Commands 131
Enabling and Disabling AutoPlay 135
Suppressing AutoPlay Programmatically 135
Using the Registry to Disable AutoPlay 137
AutoPlay for Other Types of Storage Media 138

Chapter 7: Shell Interfaces 139

Shell Interface Overview 139
Shell Interfaces 139
IACList 139
IACList2 142
IActiveDesktop Interface 144
IAsyncOperation 160
IAutoComplete 165
IAutoComplete2 169
IColumnProvider 172
ICommDlgBrowser 176
ICommDlgBrowser2 179
IContextMenu 182
IContextMenu2 188
IContextMenu3 190
ICopyHook 192
ICurrentWorkingDirectory 194
IDeskBand 196
IDockingWindow 198
IDockingWindowFrame 201
IDragSourceHelper 205
IDropTargetHelper 208
IDockingWindowSite 213
IEmptyVolumeCache 216
IEmptyVolumeCache2 223

IEmptyVolumeCacheCallBack ... 226
IEnumExtraSearch ... 229
IEnumIDList ... 232
IExtractIcon ... 236
IExtractImage ... 240
IExtractImage2 ... 243
IFileViewer ... 245
IFileViewerSite ... 248
IInputObject ... 249
IInputObjectSite ... 252
INewShortcutHook ... 253
INotifyReplica ... 258
IObjMgr ... 260
IPersistFolder ... 261
IPersistFolder2 ... 263
IPersistFolder3 ... 264
IProgressDialog ... 267
IQueryAssociations ... 277
IQueryInfo ... 284
IReconcilableObject ... 286
IReconcileInitiator ... 291
IRemoteComputer ... 294
IResolveShellLink ... 295
IRunnableTask ... 297
IShellBrowser ... 301
IShellChangeNotify ... 316
IShellDetails ... 319
IShellExecuteHook ... 322
IShellExtInit ... 324
IShellFolder ... 326
IShellFolder2 ... 343
IShellIcon ... 350
IShellIconOverlay ... 352
IShellIconOverlayIdentifier ... 355
IShellLink ... 359
IShellLinkDataList ... 375
IShellPropSheetExt ... 381
IShellView ... 383
IShellView2 ... 395

ITaskbarList 399
IUniformResourceLocator 403
IURLSearchHook 407

Chapter 8: Shell Functions 409

Shell Functions 409
Shell Callback Functions 481

Chapter 9: Shell Structures 485

Shell Structures 485

Chapter 10: Shell Enumerations and Macros 561

Shell Enumerations 561
Shell Macros 571

Chapter 11: Shell Lightweight Utility APIs 575

String Functions 575
Path Functions 610
Registry Data Types 668
Registry Functions 670
Color Palette Functions 707
Miscellaneous 710

Chapter 12: Shell Messages and Notifications 721

Shell Messages and Notifications 721

Appendix A 751

Appendix B 759

CHAPTER 1

Introduction

Welcome to the *Microsoft Win32 Developer's Reference Library*, your comprehensive reference guide to the Win32 development environment. This library, and the entire Windows Programming Reference Series, is designed to deliver the most complete, authoritative, and accessible reference information available for Windows programming—without sacrificing focus. You'll notice that each book is dedicated to a logical group of technologies or development concerns; this approach has been taken specifically to enable you—the time-pressed and information-overloaded applications developer—to find the information you need quickly, efficiently, and intuitively.

In addition to its focus on Win32 reference material, the Win32 Library contains hard-won insider tips and tricks designed to make your programming life easier. For example, a thorough explanation and detailed tour of the new version of MSDN Online is included, as is a section that helps you get the most out of your MSDN Subscription. Don't have an MSDN subscription, or don't know why you should? I've included information about that too, including the differences among the three levels of MSDN subscriptions, what each level offers, and why you'd want a subscription when MSDN Online is available over the Internet.

Microsoft is fairly well known for its programming, so doesn't it make sense to share some of that knowledge? I thought it made sense, so that's why this—the Windows Programming Reference Series—is the source where you'll find such shared knowledge. Part 1 of each volume contains advice on how to avoid common programming problems. There is a reason for including so much reference, overview, shared-knowledge, and programming information about Win32 in a single publication: the Win32 Library is geared toward being your one-stop printed reference resource for the Win32 programming environment.

To ensure that you don't get lost in all the information provided in the Win32 Library, each volume's appendixes provide an all-encompassing programming directory to help you easily find the particular programming element you're looking for. This directory suite, which covers all the functions, structures, enumerations, and other programming elements found in Win32, gets you quickly to the volume and page you need, and also provides an overview of Microsoft technologies that would otherwise take you hours of time, reams of paper, and potfuls of coffee to compile yourself.

How the Win32 Library Is Structured

The Win32 Library consists of five volumes, each of which focuses on a particular area of the Win32 programming environment. The programming areas into which the five Win32 Library volumes have been divided include:

Volume 1: Base Services

Volume 2: User Interface

Volume 3: GDI (Graphics Device Interface)

Volume 4: Common Controls

Volume 5: The Windows Shell

Dividing the Win32 Library—and therefore, dividing Win32—into these functional categories enables a software developer who's focusing on a particular programming area (such as the user interface) to maintain that focus under the confines of one volume. This approach enables you to keep one reference book open and handy, or tucked under your arm while researching that aspect of Windows programming on sandy beaches, without risking back problems (from toting around a 2,000-page Win32 tome), and without having to shuffle among multiple, less-focused books.

Within each Win32 Library volume there is also a deliberate structure. This per-volume structure has been created to further focus the reference material in a developer friendly manner and to enable developers to easily gather the information they need. To that end, each volume in the Win32 Library has the following parts:

Part 1: Introduction and Overview

Part 2: Reference

Part 3: Windows Programming Directory

Part 1 provides an introduction to the Win32 Library and to the Windows Programming Reference Series (what you're reading now), and a handful of chapters designed to help you get the most out of Win32, MSDN and MSDN Online, including a collection of insider tips and tricks. Just as each volume's Reference section (Part 2) contains different reference material, each volume's Part 1 contains different tips and tricks. To ensure that you don't miss out on some of them, make sure you take a look at Part 1 in each Win32 Library volume.

Part 2 contains the Win32 reference material particular to its volume, but it is *much* more than a simple collection of function and structure definitions. Because a comprehensive reference resource should include information about *how to use* a particular technology, as well as its definitions of programming elements, the information in Part 2 combines complete programming element definitions as well as instructional and explanatory material for each programming area.

Part 3 is the directory of Windows programming information. One of the biggest challenges of the IT professional is finding information in the sea of available resources, and Windows programming is no exception. In order to help you get a handle on Win32 programming references and Microsoft technologies in general, Part 3 puts all such information into an understandable, manageable directory that enables you to quickly find the information you need.

How the Win32 Library Is Designed

The Win32 Library, and all libraries in the Windows Programming Reference Series, is designed to deliver the most pertinent information in the most accessible way possible. The Win32 Library is also designed to integrate seamlessly with MSDN and MSDN Online by providing a look-and-feel that is consistent with their electronic counterparts. In other words, the way that a given function reference appears on the pages of this book has been designed specifically to emulate the way that MSDN and MSDN Online present their function reference pages.

The reason for maintaining such integration is simple: make it easy for you—the developer of Windows applications—to use the tools and get the ongoing information you need create quality programs. By providing a "common interface" among reference resources, your familiarity with the Win32 Library reference material can be immediately applied to MSDN or MSDN Online, and vice versa. In a word, it means *consistency*.

You'll find this philosophy of consistency and simplicity applied throughout Windows Programming Reference Series publications. I've designed the series to go hand-in-hand with MSDN and MSDN Online resources. Such consistency lets you leverage your familiarity with electronic reference material, and apply that familiarity to let you get away from your computer if you'd like, take a book with you, and—in the absence of keyboards and e-mail and upright chairs—get your programming reading and research done. Of course, each of the Win32 Library books fits nicely right next to your mouse pad as well, even when opened to a particular reference page.

With any job, the simpler and more consistent your tools are, the more time you can spend doing work rather than figuring out how to use your tools. The structure and design of the Win32 Library provides you with a comprehensive, pre-sharpened toolset to build compelling Windows applications.

CHAPTER 2

What's In This Volume?

Volume 5 of the *Microsoft Win32 Developer's Reference Library* is all about the Windows shell. With the programmatic elements and programming techniques detailed in this volume of the Win32 Library—*Volume 5: The Windows shell*—you can enhance or change all sorts of different aspects of the Windows shell.

When programming to the Windows shell, you have to be prepared to deal with versioning issues that are associated with Windows shell programming. Almost all of the programmatic elements associated with the shell are contained within three .dll files (Comctl32.dll, Shell32.dll, and Shlwapi.dll), and with each of these .dll files there are versioning issues that must be kept in check throughout the development process. Windows Common Controls share the versioning requirements of the Windows shell, so when you're programming to either the Windows shell (explained in this volume of the Win32 Library) or with Common Controls (explained in Volume 4 of the Win32 Library), you must deal with the versioning requirements.

What are the versioning requirements, you ask? The introduction to Part 2 of this volume (and Volume 4 of the Win32 Library) discusses these caveats in detail, and arms you with all the information you need to keep the associated requirements straight. You should read this explanatory introduction to Part 2 before jumping into the programmatic use of any of the Windows shell programmatic elements detailed in this volume of the Win32 Library.

The first chapter—found at the beginning of Part 2—provides guidelines to programming the Windows shell:

Chapter 6: Shell Programming Guide

After this guideline-centric chapter, you'll find reference chapters that provide you with the programmatic reference information you need to develop Windows applications that make use of the Windows shell:

Shell Interfaces
Shell Functions
Shell Structures
Shell Enumerations and Macros
Shell Lightweight Utility APIs
Shell Messages and Notifications

Programming to the Windows shell has some caveats and versioning requirements that developers must keep in mind throughout the development process. These versioning requirements, which are shared with common controls (see Volume 4 in the Win32 Library), are tied to the versions of three key .dll files. You're strongly urged to take a thorough look at the introduction to Part 2 in this volume, which details the issues that programmers must deal with when programming to the Windows shell.

CHAPTER 3

Using Microsoft Reference Resources

These days it isn't the availability of information that's the problem, it's the availability of information. You read that right...but I'll clarify.

Not long ago, getting the information you needed was a challenge because there wasn't enough of it. To find the information you needed, you had to find out where such information might be located and then actually get access to that location, because it wasn't at your fingertips or on some globally available backbone, and such searching took time. In short, the availability of information was limited.

Today, information surrounds us and sometimes stifles us. We're overloaded with too much information, and if we don't take measures to filter out what we don't need to meet our goals, soon we become inundated and unable to discern what's "junk information" and what's information that we need to stay current, and therefore competitive. In short, the overload of available information makes it more difficult for us to find what we *really* need, and wading through the deluge slows us down.

This truism applies to Microsoft's own reference material as well—not because there is information that isn't needed, but rather because there is so much information that finding what *you* need can be as challenging as figuring out what to do with it once you have it. Developers need a way to cut through the information that isn't pertinent to them and to get what they're looking for. One way to ensure you can get to the information you need is to know the tools you use; carpenters know how to use nail guns, and it makes them more efficient. Bankers know how to use ten-keys, and it makes them more adept. If you're a developer of Windows applications, two tools you should know are MSDN and MSDN Online. The third tool for developers—reference books from the Windows Programming Reference Series—can help you get the most out of the first two.

Books in the Windows Programming Reference Series, such as those found in the *Microsoft Win32 Developer's Reference Library*, provide reference material that focuses on a given area of Windows programming. MSDN and MSDN Online, in comparison, contain all of the reference material that all Microsoft programming technologies have amassed over the past few years, and create one large repository of information. Regardless of how well such information is organized, there's a lot of it, and if you don't know your way around, finding what you need (even though it's in there, somewhere) can be frustrating and time-consuming and just an overall bad experience.

This chapter will give you the insight and tips you need to navigate MSDN and MSDN Online, and to enable you to use each of them to the fullest of their capabilities. Also, other Microsoft reference resources are investigated, and by the end of the chapter, you'll know where to go for the Microsoft reference information you need (and how to quickly and efficiently get there).

The Microsoft Developer Network (MSDN)

MSDN stands for Microsoft Developer Network, and its intent is to provide developers with a network of information to enable the development of Windows applications. Many people have either worked with MSDN or have heard of it, and quite a few have one of the three available subscription levels to MSDN, but there are many, many more who don't have subscriptions and could use some concise direction on what MSDN can do for a developer or development group. If you fall into any of these categories, this section is for you.

There is some clarification to be done with MSDN and its offerings; if you've heard of MSDN, or have had experience with MSDN Online, you may have asked yourself one of these questions during the process of getting up to speed with either resource:

- Why do I need a subscription to MSDN if resources such as MSDN Online are accessible for free over the Internet?
- What are the differences among the three levels of MSDN subscriptions?
- What happened to Site Builder Network...or, What is this Web Library?
- Is there a difference between MSDN and MSDN Online, other than the fact that one is on the Internet and the other is on a CD? Do their features overlap, separate, coincide, or what?

If you have asked these questions, then lurking somewhere in the back of your thoughts has probably been a sneaking suspicion that maybe you aren't getting as much out of MSDN as you could. Or, maybe you're wondering whether you're paying too much for too little, or not enough to get the resources you need. Regardless, you want to be in the know, not in the dark.

By the end of this chapter, you will know the answers to all these questions and more, along with some effective tips and hints on how to make the most effective use of MSDN and MSDN Online.

Comparing MSDN and MSDN Online

Part of the challenge of differentiating between MSDN and MSDN Online comes with determining which has the features you need. Confounding this differentiation is the fact that both have some content in common, yet each offers content unavailable with the other. But can their differences be boiled down? Yes, if broad strokes and some generalities are used:

- MSDN provides reference content *and* the latest Microsoft product software, all shipped to its subscribers on CD (or in some cases, on DVD).

- MSDN Online provides reference content *and* a development community forum, and is available only over the Internet.

Each delivery mechanism for the content that Microsoft is making available to Windows developers is appropriate for the medium, and each plays on the strength of the medium to provide its customers with the best presentation of material possible. These strengths and medium considerations enable MSDN and MSDN Online to provide developers with different feature sets, each of which has its advantages.

MSDN is perhaps less immediate than MSDN Online because it gets to its subscribers in the form of CDs that come in the mail. However, MSDN can sit in your CD drive (or on your hard drive), and isn't subject to Internet speeds or failures. Also, MSDN has a software download feature that enables subscribers to automatically update their local MSDN content, over the Internet, as soon as it becomes available, without having to wait for the update CD to come in the mail. The interface with which MSDN displays its material—which looks a whole lot like a specialized browser window—is also linked to the Internet as a browser-like window. To further coordinate MSDN with the immediacy of the Internet, MSDN Online has a section of the site dedicated to MSDN subscribers that enables subscription material to be updated (on their local machines) as soon as it's available.

MSDN Online has lots of editorial and technical columns that are published directly to the site, and are tailored (not surprisingly) to the issues and challenges faced by developers of Windows applications or Windows-based web sites. MSDN Online also has a customizable interface (much like *MSN.com*) that enables visitors to tailor the information that's presented upon visiting the site to the areas of Windows development in which they are most interested. However, MSDN Online, while full of up-to-date reference material and extensive online developer community content, doesn't come with Microsoft product software, and doesn't reside on your local machine.

Since it's easy to become confused about the differences and similarities between MSDN and MSDN Online, it makes sense to figure out a way to quickly identify how and where they depart. Figure 3-1 puts the differences—and similarities—between MSDN and MSDN Online into a quickly identifiable format.

One feature that you will notice is shared between MSDN and MSDN Online is the interface—they are very similar. That's almost certainly a result of attempting to ensure that developers' user experience with MSDN is easily associated with the experience on MSDN Online, and vice versa.

Remember, too, that if you are an MSDN subscriber you can still use MSDN Online and its features. So it isn't an "either/or" question with regard to whether you need an MSDN subscription or whether you should use MSDN Online; if you have an MSDN subscription, you will probably continue to use MSDN Online and the additional features provided with your MSDN subscription.

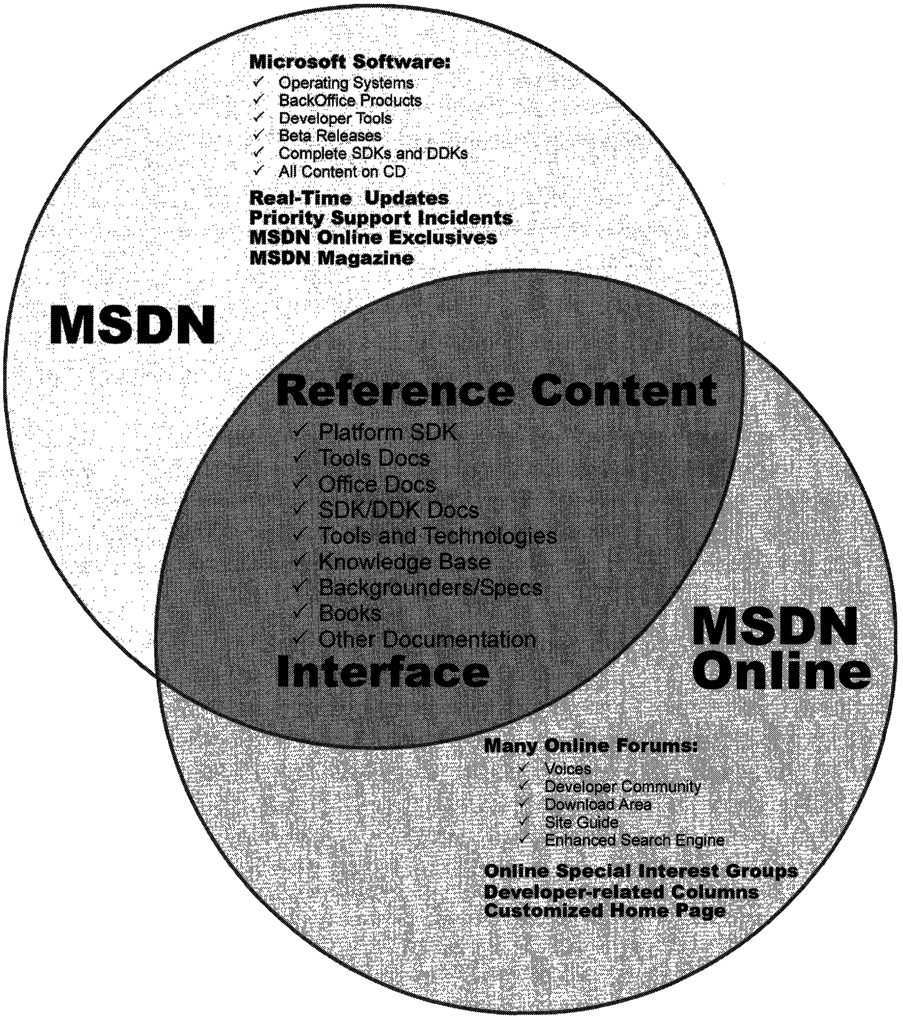


Figure 3-1: The similarities and differences in coverage between MSDN and MSDN Online.

MSDN Subscriptions

If you're wondering whether you might benefit from a subscription to MSDN, but you aren't quite sure what the differences between its subscription levels are, you aren't alone. This section aims to provide a quick guide to the differences in subscription levels, and what each subscription level costs.

There are three subscription levels for MSDN: Library, Professional, and Universal. Each has a different set of features. Each progressive level encompasses the lower level's features, and includes additional features. In other words, with the Professional subscription, you get everything provided in the Library subscription, plus additional features; with the Universal subscription, you get everything provided in the Professional subscription, plus even more features.

MSDN Library Subscription

The MSDN Library subscription is the basic MSDN subscription. While the Library subscription doesn't come with the Microsoft product software that the Professional and Universal subscriptions provide, it does come with other features that developers may find necessary in their development effort. With the Library subscription, you get the following:

- The Microsoft reference library, including SDK and DDK documentation, updated quarterly
- Lots of sample code, which you can cut-and-paste into your projects, royalty free
- The complete Microsoft Knowledge Base—*the* collection of bugs and workarounds
- Technology specifications for Microsoft technologies
- The complete set of product documentation, such as Visual Studio, Office, and others
- Complete (and in some cases, partial) electronic copies of selected books and magazines
- Conference and seminar papers—if you weren't there, you can use MSDN's notes

In addition to these items, you also get:

- Archives of MSDN Online columns
- Periodic e-mails from Microsoft chock full of development-related information
- A subscription to MSDN News, a bi-monthly newspaper from the MSDN folks
- Access to subscriber-exclusive areas and material on MSDN Online

MSDN Professional Subscription

The Professional subscription is a superset of the Library subscription. In addition to the features outlined in the previous section, MSDN Professional subscribers get the following:

- Complete set of Windows operating systems, including release versions of Windows 95, Windows 98, and Windows NT 4 Server and Workstation.
- Windows SDKs and DDKs in their entirety
- International versions of Windows operating systems (as chosen)
- Priority technical support for two incidents in a development and test environment

MSDN Universal Subscription

The Universal subscription is the all-encompassing version of the MSDN subscription. In addition to everything provided in the Professional subscription, Universal subscribers get the following:

- The latest version of Visual Studio, Enterprise Edition
- The BackOffice test platform, which includes all sorts of Microsoft product software incorporated in the BackOffice family, each with special 10-connection license for use in the development of your software products
- Additional development tools, such as Office Developer, Front Page, and Project
- Priority technical support for two additional incidents in a development and test environment (for a total of four incidents)

Purchasing an MSDN Subscription

Of course, all of the features that you get with MSDN subscriptions aren't free. MSDN subscriptions are one-year subscriptions, which are current as of this writing. Just as each MSDN subscription escalates in functionality and features, so too does each escalate in price. Please note that prices are subject to change.

The MSDN Library Subscription has a retail price of $199, but if you're renewing an existing subscription you get a $100 rebate in the box. There are other perks for existing Microsoft customers, but those vary. Check out the Web site for more details.

The MSDN Professional Subscription is a bit more expensive than the Library, with a retail price of $699. If you're an existing customer renewing your subscription, you again get a break in the box, this time in the amount of a $200 rebate. You also get that break if you're an existing Library subscriber who's upgrading to a Professional subscription.

The MSDN Universal Subscription takes a big jump in price, sitting at $2,499. If you're upgrading from the Professional subscription, the price drops to $1,999, and if you're upgrading from the Library subscription level there's an in-the-box rebate for $200.

As is often the case, there are academic and volume discounts available from various resellers, including Microsoft, so those who are in school or in the corporate environment can use their status (as learner or learned) to get a better deal—and in most cases, the deal is much better. Also, if your organization is using lots of Microsoft products, whether MSDN is a part of that group or not, whomever's in charge of purchasing should look into Microsoft Open License program. The Open License program gives purchasing breaks for customers that buy lots of products. Check out *www.microsoft.com/licensing* for more details. Who knows, if your organization qualifies, you could end up getting an engraved pen from your purchasing department, or if you're really lucky maybe even a plaque of some sort for saving your company thousands of dollars on Microsoft products.

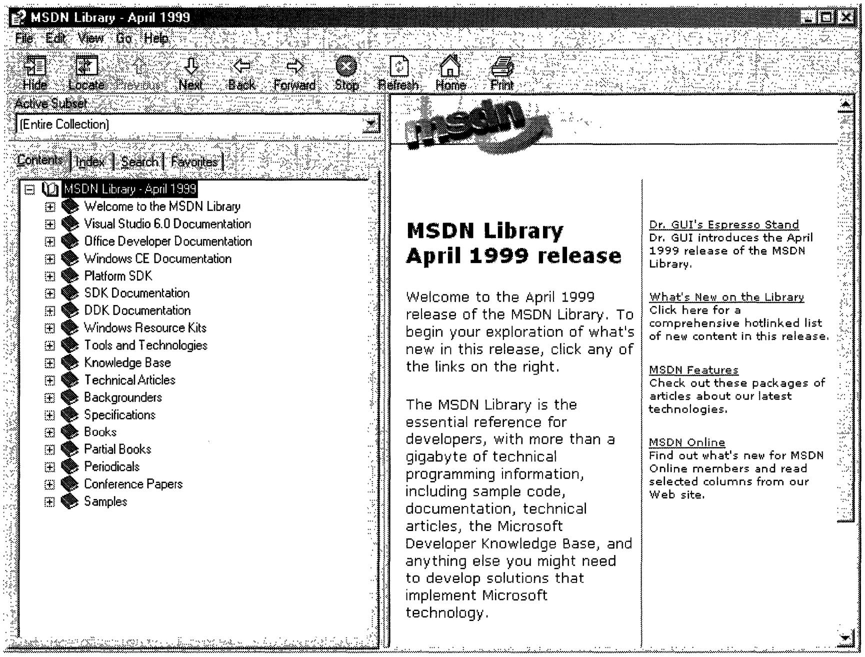
You can get MSDN subscriptions from a number of sources, including online sites specializing in computer-related information, such as *www.iseminger.com* (shameless self-promotion, I know), or from your favorite online software site. Note that not all software resellers carry MSDN subscriptions; you might have to hunt around to find one. Of course, if you have a local software reseller that you frequent, you can check out whether they carry MSDN subscriptions, too.

As an added bonus for owners of this Win32 Library, in the back of Volume 1: *Base Services*, you'll find a $200 rebate good toward an MSDN Universal subscription. For those of you doing the math, that means you actually *make* money when you purchase the Win32 Library and an MSDN Universal subscription. That means every developer in your organization can have the printed Win32 Library on their desk and the MSDN Universal subscription available on their desktop and still come out $50 ahead. That's the kind of math even accountants can like.

Using MSDN

MSDN subscriptions come with an installable interface, and the Professional and Universal subscriptions also come with a bunch of Microsoft product software such as Windows platform versions and BackOffice applications. There's no need to tell you how to use Microsoft product software, but there's a lot to be said for providing some quick but useful guidance on getting the most out of the interface to present and navigate through the seemingly endless supply of reference material provided with any MSDN subscription.

To those who have used MSDN, the interface shown in Figure 3-2 is likely familiar; it's the navigational front-end to MSDN reference material.



MSDN Library April 1999 release

Welcome to the April 1999 release of the MSDN Library. To begin your exploration of what's new in this release, click any of the links on the right.

The MSDN Library is the essential reference for developers, with more than a gigabyte of technical programming information, including sample code, documentation, technical articles, the Microsoft Developer Knowledge Base, and anything else you might need to develop solutions that implement Microsoft technology.

Dr. GUI's Espresso Stand
Dr. GUI introduces the April 1999 release of the MSDN Library.

What's New on the Library
Click here for a comprehensive hotlinked list of new content in this release.

MSDN Features
Check out these packages of articles about our latest technologies.

MSDN Online
Find out what's new for MSDN Online members and read selected columns from our Web site.

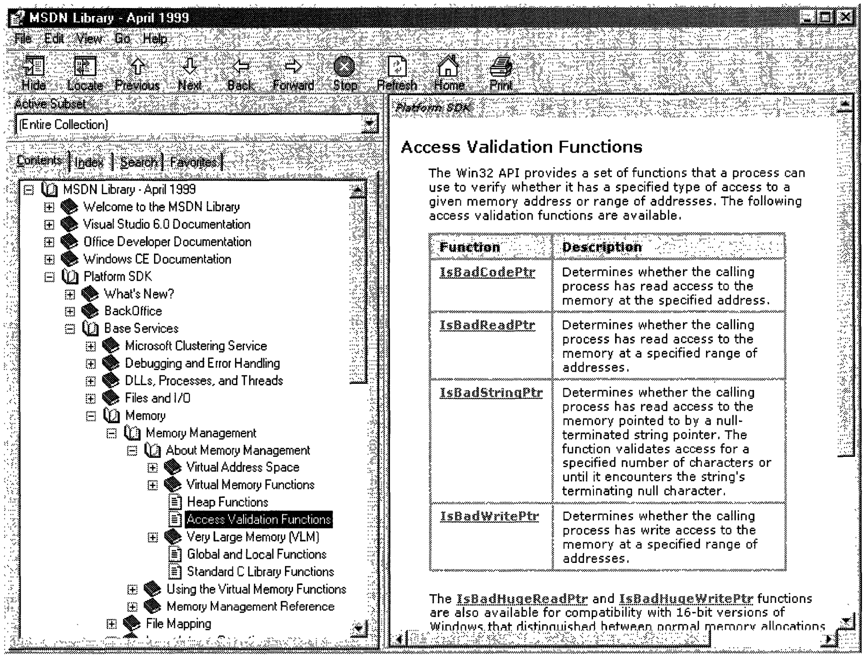
Figure 3-2: The MSDN interface.

The interface is familiar and straightforward enough, but if you don't have a grasp on its features and navigation tools, you can be left a little lost in its sea of information. With a few sentences of explanation and some tips for effective navigation, however, you can increase its effectiveness dramatically.

Navigating MSDN

One of the primary features of MSDN—and to many, its primary drawback—is the sheer volume of information it contains: over 1.1GB and growing. The creators of MSDN likely realized this, though, and have taken steps to assuage the problem. Most of those steps relate to enabling developers to selectively navigate through MSDN's content.

Basic navigation through MSDN is simple, and a lot like navigating through Windows Explorer and its folder structure. Instead of folders, MSDN has books into which it organizes its topics; expand a book by clicking the + box to its left, and its contents are displayed with its nested books or reference pages, as shown in Figure 3-3. If you don't see the left pane in your MSDN viewer, go to the **View** menu and select Navigation Tabs and they'll appear.



Access Validation Functions

The Win32 API provides a set of functions that a process can use to verify whether it has a specified type of access to a given memory address or range of addresses. The following access validation functions are available.

Function	Description
IsBadCodePtr	Determines whether the calling process has read access to the memory at the specified address.
IsBadReadPtr	Determines whether the calling process has read access to the memory at a specified range of addresses.
IsBadStringPtr	Determines whether the calling process has read access to the memory pointed to by a null-terminated string pointer. The function validates access for a specified number of characters or until it encounters the string's terminating null character.
IsBadWritePtr	Determines whether the calling process has write access to the memory at a specified range of addresses.

The IsBadHugeReadPtr and IsBadHugeWritePtr functions are also available for compatibility with 16-bit versions of Windows that distinguished between normal memory allocations

Figure 3-3: Basic navigation through MSDN.

The four tabs in the left pane of MSDN—increasingly referred to as property sheets these days—are the primary means of navigating through MSDN content. These four tabs, in coordination with the **Active Subset** drop-down box above the four tabs, are the tools you use to search through MSDN content. When used to their full extent, these coordinated navigation tools greatly improve your MSDN experience.

The **Active Subset** drop-down box is a filter mechanism; choose the subset of MSDN information you're interested in working with from the drop-down box, and the information in each of the four navigation tabs (including the **Contents** tab) limits the information it displays to the information contained in the selected subset. This means that any searches you do in the **Search** tab, and in the index presented in the **Index** tab, are filtered by their results and/or matches to the subset you define, greatly narrowing the number of potential results for a given inquiry, thereby enabling you to better find the information you're *really* looking for. In the **Index** tab, results that might match your inquiry but *aren't* in the subset you have chosen are grayed out (but still selectable). In the **Search** tab, they simply aren't displayed.

MSDN comes with the following pre-defined subsets:

Entire Collection

MSDN, Books and Periodicals

MSDN, Content on Disk 2 only

MSDN, Content on Disk 3 only

MSDN, Knowledge Base

MSDN, Office Development

MSDN, Technical Articles and Backgrounders

Platform SDK, BackOffice

Platform SDK, Base Services

Platform SDK, Component Services

Platform SDK, Data Access Services

Platform SDK, Graphics and Multimedia Services

Platform SDK, Management Services

Platform SDK, Messaging and Collaboration Services

Platform SDK, Networking Services

Platform SDK, Security

Platform SDK, Tools and Languages

Platform SDK, User Interface Services

Platform SDK, Web Services

Platform SDK, What's New?

Platform SDK, Win32 API

Repository 2.0 Documentation

Visual Basic Documentation

Visual C++ Documentation

Visual C++, Platform SDK and WinCE Docs

Visual C++, Platform SDK, and Enterprise Docs

Visual FoxPro Documentation

Visual InterDev Documentation

Visual J++ Documentation

Visual SourceSafe Documentation

Visual Studio Product Documentation

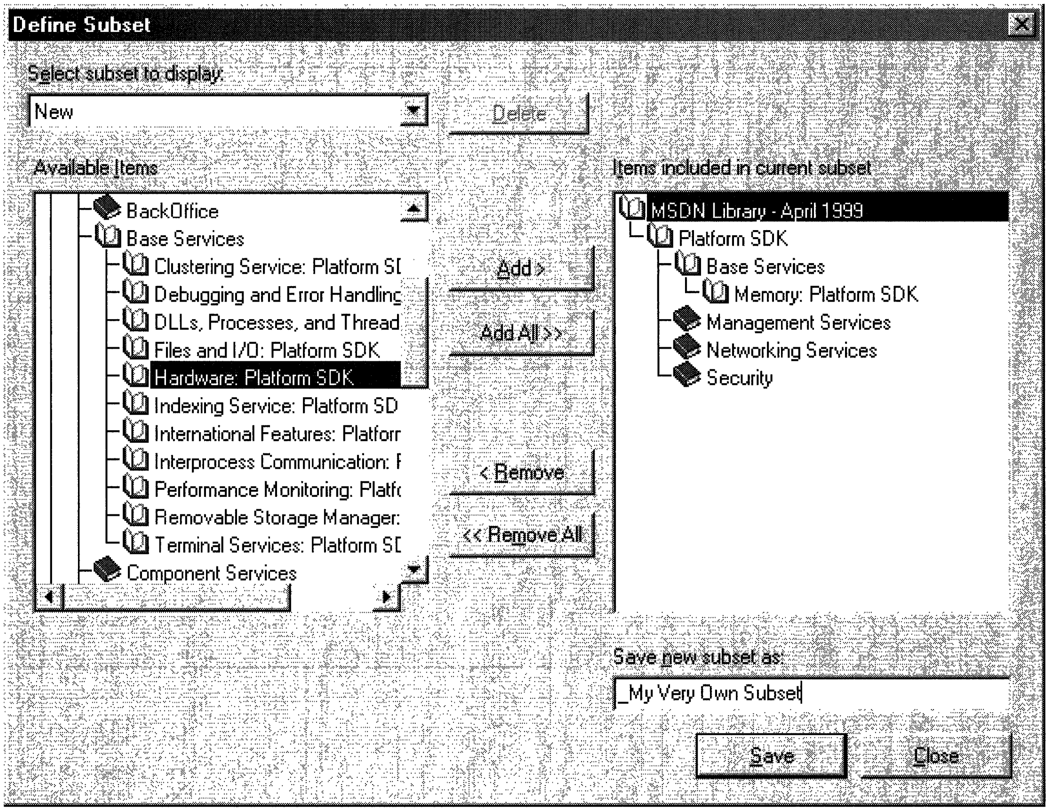
As you can see, this filtering option essentially mirrors the structure of information delivery used by MSDN. But what if you are interested in viewing the information in a handful of these subsets? For example, what if you want to search on a certain keyword through the Platform SDK's Security, Networking Services, and Management Services subsets, as well as a little section that's nested way into the Base Services subset? Simple—you define your own subset.

You define subsets by choosing the **View** menu, and then selecting the **Define Subsets** menu item. You're presented with the window shown in Figure 3-4.

Defining a subset is easy; just take the following steps:

1. Choose the information you want in the new subset; you can choose entire subsets or selected books/content within available subsets.

2. Add your selected information to the subset you're creating by clicking the **Add** button.
3. Name the newly created subset by typing in a name in the **Save New Subset As** dialog box. Note that defined subsets (including any you create) are arranged in alphabetical order.


Define Subset
Delete
Available Items
BackOffice
Base Services
Clustering Service: Platform SI
Debugging and Error Handling
DLLs, Processes, and Thread
Files and I/O: Platform SDK
Hardware: Platform SDK
Indexing Service: Platform SD
International Features: Platforn
Interprocess Communication: I
Performance Monitoring: Platfc
Removable Storage Manager:
Terminal Services: Platform SI
Component Services
Add >
Add All >>
< Remove
<< Remove All
Items included in current subset
MSDN Library - April 1999
Platform SDK
Base Services
Memory: Platform SDK
Management Services
Networking Services
Security
Save
Close

Select subset to display: New

Save new subset as: _My Very Own Subset

Figure 3-4: The Define Subsets window.

You can also delete entire subsets from the MSDN installation, if you so desire. Simply select the subset you want to delete from the **Select Subset To Display** drop-down box, and then click the nearby **Delete** button.

Once you have defined a subset, it becomes available in MSDN just like the pre-defined subsets, and filters the information available in the four navigation tabs just like the pre-defined subsets do.

Quick Tips

Now that you know how to navigate MSDN, there are a handful of tips and tricks that you can use to make MSDN as effective as it can be.

Use the Locate button to get your bearings. Perhaps it's human nature to need to know where you are in the grand scheme of things, but regardless, it can be bothersome to have a reference page displayed in the right pane (perhaps jumped to from a search), without the **Contents** tab in the left pane being synchronized in terms of the reference page's location in the information tree. Even if you know the general technology in which your reference page resides, it's nice to find out where it is in the content structure. This is easy to fix: simply click the **Locate** button in the navigation toolbar, and all will be synchronized.

Use the Back button just like a browser. The **Back** button in the Navigation toolbar functions just like a browser's **Back** button; if you need information on a reference page you viewed previously, you can use the **Back** button to get there, rather than going through the process of doing another search.

Define your own subsets, and use them. Like I said at the beginning of this chapter, the availability of information these days can sometimes make it difficult to get our work done. By defining subsets of MSDN that are tailored to the work you do, you can become more efficient.

Use an underscore at the beginning of your named subsets. Subsets in the **Active Subset** drop-down box are arranged in alphabetical order, and the drop-down box shows only a few subsets at a time (making it difficult to get a grip on available subsets, I think). Underscores come before letters in alphabetical order, so if you use an underscore on all of your defined subsets, you get them placed at the front of the **Active Subset** listing of available subsets. Also, by using an underscore, you can immediately see which subsets you've defined, and which ones come with MSDN—it saves a few seconds at most, but those seconds can add up.

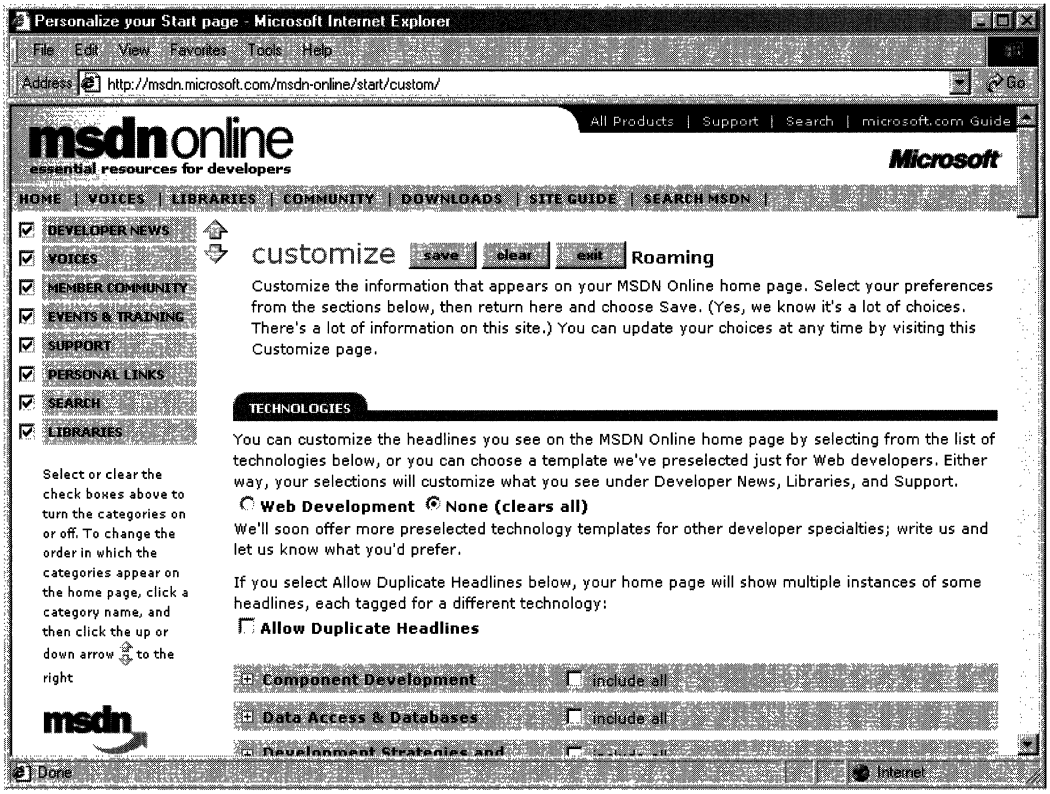
Using MSDN Online

MSDN Online shares a lot of similarities with MSDN, and that probably isn't by accident; when you can go from one developer resource to another and immediately be able to work with its content, your job is made easier. However, MSDN Online is different enough that it merits explaining in its own right...and it should be; it's a different delivery medium, and can take advantage of the Internet in ways that MSDN simply cannot.

If you've used Microsoft's home page before (*www.msn.com* or *home.microsoft.com*), you're familiar with the fact that you can customize the page to your liking; choose from an assortment of available national news, computer news, local news, local weather, stock quotes, and other collections of information or news that suit your tastes or interests. You can even insert a few Web links, and have them readily accessible when you visit the site. The MSDN Online home page can be customized in a similar way, but its collection of headlines, information, and news sources are all about development. The information you choose specifies the information you see when you go to the MSDN Online home page, just like the Microsoft home page.

There are a couple of ways to get to the customization page; you can go to the MSDN Online home page (*msdn.microsoft.com*) and click the **Customize** button at the top of the page, or you can go there directly by pointing your browser to *msdn.microsoft.com/msdn-online/start/custom*. However you get there, the page you'll see is shown in Figure 3-5.

As you can see from Figure 3-5, there are lots of technologies to choose from. If you're interested in Web development, you can choose the **Option** button near the top of the **Technologies** section for Web Development, and a pre-defined subset of Web-centric technologies is selected. For more Win32–centric technologies, you can go through and choose the appropriate technologies. If you want to choose all the technologies in a given technology group, check the **Include All** box in the technology's shaded title area.


Personalize your Start page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://msdn.microsoft.com/msdn-online/start/custom/
Go
All Products | Support | Search | microsoft.com Guide
msdnonline
essential resources for developers
Microsoft
HOME | VOICES | LIBRARIES | COMMUNITY | DOWNLOADS | SITE GUIDE | SEARCH MSDN |
customize
save
clear
exit
Roaming
TECHNOLOGIES
msdn
Done
Internet

☑ DEVELOPER NEWS
☑ VOICES
☑ MEMBER COMMUNITY
☑ EVENTS & TRAINING
☑ SUPPORT
☑ PERSONAL LINKS
☑ SEARCH
☑ LIBRARIES

Select or clear the check boxes above to turn the categories on or off. To change the order in which the categories appear on the home page, click a category name, and then click the up or down arrow to the right

Customize the information that appears on your MSDN Online home page. Select your preferences from the sections below, then return here and choose Save. (Yes, we know it's a lot of choices. There's a lot of information on this site.) You can update your choices at any time by visiting this Customize page.

You can customize the headlines you see on the MSDN Online home page by selecting from the list of technologies below, or you can choose a template we've preselected just for Web developers. Either way, your selections will customize what you see under Developer News, Libraries, and Support.

○ Web Development ● None (clears all)

We'll soon offer more preselected technology templates for other developer specialties; write us and let us know what you'd prefer.

If you select Allow Duplicate Headlines below, your home page will show multiple instances of some headlines, each tagged for a different technology:

□ Allow Duplicate Headlines

⊞ Component Development □ include all

⊞ Data Access & Databases □ include all

⊞ Development Strategies and □ include all

Figure 3-5: The MSDN Online configuration page.

You can also choose which categories are included in the information MSDN Online presents to you, as well as their arranged order. The available categories include:

Developer News
Categories
Libraries
Search

Member Community
Events & Training
Support
Personal Links

Once you've defined your profile—that is, customized the MSDN Online content you want to see—MSDN Online shows you the most recent information pertinent to your profile when you go to MSDN Online's home page, with the categories you've chosen included in the order you specify. Note that clearing a given category—such as Libraries—clears that category from the body of your MSDN Online home page (and excludes headlines for that category), but does not remove that category from the MSDN Online site navigation bar. In other words, if you clear the category it won't be part of your customized MSDN Online page's headlines, but it'll still be available as a site feature.

Finally, if you want your profile to be available to you regardless of which computer you're using, you can direct MSDN Online to create a *roaming profile*. Creating a roaming profile for MSDN Online results in your profile being stored on MSDN Online's server, much like roaming profiles in Windows 2000, and thereby makes your profile available to you regardless of the computer you're using. The option of creating a roaming profile is available when you customize your MSDN Online home page (and can be done any time thereafter). The creation of a roaming profile, however, requires that you become a registered member of MSDN Online. More information about becoming a registered MSDN Online user is provided in the section titled *MSDN Online Registered Users*.

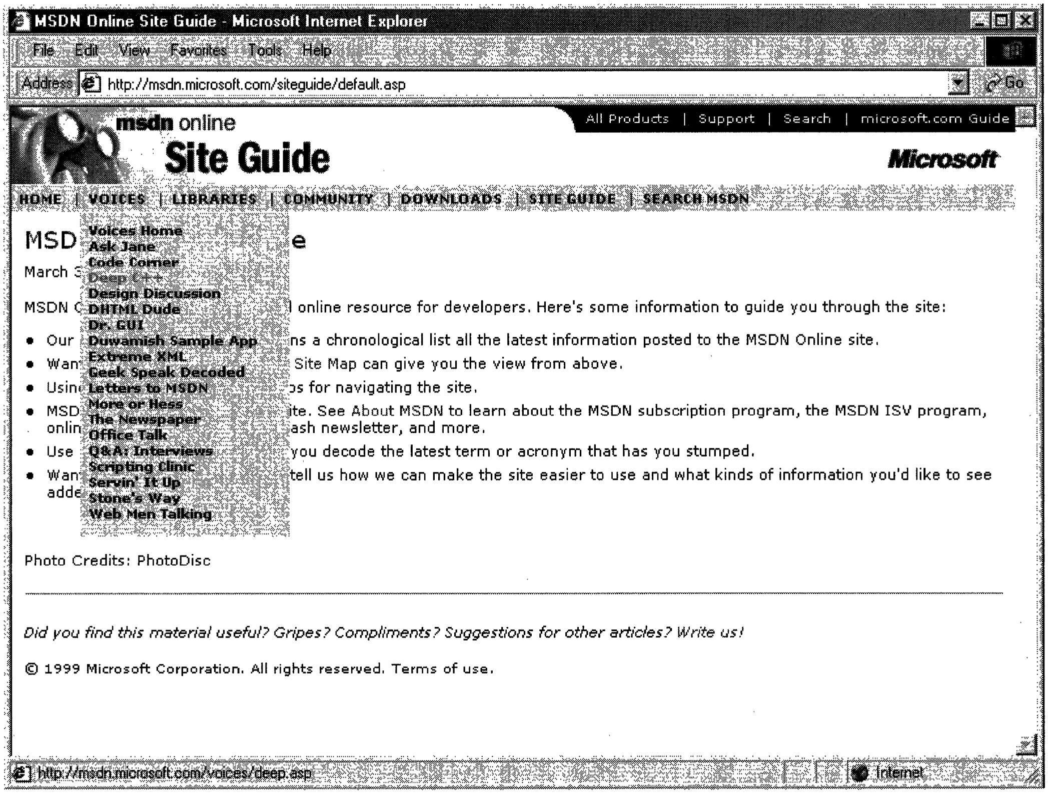
Navigating MSDN Online

Once you're done customizing the MSDN Online home page to get the headlines you're most interested in seeing, navigating through MSDN Online is really easy. A banner that sits just below the MSDN Online logo functions as a navigation bar with drop-down menus that can take you to the available areas on MSDN Online, as Figure 3-6 illustrates.

The list of available menu categories—which group the available sites and features within MSDN Online—includes:

Home
Voices
Libraries
Community
Downloads
Site Guide
Search MSDN

The **Navigation** bar is available regardless of where you are in MSDN Online, so the capability to navigate the site from this familiar menu is always available, leaving you a click away from any area on MSDN Online. These menu categories create a functional and logical grouping of MSDN Online's feature offerings.


MSDN Online Site Guide - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://msdn.microsoft.com/siteguide/default.asp
Go
msdn online
Site Guide
All Products | Support | Search | microsoft.com Guide
Microsoft
HOME | VOICES | LIBRARIES | COMMUNITY | DOWNLOADS | SITE GUIDE | SEARCH MSDN
Voices Home
Ask Jane
Code Corner
Deep C++
Design Discussion
DHTML Dude
Dr. GUI
Duwamish Sample App
Extreme XML
Geek Speak Decoded
Letters to MSDN
More or Hess
The Newspaper
Office Talk
Q&A: Interviews
Scripting Clinic
Servin' It Up
Stone's Way
Web Men Talking
http://msdn.microsoft.com/voices/deep.asp
Internet

MSD e

March 3

MSDN C l online resource for developers. Here's some information to guide you through the site:

- Our ns a chronological list all the latest information posted to the MSDN Online site.
- Wan Site Map can give you the view from above.
- Usin ps for navigating the site.
- MSD ite. See About MSDN to learn about the MSDN subscription program, the MSDN ISV program, onlin ash newsletter, and more.
- Use you decode the latest term or acronym that has you stumped.
- Wan tell us how we can make the site easier to use and what kinds of information you'd like to see adde

Photo Credits: PhotoDisc

Did you find this material useful? Gripes? Compliments? Suggestions for other articles? Write us!

© 1999 Microsoft Corporation. All rights reserved. Terms of use.

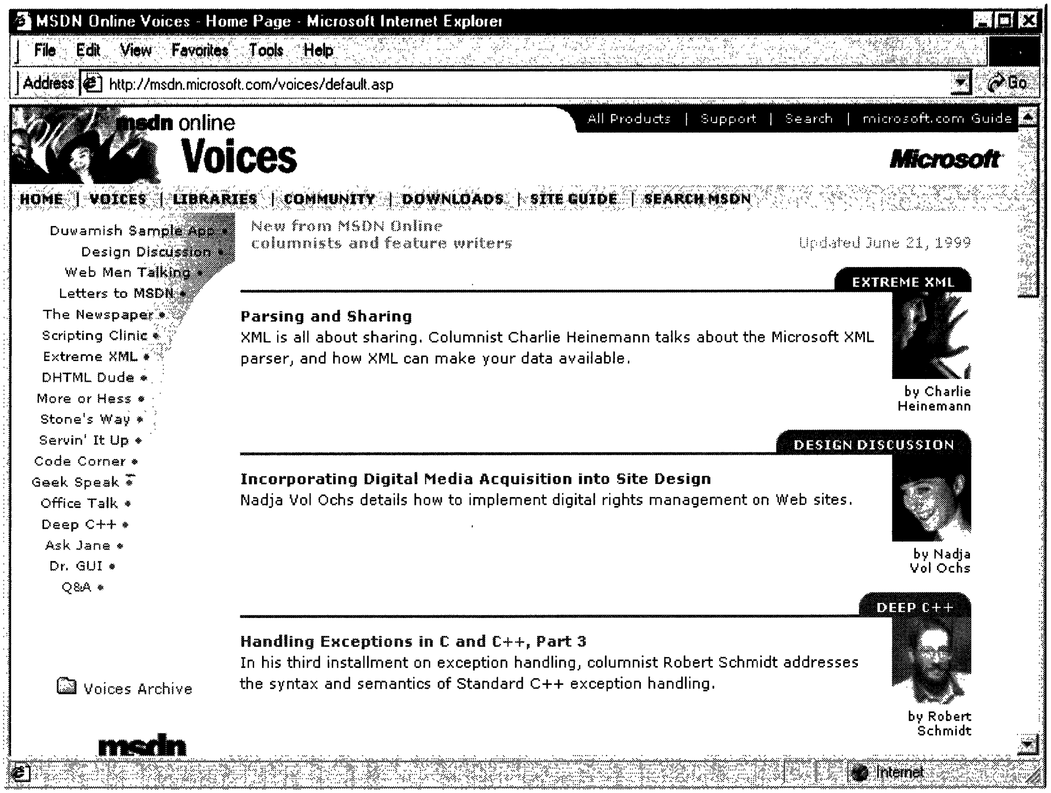
Figure 3-6: The MSDN Online Navigation bar with its drop-down menus.

MSDN Online Features

Each of MSDN Online's seven feature categories contains various sites that comprise the features available to developers visiting MSDN Online.

Home is already familiar; clicking on **Home** in the navigation bar takes you to the MSDN Online home page that you've (perhaps) customized, showing you all the latest headlines for technologies that you've indicated you're interested in reading about.

Voices is a collection of columns and articles that comprise MSDN Online's magazine section, and can be linked to directly at *msdn.microsoft.com/voices*. The Voices home page is shown in Figure 3-7.



New from MSDN Online
columnists and feature writers

Updated June 21, 1999

Parsing and Sharing
XML is all about sharing. Columnist Charlie Heinemann talks about the Microsoft XML parser, and how XML can make your data available.

by Charlie Heinemann

Incorporating Digital Media Acquisition into Site Design
Nadja Vol Ochs details how to implement digital rights management on Web sites.

by Nadja Vol Ochs

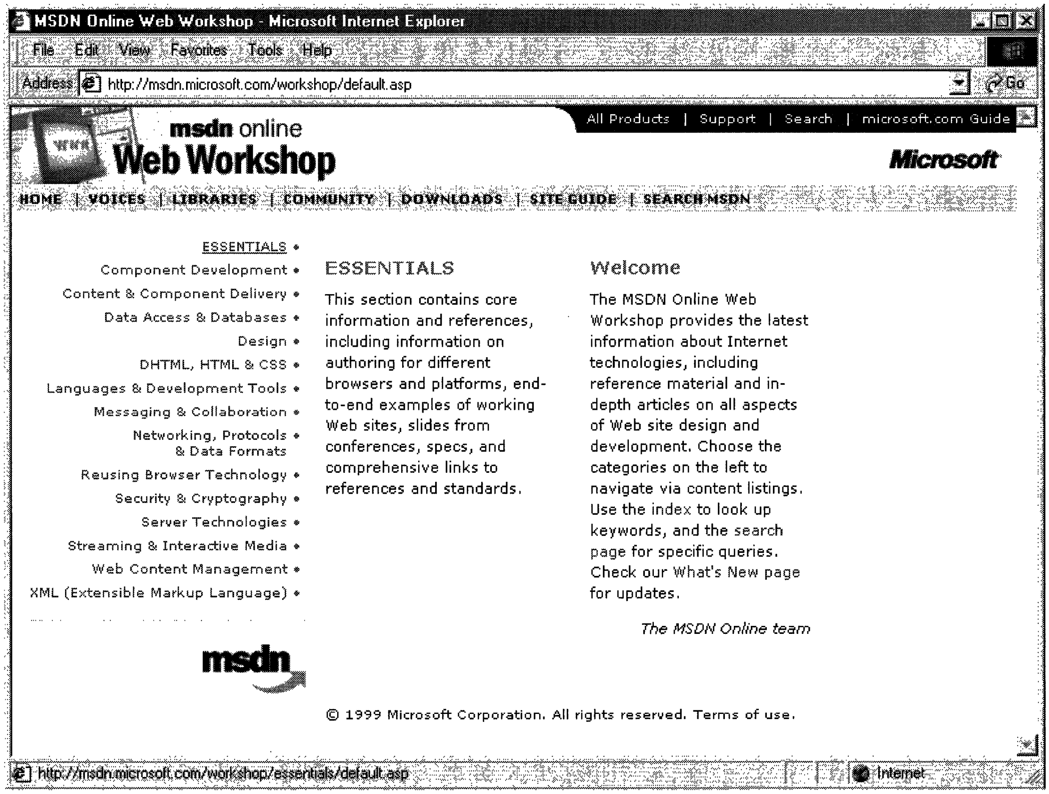
Handling Exceptions in C and C++, Part 3
In his third installment on exception handling, columnist Robert Schmidt addresses the syntax and semantics of Standard C++ exception handling.

by Robert Schmidt

Figure 3-7: The Voices home page.

There are a bunch of different "voices" in the Voices site, each of which adds its own particular twist on the issues that face developers. Both application and Web developers can get their fill of magazine-like articles from the sizable list of different articles available (and frequently refreshed) in the Voices site.

Libraries is where the reference material available on MSDN Online lives. The Libraries site is divided into two sections: Library and Web Workshop. This distinction divides the reference material between what used to be MSDN and Site Builder Network; that is, Windows application development and Web development. Choosing **Library** from the **Libraries** menu takes you to a page you can navigate in traditional MSDN fashion, and gain access to traditional MSDN reference material; the Library home page can be linked to directly at *msdn.microsoft.com/library*. Choosing **Web Workshop** takes you to a site that enables you to navigate the Web Workshop in a slightly different way, starting with a bulleted list of start points, as shown in Figure 3-8. The Web Workshop home page can be linked to directly at *msdn.microsoft.com/workshop*.


MSDN Online Web Workshop - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://msdn.microsoft.com/workshop/default.asp
Go
msdn online
Web Workshop
All Products | Support | Search | microsoft.com Guide
Microsoft
HOME | VOICES | LIBRARIES | COMMUNITY | DOWNLOADS | SITE GUIDE | SEARCH MSDN
msdn
http://msdn.microsoft.com/workshop/essentials/default.asp
Internet

ESSENTIALS •
Component Development •
Content & Component Delivery •
Data Access & Databases •
Design •
DHTML, HTML & CSS •
Languages & Development Tools •
Messaging & Collaboration •
Networking, Protocols & Data Formats •
Reusing Browser Technology •
Security & Cryptography •
Server Technologies •
Streaming & Interactive Media •
Web Content Management •
XML (Extensible Markup Language) •

ESSENTIALS

This section contains core information and references, including information on authoring for different browsers and platforms, end-to-end examples of working Web sites, slides from conferences, specs, and comprehensive links to references and standards.

Welcome

The MSDN Online Web Workshop provides the latest information about Internet technologies, including reference material and in-depth articles on all aspects of Web site design and development. Choose the categories on the left to navigate via content listings. Use the index to look up keywords, and the search page for specific queries. Check our What's New page for updates.

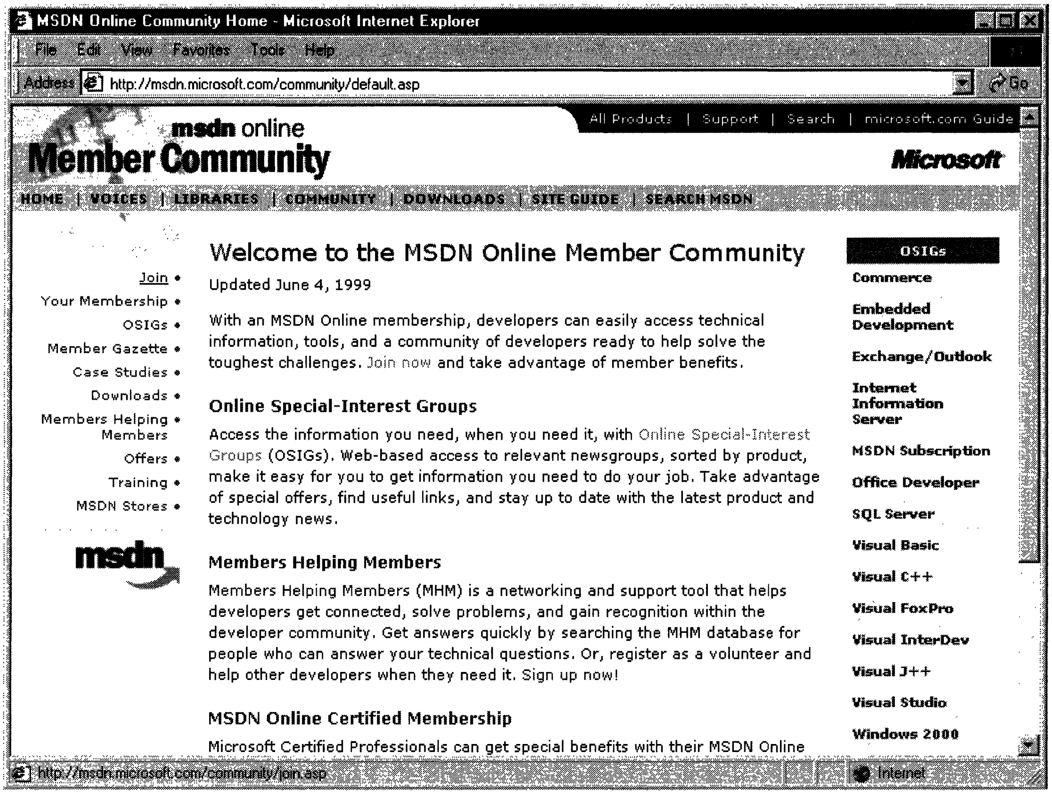
The MSDN Online team

© 1999 Microsoft Corporation. All rights reserved. Terms of use.

Figure 3-8: The Web Workshop home page, with its bulleted list of navigation start points.

Community is a place where developers can go to take advantage of the online forum of Windows and Web developers, in which ideas or techniques can be shared, advice can be found or given (through MHM, or Members Helping Members), and Online Special Interest Groups (OSIGs) can find a forum to voice their opinions or chat with other developers. The Community site is full of all sorts of useful stuff, including featured books, promotions and downloads, case studies, and more. The Community home page can be linked to directly at *msdn.microsoft.com/community*. Figure 3-9 provides a look at the Community home page.

The **Site Guide** is just what its name suggests; a guide to the MSDN Online site that aims at helping developers find items of interest, and includes links to other pages on MSDN Online such as a recently posted files listing, site maps, glossaries, and other useful links. The Site Guide home page can be linked to directly at *msdn.microsoft.com/siteguide*.


MSDN Online Community Home - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://msdn.microsoft.com/community/default.asp
Go
All Products | Support | Search | microsoft.com Guide
msdn online
Member Community
Microsoft
HOME | VOICES | LIBRARIES | COMMUNITY | DOWNLOADS | SITE GUIDE | SEARCH MSDN
Join •
Your Membership •
OSIGs •
Member Gazette •
Case Studies •
Downloads •
Members Helping Members •
Offers •
Training •
MSDN Stores •
msdn
OSIGs
Commerce
Embedded Development
Exchange/Outlook
Internet Information Server
MSDN Subscription
Office Developer
SQL Server
Visual Basic
Visual C++
Visual FoxPro
Visual InterDev
Visual J++
Visual Studio
Windows 2000
http://msdn.microsoft.com/community/join.asp
Internet

Welcome to the MSDN Online Member Community

Updated June 4, 1999

With an MSDN Online membership, developers can easily access technical information, tools, and a community of developers ready to help solve the toughest challenges. Join now and take advantage of member benefits.

Online Special-Interest Groups

Access the information you need, when you need it, with Online Special-Interest Groups (OSIGs). Web-based access to relevant newsgroups, sorted by product, make it easy for you to get information you need to do your job. Take advantage of special offers, find useful links, and stay up to date with the latest product and technology news.

Members Helping Members

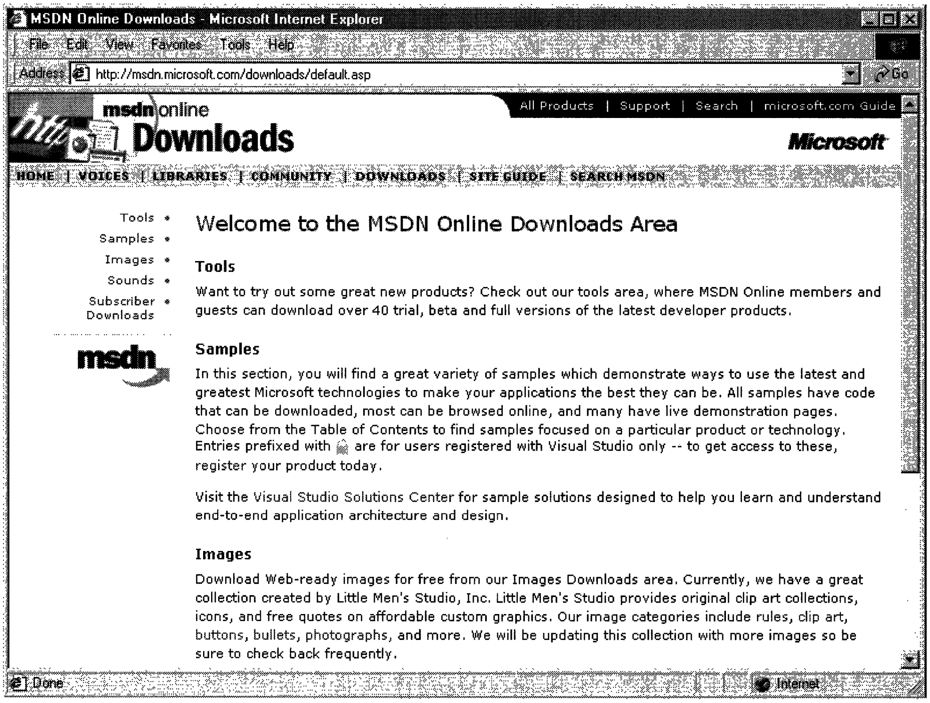
Members Helping Members (MHM) is a networking and support tool that helps developers get connected, solve problems, and gain recognition within the developer community. Get answers quickly by searching the MHM database for people who can answer your technical questions. Or, register as a volunteer and help other developers when they need it. Sign up now!

MSDN Online Certified Membership

Microsoft Certified Professionals can get special benefits with their MSDN Online

Figure 3-9: The Community home page.

The **Downloads** site is where developers can find all sorts of items that can be downloaded, such as tools, samples, images, and sounds. The Downloads site is also where MSDN subscribers go to get their subscription content updated over the Internet to the latest and greatest releases, as described previously in this chapter in the *Using MSDN* section. The Downloads home page can be linked to directly at *msdn.microsoft.com/downloads*. The Downloads home page is shown in Figure 3-10.


MSDN Online Downloads - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://msdn.microsoft.com/downloads/default.asp
Go
msdn online
All Products | Support | Search | microsoft.com Guide
Downloads
Microsoft
HOME | VOICES | LIBRARIES | COMMUNITY | DOWNLOADS | SITE GUIDE | SEARCH MSDN
Tools
Samples
Images
Sounds
Subscriber Downloads
msdn
Done
Internet

Welcome to the MSDN Online Downloads Area

Tools

Want to try out some great new products? Check out our tools area, where MSDN Online members and guests can download over 40 trial, beta and full versions of the latest developer products.

Samples

In this section, you will find a great variety of samples which demonstrate ways to use the latest and greatest Microsoft technologies to make your applications the best they can be. All samples have code that can be downloaded, most can be browsed online, and many have live demonstration pages. Choose from the Table of Contents to find samples focused on a particular product or technology. Entries prefixed with 🔒 are for users registered with Visual Studio only -- to get access to these, register your product today.

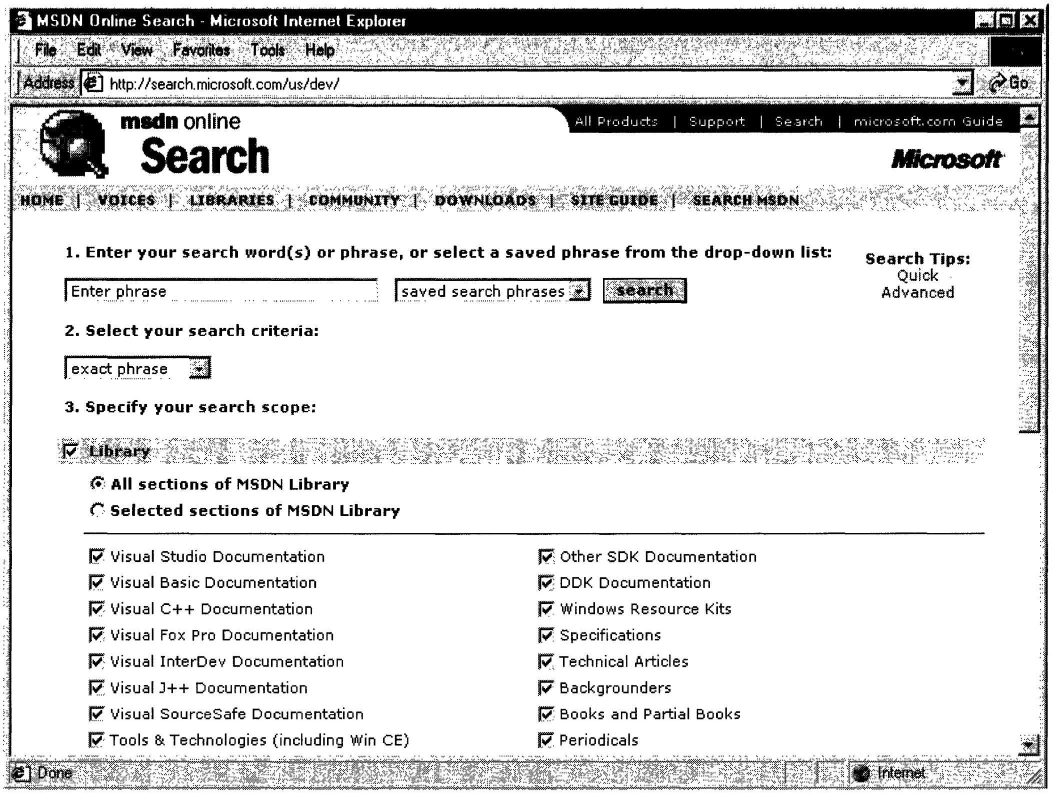
Visit the Visual Studio Solutions Center for sample solutions designed to help you learn and understand end-to-end application architecture and design.

Images

Download Web-ready images for free from our Images Downloads area. Currently, we have a great collection created by Little Men's Studio, Inc. Little Men's Studio provides original clip art collections, icons, and free quotes on affordable custom graphics. Our image categories include rules, clip art, buttons, bullets, photographs, and more. We will be updating this collection with more images so be sure to check back frequently.

Figure 3-10: The Downloads home page.

The **Search MSDN** site on MSDN Online has been improved over previous versions, and includes the capability to restrict searches to either library (Library or Web Workshop), in addition to other search capabilities. The Search MSDN home page can be linked to directly at *msdn.microsoft.com/search*. The Search MSDN home page is shown in Figure 3-11.


MSDN Online Search - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
Go
msdn online
Search
All Products | Support | Search | microsoft.com Guide
Microsoft
HOME | VOICES | LIBRARIES | COMMUNITY | DOWNLOADS | SITE GUIDE | SEARCH MSDN
search
Search Tips:
Quick
Advanced
Done
Internet

http://search.microsoft.com/us/dev/

1. Enter your search word(s) or phrase, or select a saved phrase from the drop-down list:

Enter phrase | saved search phrases

2. Select your search criteria:

exact phrase

3. Specify your search scope:

☑ Library

● All sections of MSDN Library

○ Selected sections of MSDN Library

☑ Visual Studio Documentation
☑ Visual Basic Documentation
☑ Visual C++ Documentation
☑ Visual Fox Pro Documentation
☑ Visual InterDev Documentation
☑ Visual J++ Documentation
☑ Visual SourceSafe Documentation
☑ Tools & Technologies (including Win CE)

☑ Other SDK Documentation
☑ DDK Documentation
☑ Windows Resource Kits
☑ Specifications
☑ Technical Articles
☑ Backgrounders
☑ Books and Partial Books
☑ Periodicals

Figure 3-11: The Search home page.

MSDN Online Registered Users

You may have noticed that some features of MSDN Online—such as the capability to create a roaming profile of the entry ticket to some community features—require you to become a registered user. Unlike MSDN subscriptions, becoming a registered user of MSDN Online won't cost you anything more than a few minutes of registration time.

Some features of MSDN Online require registration before you can take advantage of their offerings. For example, becoming a member of an Online Special Interest Group (OSIG) requires registration. That feature alone is incentive enough to register; rather than attempting to call your developer buddy for an answer to a question (only to find out that she's on vacation for two days, and your deadline is in a few hours), you can go to MSDN Online's Community site and ferret through your OSIG to find the answer in a handful of clicks. Who knows; maybe your developer buddy will begin calling you with questions—you don't have to tell her where you're getting all your answers.

There are actually a number of advantages to being a registered user, such as the choice to receive newsletters right in your inbox—if you want to. You can also get all sorts of other timely information, such as chat reminders that let you know when experts on a given subject will be chatting in the MSDN Online Community site. You can also sign up to get newsletters based on your membership in various OSIGs—again, only if you want to. It's easy for me to suggest that you become a registered user for MSDN Online—I'm a registered user, and it's a great resource.

The Windows Programming Reference Series

The Windows Programming Reference Series provides developers with timely, concise, and focused material on a given topic, enabling developers to get their work done as efficiently as possible. In addition to providing reference material for Microsoft technologies, each Library in the Windows Programming Reference Series also includes material that helps developers get the most out of its technologies, and provides insights that might otherwise be difficult to find.

The Windows Programming Reference Series is currently planned to include the following libraries:

Win32 Library

Active Directory Services Library

Networking Services Library

In the near future (subject, of course, to technology release schedules, demand, and other forces that can impact publication decisions), you can look for these prospective Windows Programming Reference Series Libraries that cover the following material:

Web Technologies Library

Web Reference Library

COM/DCOM 2.0 Library

What else might you find in the future? Topics such as a Security, Languages and MFC, BackOffice, and other pertinent topics that developers using Microsoft products need in order to get the most out of their development efforts, are prime subjects for future libraries in the Windows Programming Reference Series. If you have feedback you want to provide on such libraries, or on the Windows Programming Reference Series in general, you can send mail to the following address: *winprs@microsoft.com.*

If you're sending mail about a particular Library, make sure you put the name of the library in the subject line. For example, an e-mail about the Win32 Library would have a subject line that reads "Win32 Library." There aren't any guarantees that you'll get a reply, but I'll read all of the mail and do what I can to ensure your comments, concerns, or (especially) compliments get to the right place.

CHAPTER 4

Finding the Developer Resources You Need

There are all sorts of resources out there for developers of Windows applications, and they can provide answers to a multitude of questions or problems that developers face every day, but finding those resources is sometimes harder than the original problem. This chapter aims to provide you with a one-stop resource to find as many developer resources as are available, again making your job of actually developing the application just a little easier.

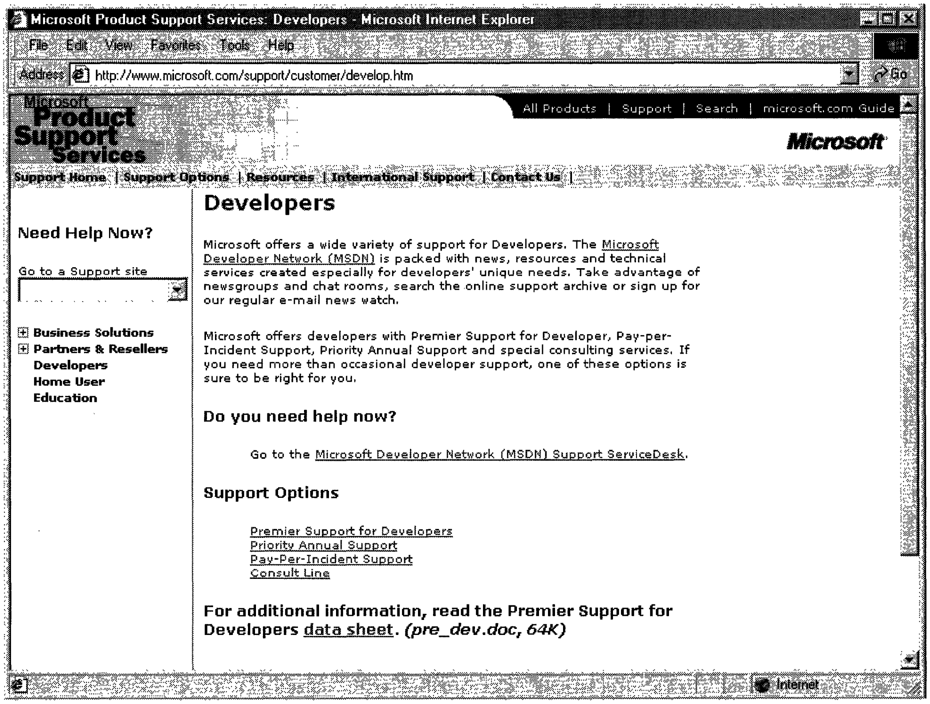
While Microsoft provides lots of resource material through MSDN and MSDN Online, and although the Windows Programming Resource Series provides lots of focused reference material and development tips and tricks, there is a *lot* more information to be had. Some of it is from Microsoft, some from the general development community, and some from companies that specialize in such development services. Regardless of which resource you choose, in this chapter you can find out what your development resource options are and, therefore, be more informed about the resources that are available to you.

Microsoft provides developer resources through a number of different media, channels, and approaches. The extensiveness of Microsoft's resource offerings mirrors the fact that many are appropriate under various circumstances. For example, you wouldn't go to a conference to find the answer to a specific development problem in your programming project; instead, you might use one of the other Microsoft resources.

Developer Support

Microsoft's support sites cover a wide variety of support issues and approaches, including all of Microsoft's products, but most of those sites are not pertinent to developers. Some sites, however, *are* designed for developer support; the Product Services Support page for developers is a good central place to find the support information you need. Figure 4-1 shows the Product Services Support page for developers, which can be found at *www.microsoft.com/support/customer/develop.htm.*

Note that there are a number of options for support from Microsoft, including everything from simple online searches of known bugs in the Knowledge Base to hands-on consulting support from Microsoft Consulting Services, and everything in between. The Web page displayed in Figure 4-1 is a good starting point from which you can find out more information about Microsoft's support services.


Microsoft Product Support Services: Developers - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://www.microsoft.com/support/customer/develop.htm
Go
Microsoft Product Support Services
All Products | Support | Search | microsoft.com Guide
Microsoft
Support Home | Support Options | Resources | International Support | Contact Us |
Need Help Now?
Go to a Support site
Business Solutions
Partners & Resellers
Developers
Home User
Education
Internet

Developers

Microsoft offers a wide variety of support for Developers. The Microsoft Developer Network (MSDN) is packed with news, resources and technical services created especially for developers' unique needs. Take advantage of newsgroups and chat rooms, search the online support archive or sign up for our regular e-mail news watch.

Microsoft offers developers with Premier Support for Developer, Pay-per-Incident Support, Priority Annual Support and special consulting services. If you need more than occasional developer support, one of these options is sure to be right for you.

Do you need help now?

Go to the Microsoft Developer Network (MSDN) Support ServiceDesk.

Support Options

Premier Support for Developers
Priority Annual Support
Pay-Per-Incident Support
Consult Line

For additional information, read the Premier Support for Developers data sheet. *(pre_dev.doc, 64K)*

Figure 4-1: The Product Services Support page for developers.

Premier Support from Microsoft provides extensive support for developers, and there are different packages geared toward different Microsoft customers. The packages of Premier Support that Microsoft provides are:

- Premier Support for Enterprises
- Premier Support for Developers
- Premier Support for Microsoft Certified Solution Providers
- Premier Support for OEMs

If you're a developer, you might fall into any of these categories. To find out more information about Microsoft's Premier Support, get in contact with them at 1-800-936-2000.

Priority Annual Support from Microsoft is geared toward developers or organizations that have more than an occasional need to call Microsoft with support questions, and need priority handling of their support questions or issues. There are three packages of Priority Annual Support offered by Microsoft:

- Priority Comprehensive Support
- Priority Developer Support
- Priority Desktop Support

As a developer, the best support option for you is the Priority Developer Support. To get more information about Priority Developer Support, you can reach Microsoft at 1-800-936-3500.

Microsoft also offers a **Pay-Per-Incident** support option, so you can get help if there's just one question for which you must have an answer. With Pay-Per-Incident support, you call a toll-free number and provide your Visa, MasterCard, or American Express card number, after which you receive support for your incident. In loose terms, an incident is some problem or issue that can't be broken down into sub-issues or sub-problems (that is, it can't be broken down into smaller pieces). The number to call for Pay-Per-Incident support is 1-800-936-5800.

Note that Microsoft provides two priority technical support incidents as part of the MSDN Professional Subscription, and provides four priority technical support incidents as part of the MSDN Universal Subscription.

You can also **submit questions** to Microsoft engineers through Microsoft's support Web site, but if you're on a deadline you might want to rethink this approach, or consider going to MSDN Online and looking into the Community site there for help with your development question. To submit a question to Microsoft engineers online, go to *support.microsoft.com/support/webresponse.asp*.

Online Resources

Microsoft also provides extensive developer support through its community of developers found on MSDN Online. At MSDN Online's Community site, you will find OSIGs that cover all sorts of issues in an online, ongoing fashion. To get to MSDN Online's Community site, go to *msdn.microsoft.com/community*.

Microsoft's MSDN Online also provides its **Knowledge Base** online, which is part of the Personal Support Center on Microsoft's corporate site. You can search the Knowledge Base online at *support.microsoft.com/support/search*.

Microsoft provides a number of **newsgroups** that developers can use to view information on newsgroup-specific topics, providing yet another developer resource for finding information about creating Windows applications. To find out which newsgroups are available, and how to get to them, go to *support.microsoft.com/support/news*.

There is a handful of newsgroups that will probably be of particular interest to readers of the *Microsoft Win32 Developer's Reference Library*, and they are the following:

*microsoft.public.win32.programmer.**

*microsoft.public.vc.**

*microsoft.public.vb.**

*microsoft.public.platformsdk.**

*microsoft.public.cert.**

*microsoft.public.certification.**

Of course, Microsoft isn't the only newsgroup provider on which newsgroups pertaining to Windows development are hosted. Usenet has all sorts of newsgroups—too many to list—that host ongoing discussions pertaining to developing applications on the Windows platform. You can access newsgroups on Windows development just as you access any other newsgroup; generally, you'll need to contact your ISP to find out the name of the mail server, and then use a newsreader application to visit, read, or post to the Usenet groups.

Learning Products

Microsoft provides a number of products that help enable developers to learn the particular tasks or tools that they need to achieve their goals (or to finish their tasks). One product line that is geared toward developers is called the **Mastering Series**, and its products provide comprehensive, well-structured, interactive teaching tools for a wide variety of development topics.

The Mastering Series from Microsoft consists of interactive tools that group books and CDs together so that you can master the topic in question. To get more information about the Mastering Series of products, or to find out what kind of offerings the Mastering Series has, check out *msdn.microsoft.com/mastering*.

Other learning products are available from other vendors, too, such as other publishers, other applications providers that create tutorial-type content and applications, and companies that issue videos (both taped and broadcast over the Internet) on specific technologies. For one example of a company that issues technology-based instructional or overview videos, take a look at *www.compchannel.com*.

Another way of learning about development in a particular language (such as Visual C++, Visual FoxPro, or Visual Basic), for a particular operating system, or for a particular product (such as SQL Server or Commerce Server) is to go through and read the preparation materials available to get certified as a Microsoft Certified Solution Developer (MCSD). Before you get too defensive about not having enough time to get certified, or in having no interest in getting your certification (maybe you do—there *are* benefits, you know), let me state that the point of the journey is not necessarily to arrive. In other words, you don't have to get your certification for the preparation materials to be useful; in fact, they might teach you things that you thought you knew well, but actually didn't know as well as you thought you did. The fact of the matter is that the coursework and the requirements to get through the certification process are rigorous, difficult, and quite detail-oriented. If you have what it takes to get your certification, you have an extremely strong grasp on the fundamentals (and then some) of application programming and the developer-oriented information about Windows platforms.

You are required to take a set of core exams to get an MCSD certification, and then you must choose one topic from many available elective exams to complete your certification requirements. Core exams are chosen from among a group of available exams; you must pass a total of three exams to complete the core requirements. There are "tracks" that candidates generally choose and that point their certification in a given direction, such as Visual C++ development or Visual Basic development. The core exams and their exam numbers are as follows.

Desktop Applications Development (one required):

- Designing and Implementing Desktop Applications with Microsoft Visual C++ 6.0 (70-016)
- Designing and Implementing Desktop Applications with Microsoft Visual FoxPro 6.0 (70-155)
- Designing and Implementing Desktop Applications with Microsoft Visual Basic 6.0 (70-176)

Distributed Applications Development (one required):

- Designing and Implementing Distributed Applications with Microsoft Visual C++ 6.0 (70-015)
- Designing and Implementing Distributed Applications with Microsoft Visual FoxPro 6.0 (70-156)
- Designing and Implementing Distributed Applications with Microsoft Visual Basic 6.0 (70-175)

Solutions Architecture:

- Analyzing Requirements and Defining Solution Architectures (70-100)

Elective exams enable candidates to choose from a number of additional exams to complete their MCSD exam requirements. The following lists the available MCSD elective exams.

Available elective exams:

- Any Desktop or Distributed exam not used as a core requirement
- Designing and Implementing Data Warehouses with Microsoft SQL Server 7.0 and Microsoft Decision Support Services 1.0
- Developing Applications with C++ Using the Microsoft Foundation Class Library 4.0 Library
- Implementing OLE in Microsoft Foundation Class Library 4.0 Applications
- Implementing a Database Design on Microsoft SQL Server 6.5
- Designing and Implementing Databases with Microsoft SQL Server 7.0
- Designing and Implementing Web Sites with Microsoft FrontPage 98
- Designing and Implementing Commerce Solutions with Microsoft Site Server 3.0, Commerce Edition
- Microsoft Access for Windows 95 and the Microsoft Access Developer's Toolkit
- Designing and Implementing Solutions with Microsoft Office 2000 and Microsoft Visual Basic for Applications

- Designing and Implementing Database Applications with Microsoft Access 2000
- Designing and Implementing Collaborative Solutions with Microsoft Outlook 2000 and Microsoft Exchange Server 5.5
- Designing and Implementing Web Solutions with Microsoft Visual InterDev 6.0
- Designing and Implementing Distributed Applications with Microsoft Visual FoxPro 6.0
- Designing and Implementing Desktop Applications with Microsoft Visual FoxPro 6.0
- Developing Applications with Microsoft Visual Basic 5.0
- Designing and Implementing Distributed Applications with Microsoft Visual Basic 6.0
- Designing and Implementing Desktop Applications with Microsoft Visual Basic 6.0

The best news about these exams isn't that there are lots from which to choose. The best news is that, because there are exams that must be passed to become certified, there are books and other materials out there to *teach you* how to meet the knowledge level necessary to pass the exams, and that means those resources are available to you—regardless of whether you care one whit about becoming an MCSD or not.

The way to leverage this information is to get study materials for one or more of these exams—and don't be fooled by believing that if the book is bigger it must be better, because that certainly isn't always the case—and go through the exam preparation material. Such exam preparation material is available from all sorts of publishers, including Microsoft Press, IDG, Sybex, and others. Most exam preparation texts also have practice exams that let you self-assess your grasp of the material. You might be surprised by how much you learn, even though you might have been in the field working on complex projects for some time.

Of course, these exam requirements, and the exams themselves, can change over time; more electives become available, exams based on revised versions of software are retired, and so on. For more information about the certification process, or for more information about the exams, check out *www.microsoft.com/train_cert/dev.*

Conferences

As in any industry, Microsoft and the development community as a whole sponsor conferences throughout the year—occurring throughout the country and around the world—on various topics. There are probably more conferences available than any human being could possibly attend and still be sane, but often a given conference is geared toward a particular topic, so choosing to focus on a given development topic enables developers to select the number of conferences that apply to their efforts and interests.

MSDN itself hosts or sponsors almost a hundred conferences a year (some of them are regional and duplicated in different locations, so these could be considered one conference that happens multiple times). Other conferences are held in one central location, such as the big one—the Professional Developers Conference (PDC). Regardless of which conference you're looking for, Microsoft has provided a central site for providing event information, and enables users (such as yourself) to search the site for conferences, based on many different criteria. To find out what conferences or other events are going on in your area of interest of development focus, go to *events.microsoft.com.*

Other Resources

There are other resources available for developers of Windows applications, some of which might be mainstays for one developer and unheard of for another. The listing of developer resources in this chapter has been geared toward getting you more than started with finding the developer resources you need: it's geared toward getting you 100 percent of the way, but there are always exceptions.

Perhaps you're just getting started, and you want to get more hands-on instruction than MSDN Online or MCSD preparation materials provide. Where can you go? One option is to check out your local college for instructor-led courses. Most community colleges offer night classes, in case you have that pesky day job with which to contend and, increasingly, community colleges are outfitted with rather nice computer labs that enable you to get hands-on development instruction and experience, without having to work on a 386/20.

There are undoubtedly other resources that some people know about that have been useful, or maybe invaluable. If you have a resource that should be shared with others, let me know about it by sending me e-mail at the following address, and—who knows?—maybe someone else will benefit from your knowledge: *winprs@microsoft.com*

If you're sending e-mail about a particularly useful resource, type "Resources" in the subject line. There aren't any guarantees that you'll get a reply, but I'll read all of the e-mail and do what I can to ensure your resource idea gets considered.

CHAPTER 5

Getting the Most out of Win32 Library Technologies: Part 5

This chapter is the last of the five-part collection of common programming errors, included in the *Microsoft Win32 Developer's Reference Library* to help you avoid these simple programming pitfalls. This collection of common programming errors is distributed in each Win32 Library volume's Chapter 5 in the following fashion:

Volume 1: Overview and Solution Summary

Volume 2: Avoiding Invalid Validations

Volume 3: RPC Errors and Kernel-Mode Specifiers

Volume 4: Buffer Overflows and Miscellaneous Errors

Volume 5: Memory Abuse and Miscalculations

As you'll notice, not all of these pitfalls are necessarily confined to Win32 programming (some are networking services based, for example). However, since these common coding errors must be avoided in any Windows application, they're provided here in their entirety to round out the benefits of owning the Win32 Library.

This, of course, is Volume 5, and the errors and examples found in this chapter provide insights that can help you avoid problems with memory abuse and miscalculations in your programming projects. So, without further ado, here they are!

Memory Abuse

Memory abuse is an ailment that can plague any development project and can cause all sorts of unpleasant problems. Most problems associated with memory abuse can be avoided with a little care and understanding of the following basic rules:

- Always check for allocation failure.
- Always initialize data.
- Release (free/delete) any allocation once it's no longer needed.
- After memory is released, don't access it again! (Suggestion: Set the pointer to NULL upon freeing the memory.)
- Have quotas for how much a client can allocate (and ensure that client-specific data is protected).

Allocation failures

The most basic occurrence of allocation failures is the general allocation failure class. Throughout many programming projects, there are numerous cases where an allocation is not checked for failure before it's written to or read from. Don't fall victim to this common error; always check for allocation failure.

Example

```
BOOL
NtQaz(VOID)
{
    OBJECT *p;
    p = ExAllocatePoolWithTag(PagedPool,
        sizeof(OBJECT),
        'yguB');
    p -> FirstField = 0;
    […]
}
```

Remarks

The programmer who wrote this code is in trouble for two reasons. First, allocations sometimes fail, and code should behave gracefully when they do. The second problem is far subtler: On Windows NT and Windows 2000 it is possible to map a page at address 0x0; get two of these in a row and you have a fairly serious collision. Worse still is a kernel-mode caller that does this (as the example above implies), since the data can no longer be trusted.

Uninitialized memory

Another common memory abuse error is reading or returning uninitialized memory. While the problems associated with uninitialized memory are less obvious, the side effects of reading or returning uninitialized memory can be difficult to track because they can manifest themselves in seemingly random behavior. Non-static function variables are not initialized by default. Use of uninitialized values can result in random behavior, including exceptions and spurious failures. Good compilers might catch some of these problems, but not all. Furthermore, if uninitialized memory is returned to an external caller, it might contain sensitive application data. Somebody attacking your application or your system could analyze and use this data, especially from a kernel-mode allocation.

Example

```
typedef enum _WIDGET_TYPE {
    RedWidget,
    BlueWidget,
    GreenWidget,
    YellowWidget
};
```

```
NTSTATUS
FormatAWidget(
    WIDGET *NewWidget,
    WIDGET_TYPE Type
)
{
    NTSTATUS Status;

    switch (Type) {
    case BlueWidget:
    case GreenWidget:
        // Format the Widget
        […]
        Status = STATUS_SUCCESS;
        break;

    case RedWidget:
        // How did a Red Widget get here?
        Status = STATUS_INVALID_WIDGET;
        break;
    }
    return Status;
}
```

Remarks

If we get a yellow widget, what status is returned? We can't get a yellow widget because no code calls this function with a yellow widget. That might be true today, but not tomorrow. Why allow a bug to appear in the future? This is a good argument for a "default" in all switch statements to handle unexpected situations.

Avoiding leaks

Memory leaks are common occurrences in any code that makes allocations. As a general rule, make sure that allocated buffers that are not returned are freed when they are no longer needed. Specifically, be sure that all appropriate memory is released when you have common cleanup code for a number of paths.

Example

```
BOOL
Bar(VOID)
{
        OBJECT *p;
    p = malloc(sizeof (OBJECT));
    if (NULL == p) {
        return FALSE;
    } else {
```

(continued)

(continued)

```
        if (!OpenEvent(…)) {
                return FALSE;// BUG:  free(p)!
        } else {
            […]
        }
    }
    free(p);
    return TRUE;
}
```

Remarks

In this code sample, it's obvious that **p** leaks. Because **p** is freed in the success path, it should be freed in appropriate failure paths as well.

Don't use freed resources

Once memory is freed, it shouldn't be accessed again. This is an obvious rule, but this common programming error occurs all the time. This avoidance of using freed resources includes memory, resources, and objects controlled by reference counts. A common way that freed resources are accessed is when an attempt is made to release them a second time; a common way to prevent such problems is to set pointers to NULL upon release.

Example

```
BOOL
MumbleFrotz(OBJECT **p)
{
    *p = malloc(sizeof (OBJECT));
    if (NULL == *p) {
        return FALSE;
    } else {
        RtlCopyMemory(*p,
                        Source,
                        sizeof (OBJECT));
    }
    if (!SomethingThatFails(…)) {
        //
        // All well-behaved functions clean up
        // after themselves on failure.
        //
        free(*p);
        return FALSE;
    }
    return TRUE;
}
```

```
BOOL
Func(VOID)
{
    OBJECT *p = NULL;
    if (!MumbleFrotz(&p)) {
        free(p);    // BUG: Already freed.
        return FALSE;
    }
    […]
    free(p);
    return TRUE;
}
```

Remarks

This example is a case in which rearranging code from the start would have prevented the bug. **Func** should perform the allocation because **Func** releases it.

Resource attacks

While the other cases in this section are simple coding mistakes, protecting an application or service against a resource attack is a more complicated process. In general, a user should not be allowed to cause an application or a service to become inaccessible (a pretty obvious statement). To prevent denial-of-service attacks, it is important to set limits or quotas for any given application or service resource. Furthermore, to ensure data integrity and privacy, client-specific data should be private to each given client, and if the data must be global, it should be protected.

Miscalculations

Any time a calculation is made, the chance for errors crops up. Some of these calculation errors are obvious, some are a bit more involved. By being aware of the problems to look for, you're armed with most of what you need to know to prevent the errors. Specifically, be aware of the following:

- Be sure to check for zero for any division.
- Any signed value can be negative. Furthermore, be wary of the following:
  - Watch for implicit signed values. The values **int** and **enum** are signed. The value **char** is signed on Intel-based platforms (x86), but *not* on Alpha.
  - Use unsigned values where signed values don't make sense. Counts and lengths shouldn't be negative.
  - For range checks, check both upper and lower bounds (or specify unsigned).
- Floating-point operations: All floating-point operations should be surrounded by **try-except** protection.

Division by zero

Division by zero is one of the most common math-based exceptions. Whenever you're doing any division, make sure to check the divisor, even if the caller is supposed to be a trusted source. On algorithms that require division with either user-supplied variables or derivatives of user-supplied variables, take care not to allow a division by zero.

Example

```
NTSTATUS
NtFunc(ULONG x)
{
    ULONG r;
    if (0 == x) {
        return STATUS_INVALID_PARAMETER;
    }
    r = 4096 / (x / sizeof (ULONG));
    [...]
}
```

Remarks

Evaluate for *x* == 1.

Signed versus unsigned variables

A common blind spot when looking for programming errors is with negative numbers, especially when the type is implicitly signed (such as **int**, **enum**, and **char**). In general, check both upper and lower limits of the valid range. Furthermore, when negative values don't make sense (such as lengths and counts), use unsigned types. In most cases, unsigned variables make more sense. However, even unsigned variables must be used properly, as shown in the following example:

Example

```
typedef enum {
    ThisValue,
    ThatValue,
    MAX_ENUM_TYPE
} ENUM_TYPE;

BOOL
Api(ENUM_TYPE Val)
{
    ULONG t;
    if (Val > MAX_ENUM_TYPE) { // Should also check lower bound.
        SetLastError(ERROR_INVALID_PARAMETER);
        return FALSE;
    }
    t = GlobalArray[Val];
    […]
}
```

Remarks

Members of an enumerated type are signed. Passing 0x80000000 to this function will probably cause an exception.

```
BOOL
UnsignedProblem(USHORT x)
{
    while (--x >= 0) { // Infinite loop! Unsigned never negative!
        […]
    }
    return TRUE;
}
```

Floating-point variables

The basis of a floating-point variable problem is the fact that not all bit patterns are valid floating-point values. The IEEE floating-point definition specifies some special floating-point values, among them signaling values; these values cause exceptions to be raised when used. Although most cases will succeed, a malicious caller could cause an exception. To avoid such a situation, surround all floating-point calculations with exception handlers. A case to be particularly concerned with is using floating-point variables in kernel mode on Alpha computers.

Example

```
FLOAT
CalculateOffset(
    FLOAT Start,
    FLOAT End
)
{
    return End - Start;
}
```

Remarks

This example is simple enough, except that there are values defined as “signaling” in the IEEE floating-point specification that cause exceptions to be raised whenever they are encountered or created.

Solution Summary

It's nice to have a concise version of the solutions to these common programming problems, so this section summarizes how to avoid the issues discussed in this chapter.

Memory Abuse

1. Allocation failures: Always check for allocation failure
2. Uninitialized memory: Always initialize data.
3. Avoiding leaks: Release (free/delete) any allocation after it's no longer needed.
4. Don't use freed resources: After memory is released, don't access it again! (Suggestion: Set the pointer to NULL upon freeing the memory.)
5. Resource attacks: Have quotas for how much a client can allocate (and ensure that client-specific data is protected).

Miscalculations

1. Division by zero: Be sure to check for zero for any division.
2. Signed versus unsigned variables: Any signed value can be negative. Furthermore, be wary of the following:
   - Implicit signed values. The values **int** and **enum** are signed; **char** is signed on x86, but *not* on Alpha.
   - Use unsigned values where signed values don't make sense. Counts and lengths are not negative.
   - For ranges, check both upper and lower bounds (or specify unsigned)
3. Floating-point variables: All floating-point operations should be surrounded by **try-except** protection.

PART 2

Introduction

There are a handful of questions and versioning issues that, once addressed, can make your Shell programming experience a little easier. This introduction provides answers to questions that are commonly asked about the Windows Shell, and provides you with easy to find Common Control and Shell version information.

Commonly Asked Shell Questions

This section provides answers to commonly asked questions about the Windows Shell. For more information about any of these answers, read through the rest of this book's Part 2, and get the insights into Windows shell programming you need.

What is the shell namespace? What is a namespace object?

The shell *namespace* organizes the file system and other objects managed by the shell into a single tree-structured hierarchy. Conceptually, it is essentially a larger and more inclusive version of the file system. Name space objects include file system folders and files, along with "virtual" objects, such as the Recycle Bin and Printers folders.

Is there a dialog box that I can display to let the user choose a folder?

The **SHBrowseForFolder** function displays a dialog box that allows a user to select a directory, and returns its PIDL.

What is a PIDL? Why not just use file system paths?

A PIDL is a way of identifying any namespace object. You can also use paths to identify namespace objects, but only if they are part of the file system. With namespace objects that are not part of the file system, you must use PIDLs.

How do I get the PIDL of a namespace object?

There are a variety of ways to get an object's PIDL. Some common approaches are:

- Use the desktop's **IShellFolder::ParseDisplayName** method to convert a file system path into an equivalent PIDL. This method will also convert the GUID that identifies a virtual folder into a PIDL.
- Display a dialog box that allows the user to select a folder, and returns its PIDL.
- Use the folder's **CSIDL** to get its PIDL. Special folders, such as Program Files or Printers are assigned a token called a CSIDL. You can use a special folder's CSIDL to obtain its PIDL. If a special folder is in the file system, you can also use its CSIDL to obtain its path.
- Navigate the namespace until you locate the object.

How do I use SHGetFolderPath on systems prior to Windows 2000?

It is available as a redistributable DLL, ShFolder.dll.

How do I convert a PIDL back into a file path?

With the shell API's **SHGetPathFromIDList** function.

What is the difference between relative and fully-qualified PIDLs?

It is much like the difference between a relative and fully-qualified file path. Like a file path, a PIDL defines a path through the namespace, with one element for each portion of the path. Fully-qualified PIDLS start from the root of the namespace, the desktop. Relative PIDLS start from some other point in the namespace. Some shell functions expect fully-qualified PIDLs, and others expect relative PIDLs, so it is important to understand which is required.

How do I get a file's icon. How do I get a file's friendly name for a file?

The Shell API provides a function for this purpose: **SHGetFileInfo**.

Is there a way to customize how objects are displayed in Windows Explorer?

There are two ways to customize Windows Explorer:

- Create a Desktop.ini file for a folder.
- Create a custom folder.htt file.

How can I use drag-drop or the clipboard to transfer namespace objects such as files or folders?

There are a number of shell-specific clipboard formats that you can use to transfer shell objects.

What is a file association? What is a file class? How do I create one? What are they good for?

The terms *file association* and *file class* mean essentially the same thing. A file association or file class consists of all the files that have the same filename extension. File classes are created with the registry. Once a file class has been created, you can customize the behavior of its files. For instance, you can replace the standard file icon with a custom icon or add items to the context menu.

How can I get AutoPlay to launch my CD-ROM application?

By creating an AutoRun.inf file on the CD-ROM.

What exactly is the My Documents folder and how do I use it?

It provides a default location for the current user's document files. It automatically maps to the location in the file system where the current user's document files are stored. You use it much like a normal file system folder.

Shell and Common Controls Versions

This section describes how to determine which version of the Shell or Common Controls DLLs your application is running on and how to target your application for a specific version.

DLL Version Numbers

All but a handful of the programming elements discussed in the shell and common controls documentation are contained in three DLLs: Comctl32.dll, Shell32.dll, and Shlwapi.dll. Because of ongoing enhancements, different versions of these DLLs implement different features. Throughout this document, programming elements are marked with a version number. This version number indicates that the programming element was first implemented in that version and will also be found in all subsequent versions of the DLL. If no version number is specified, the programming element is implemented in all versions. The following table outlines the different DLL versions, and how they were distributed:

Version	DLL	Distribution platform
4.00	All	Microsoft Windows 95/Windows NT 4.0.
4.70	All	Microsoft Internet Explorer 3.x.
4.71	All	Microsoft Internet Explorer 4.0 (see Note 2).
4.72	All	Microsoft Internet Explorer 4.01 and Windows 98 (see Note 2).
5.00	Shlwapi.dll	Microsoft Internet Explorer 5 (see Note 3).
5.00	Shell32.dll	Microsoft Windows 2000. (see Note 3).
5.80	Comctl32.dll	Microsoft Internet Explorer 5 (see Note 3).
5.81	Comctl32.dll	Microsoft Windows 2000(see Note 3).

Note The 4.00 versions of Shell32.dll and Comctl32.dll are found on the original versions of Windows 95 and Windows NT 4. New versions of Commctl.dll were shipped with all Internet Explorer releases. Shlwapi.dll first shipped with Internet Explorer 4.0, so its first version number is 4.71. The shell was not updated with the Internet Explorer 3.0 release, so Shell32.dll does not have a version 4.70. While Shell32.dll versions 4.71 and 4.72 were shipped with the corresponding Internet Explorer releases, they were not necessarily installed (see Note 2). For subsequent releases, the version numbers for the three DLLs are not identical. In general, you should assume that all three DLLs may have different version numbers, and test each one separately.

Note All systems with Internet Explorer 4.0 or 4.01 will have the associated version of Comctl32.dll and Shlwapi.dll (4.71 or 4.72, respectively). However, for systems prior to Windows 98, Internet Explorer 4.0 and 4.01 can be installed with or without the *integrated shell*. If they are installed with the integrated shell, the associated version of Shell32.dll will be installed. If they are installed without the integrated shell, Shell32.dll is not updated. In other words, the presence of version 4.71 or 4.72 of Comctl32.dll or Shlwapi.dll on a system does not guarantee that Shell32.dll has the same version number. All Windows 98 systems have version 4.72 of Shell32.dll.

Note Version 5.80 of Comctl32.dll and version 5.0 of Shlwapi.dll are distributed with Internet Explorer 5. They will be found on all systems on which Internet Explorer 5 is installed, except Windows 2000. Internet Explorer 5 does not update the shell, so version 5.0 of Shell32.dll will not be found on Windows NT, Windows 95, or Windows 98 systems. Version 5.0 of Shell32.dll will be distributed with Windows 2000, along with version 5.0 of Shlwapi.dll, and version 5.81 of Comctl32.dll.

Using DllGetVersion to Determine the Version Number

Starting with version 4.71, the Shell and Common Controls DLLs, among others, began exporting **DllGetVersion**. This function can be called by an application to determine which DLL version is present on the system. It returns a structure that contains version information.

Note DLLs do not necessarily export **DllGetVersion**. Always test for it before attempting to use it.

For systems earlier than Windows 2000, **DllGetVersion** returns a **DLLVERSIONINFO** structure that contains the major and minor version numbers, the build number, and a platform ID. For Windows 2000 and later systems, **DllGetVersion** may instead return a **DLLVERSIONINFO2** structure. This structure contains the QFE number that identifies the service pack and provides a more robust way to compare version numbers than **DLLVERSIONINFO**. Since the first member of **DLLVERSIONINFO2** is a **DLLVERSIONINFO** structure, the new structure is backward-compatible.

Using DllGetVersion

The following sample function loads a specified DLL and attempts to call its **DllGetVersion** function. If successful, it uses a macro to pack the major and minor version numbers from the **DLLVERSIONINFO** structure into a DWORD that is returned to the calling application. If the DLL does not export **DllGetVersion**, the function returns zero. With Window 2000 and later systems, you can modify the function to handle the possibility that **DllGetVersion** returns a **DLLVERSIONINFO2** structure. If so, use the information contained in the **ullVersion** member to compare versions, build numbers, and service pack releases. The **MAKEDLLVERULL** macro is designed to simplify the task of comparing these values to those contained in **ullVersion**.

```
#define PACKVERSION(major,minor) MAKELONG(minor,major)

DWORD GetDllVersion(LPCTSTR lpszDllName)
{
    HINSTANCE hinstDll;
    DWORD dwVersion = 0;

    hinstDll = LoadLibrary(lpszDllName);

    if(hinstDll)
    {
        DLLGETVERSIONPROC pDllGetVersion;

        pDllGetVersion = (DLLGETVERSIONPROC) GetProcAddress(hinstDll,
"DllGetVersion");

/*Because some DLLs may not implement this function, you
 *must test for it explicitly. Depending on the particular
 *DLL, the lack of a DllGetVersion function may
 *be a useful indicator of the version.
*/
        if(pDllGetVersion)
        {
            DLLVERSIONINFO dvi;
            HRESULT hr;

            ZeroMemory(&dvi, sizeof(dvi));
            dvi.cbSize = sizeof(dvi);

            hr = (*pDllGetVersion)(&dvi);

            if(SUCCEEDED(hr))
            {
                dwVersion = PACKVERSION(dvi.dwMajorVersion, dvi.dwMinorVersion);
            }
        }

        FreeLibrary(hinstDll);
    }
    return dwVersion;
}
```

The following code fragment illustrates how you can use **GetDllVersion** to test if Comctl32.dll is version 4.71 or later.

```
if(GetDllVersion(TEXT("Comctl32.dll")) >= PACKVERSION(4,71))
{
    //Proceed
}
else
{
    //Use an alternate approach for older DLL versions
}
```

Project Versions

To ensure that your application is compatible with different targeted versions of Comctl32.dll and Shell32.dll, a version macro was added to the header files. This macro is used to define, exclude, or redefine certain definitions for different versions of the DLL. The macro name is **_WIN32_IE** and you, the developer, are responsible for defining the macro as a hexadecimal number. This version number defines the target version of the application that is using the DLL. The following are the currently available version numbers and the effect each has on your application.

Version	Description
0x0200	The application will be compatible with Comctl32.dll and Shell32.dll version 4.00 and later. The application will not be able to implement features that were added after version 4.00 of Comctl32.dll.
0x0300	The application will be compatible with Comctl32.dll and Shell32.dll version 4.70 and later. The application will not be able to implement features that were added after version 4.70 of Comctl32.dll.
0x0400	The application will be compatible with Comctl32.dll and Shell32.dll version 4.71 and later. The application will not be able to implement features that were added after version 4.71 of Comctl32.dll.
0x0401	The application will be compatible with Comctl32.dll and Shell32.dll version 4.72 and later. The application will not be able to implement features that were added after version 4.72 of Comctl32.dll.
0x0500	The application will be compatible with Comctl32.dll version 5.80 and later, and Shell32.dll and Shlwapi.dll version 5.0 and later. The application will not be able to implement features that were added after version 5.80 of Comctl32.dll or version 5.0 of Shell32.dll and Shlwapi.dll.
0x0501	The application will be compatible with Comctl32.dll version 5.81 and later and Shell32.dll and Shlwapi.dll version 5.0 and later. The application will not be able to implement features that were added after version 5.81 of Comctl32.dll or version 5.0 of Shell32.dll and Shlwapi.dll.

If you do not define this macro in your project, it will automatically be defined as 0x0500. To define a different value, you can add the following to the compiler directives in your make file (substitute the desired version number for 0x0400):

```
/D _WIN32_IE=0x0400
```

Another method is to add a line similar to the following in your source code *before* including the shell and common control header files (substitute the desired version number for 0x0400). For example:

```
#define _WIN32_IE 0x0400
#include <commctrl.h>
```

CHAPTER 6

Shell Programmer's Guide

The Microsoft Windows user interface (UI) gives users access to a wide variety of *objects* necessary for running applications and managing the operating system. The most numerous and familiar of these objects are the folders and files that reside on computer disk drives. There are also a number of *virtual objects* that allow the user to do tasks, such as send files to remote printers or access the recycle bin.

The *shell* organizes these objects into a hierarchical structure called the *namespace*, which provides users and applications with a consistent and efficient way to access and manage objects. Users interact with the namespace through the shell's graphical UI or through an application. Applications interact with the namespace through the shell's application programming interface (API). This chapter is an introduction to the shell API.

The shell API consists of a collection of functions, Component Object Model (COM) interfaces, and COM objects that provide applications with a rich set of tools to access and manage the namespace. It can used anywhere in an application with one important exception. Like all COM-based services, the shell API should not be used within a DLL's **DllMain** entry point function. Doing so may cause unpredictable behavior.

To use the shell API effectively in an application, you first need to understand the structure of the namespace and how namespace objects are identified.

Programming the Shell

The Shell API allows applications to perform a variety of tasks. Some of the more common ones are discussed in the following sections:

Getting a Folder's ID

Getting Information About the Contents of a Folder

Navigating the Namespace

Launching Applications

Managing the File System

Managing Printers

Transferring Shell Objects with Drag-Drop and the Clipboard

Integrating an Application with the Shell

By integrating your application with the shell, you can extend the shell's functionality and customize certain aspects of its behavior. In order of increasing complexity, you can extend the shell by:

1. Putting information in the registry or in special files.
2. Implementing a shell extension handler.
3. Implementing a namespace extension.

The first approach, which is used by many applications, is discussed here:

- Creating a File Association
- Customizing Icons
- Extending Context Menus
- Customizing Folders with Desktop.ini
- Creating an AutoPlay-enabled CD-ROM Application

For a discussion of how to write extension handlers, see *Creating Shell Extension Handlers*. For a discussion of how to write namespace extensions, see *Namespace Extensions*.

Note To improve the readability of the sample code in Shell Basics, most of the normal error-correction code has been removed. You should add error code, as appropriate, to your own applications. To make registry samples more readable, key names are in a bold font and values are in a normal font.

The Shell Namespace

The shell *namespace* organizes the file system and other objects managed by the shell into a single tree-structured hierarchy. Conceptually, it is essentially a larger and more inclusive version of the file system.

Introduction

One of the primary responsibilities of the shell is managing and providing access to the wide variety of objects that make up the system. The most numerous and familiar of these objects are the folders and files that reside on computer disk drives. However, the shell manages a number of nonfile system, or *virtual* objects, as well. Some examples include:

- Network printers
- Other networked computers
- Control Panel applications
- The Recycle Bin

Some virtual objects do not involve physical storage at all. The printer object, for instance, contains a collection of links to networked printers. Other virtual objects, such as the Recycle Bin, may contain data that is stored on a disk drive, but needs to be handled differently than normal files. For example, a virtual object can be used to represent data stored in a database. In terms of the namespace, the various items in the database could appear in the Windows Explorer as separate objects, even though they are all stored in a single disk file.

Virtual objects may even be located on remote computers. For instance, to facilitate roaming, a user's document files might be stored on a server. To give them access to their files from multiple desktop PCs, the My Documents folder on the desktop PC they are currently using will point to the server, not the desktop PC's hard disk. Its path will include either a mapped network drive, or a UNC path name.

Like the file system, the namespace includes two basic types of object: folders and files. Folder objects are the *nodes* of the tree; they are containers for file objects and other folders. File objects are the *leaves* of the tree; they are either normal disk files or virtual objects, such as printer links. Folders that are not part of the file system are sometimes referred to as *virtual folders*.

Like file system folders, the collection of virtual folders generally varies from system to system. There are three classes of virtual folders:

- Standard virtual folders, such as the Recycle Bin, that are found on all systems.
- Optional virtual folders that have standard names and functionality, but may not be present on all systems.
- Non-standard folders that are installed by the user.

Unlike file system folders, users cannot create new virtual folders themselves. They can only install ones created by third-party developers. The number of virtual folders is thus normally much fewer than the number of file system folders.

You can see a visual representation of how the namespace is structured in the Explorer Bar of the Windows Explorer. For example, Figure 6-1 shows a relatively simple namespace in Windows Explorer.

The ultimate root of the namespace hierarchy is the desktop. Immediately below the root are several virtual folders such as My Computer and the Recycle Bin.

The file systems of the various disk drives can be seen to be subsets of the larger namespace hierarchy. The roots of these file systems are subfolders of the My Computer folder. My Computer also includes the roots of any mapped network drives. Other nodes in the tree, such as My Documents, are virtual folders.

Identifying Namespace Objects

Before you can use a namespace object, you must first have a way of identifying it. An object in the file system could have a name such as MyFile.htm. Because there might be other files with that name elsewhere in the system, uniquely identifying a file or folder requires a fully-qualified path such as "C:\MyDocs\MyFile.htm". This path is basically an ordered list of all folders in a path from the file system root, C:\, ending with the file.

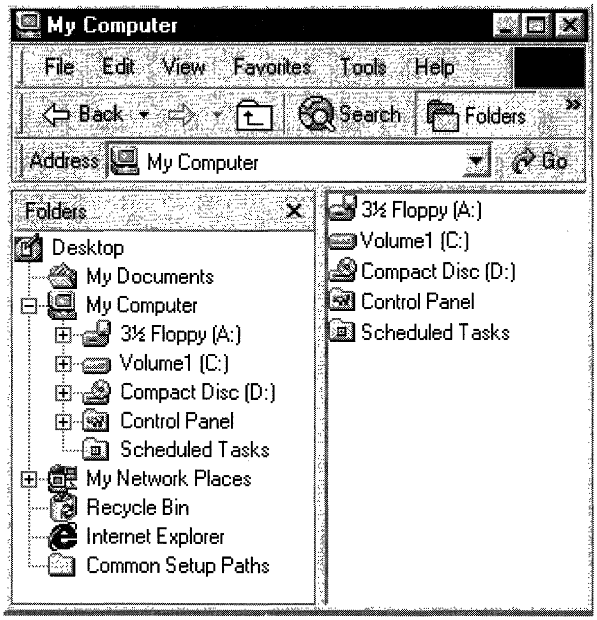


Figure 6-1: A simple namespace in Windows Explorer.

In the context of the namespace, paths are still quite useful for identifying objects located in the file system part of the namespace. However, they cannot be used for virtual objects. Instead, the shell provides an alternative means of identification that can be used with any namespace object.

Item IDs

Within a folder, each object has an *item ID*, which is the functional equivalent of a file or folder name. The item ID is actually a **SHITEMID** structure:

```
typedef struct _SHITEMID {
    USHORT cb;
    BYTE   abID[1];
} SHITEMID, * LPSHITEMID;
```

The **abID** member is the object's identifier. The length of **abID** is not defined, and its value is determined by the folder that contains the object. Because there is no standard definition for how **abID** values are assigned by folders, they are only meaningful to the associated folder object. Applications should simply treat them as a token that identifies an object in a particular folder. Because the length of **abID** varies, the **cb** member holds the size of the **SHITEMID** structure, in bytes.

Because item IDs aren't useful for display purposes, the folder that contains the object normally assigns it a display name. This is the name that is used by Windows Explorer when it displays the contents of a folder. For more information on how display names are handled, see *Getting Information From a Folder.*

Item ID Lists

The item ID is rarely used by itself. Normally, it is part of an item ID list, which serves the same purpose as a file system path. However, instead of the character string used for paths, an item ID list is an **ITEMIDLIST** structure. This structure is an ordered sequence of one or more item IDs, terminated by a two-byte NULL. Each item ID in the item ID list corresponds to a namespace object. Their order defines a path in the namespace, much like a file system path.

Figure 6-2 shows a schematic representation of the **ITEMIDLIST** structure that corresponds to C:\MyDocs\MyFile.htm. The display name of each item ID is shown above it. The varying widths of the **abID** members are arbitrary; they illustrate the fact that the size of this member can vary.

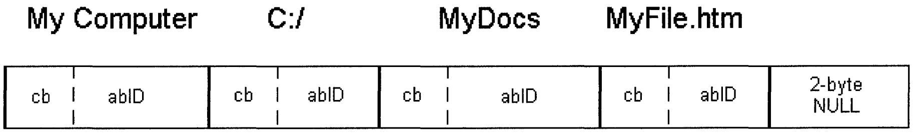


Figure 6-2: Schematic representation of the ITEMIDLIST **structure.**

PIDLs

For the shell API, namespace objects are usually identified by a pointer to their **ITEMIDLIST** structure, or *PIDL*. For convenience, the term PIDL will generally refer in this documentation to the structure itself rather than the pointer to it.

The PIDL shown in the preceding illustration is referred to as a *full*, or *absolute*, PIDL. A full PIDL starts from the desktop, and contains the item IDs of all intermediate folders in the path. It ends with the object's item ID followed by a terminating two-byte NULL. A full PIDL is essentially similar to a fully qualified path and uniquely identifies the object in the shell namespace.

Full PIDLs are actually used relatively infrequently. Many functions and methods expect a *relative PIDL*. Relative PIDLs have fewer item IDs, so they cannot be traced all the way back to the desktop. As with relative paths, the series of item IDs that make up the structure define a path in the namespace between two objects. Although they do not uniquely identify the object, they are generally smaller than a full PIDL and sufficient for many purposes.

The most commonly used relative PIDLs are relative to the object's parent folder; they contain only the object's item ID and a terminating NULL. Note that a PIDL can contain only a single item ID and still be a fully-qualified PIDL. In particular, desktop objects are children of the desktop, so their fully-qualified PIDLs contain only one item ID.

As discussed in *Getting a Folder's ID*, the shell API provides a number of ways to get an object's PIDL. Once you have it, you commonly just use it to identify the object when you call other shell API functions and methods. In this context, a PIDL's internal contents are opaque and irrelevant. For the purposes of this discussion, think of PIDLs as tokens that represent particular namespace objects, and focus on how to use them for common tasks.

Allocating PIDLs

Although PIDLs have some similarity to paths, using them requires a somewhat different approach. The primary difference is in how to allocate and deallocate memory for them.

Like the string used for a path, memory must be allocated for a PIDL. If an application creates a PIDL, it must allocate sufficient memory for the **ITEMIDLIST** structure. For most of the cases discussed here, the shell creates the PIDL and handles memory allocation. Regardless of what allocated the PIDL, the application is usually responsible for deallocating the PIDL when it is no longer needed.

To allocate and deallocate PIDLs, you must use the **IMalloc** interface exposed by the shell's allocator. To get a pointer to this interface, call **SHGetMalloc**. Use the **IMalloc::Alloc** method to allocate the PIDL, and the **IMalloc::Free** method to deallocate it. For an example of how to use this interface to handle shell memory allocation, see *Getting a Folder's ID*.

Getting a Folder's ID

Before you can make use of a namespace object, you need a way to identify it. This means obtaining either its PIDL or, in the case of file system objects, its path. This section discusses two of the simpler ways to obtain object IDs.

A more powerful approach that will work with any folder is to use the IShellFolder interface. See *Getting Information About the Contents of a Folder* for more details.

The SHBrowseForFolder Dialog Box

To enable the user to navigate the namespace and select a folder, your application can simply invoke **SHBrowseForFolder**. Calling this function launches a dialog box with a user interface (UI) that works somewhat like the Open or SaveAs common dialog boxes.

When the user selects a folder, **SHBrowseForFolder** returns its PIDL as well as its display name. If it is a file system folder, the application can convert the PIDL to a path by calling **SHGetPathFromIDList**. The application can also restrict the range of folders that the user can select from by specifying a root folder. Only folders that are below that root in the namespace will appear. Figure 6-3 shows the **SHBrowseForFolder** dialog box, with the root folder set to Program Files.

A simple example of how to use **SHBrowseForFolder** is provided later.

Special Folders and CSIDLs

A number of commonly used folders are designated as *special* by the system. These folders have a well-defined purpose, and most of them are present on all systems. Even if they are not present initially, their names and locations are still defined, so they can be added later. The collection of special folders includes all of the system's standard virtual folders, such as Printers, My Documents, and Network Neighborhood. It also includes a number of standard file system folders, such as Program Files and System.

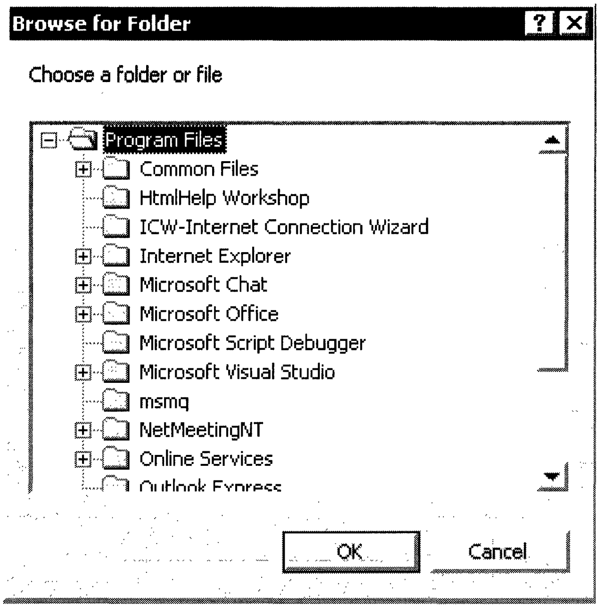


Figure 6-3: The SHBrowseForFolder **dialog box.**

Even though the folders are a standard component of all systems, their names and locations in the namespace can vary. For example, the System directory is C:\Winnt\System32 on some systems and C:\Windows\System32 on others. In the past, environment variables provided a way to determine the name and location of a special folder on any particular system. The shell now provides a more robust and flexible way to identify special folders, CSIDLs. You should generally use them instead of environment variables.

CSIDLs provide a uniform way of identifying and locating special folders, regardless of their name or location on a particular system. Unlike environment variables, CSIDLs can be used with virtual folders as well as file system folders. Each special folder has a unique CSIDL assigned to it. For example, the Program Files file system folder has a CSIDL of CSIDL_PROGRAM_FILES, and the Network Neighborhood virtual folder has a CSIDL of CSIDL_NETWORK.

A CSIDL is used in conjunction with one of several shell functions to retrieve a special folder's PIDL, or a special file system folder's path. If the folder doesn't exist on a system, your application can force it to be created by combining its CSIDL with CSIDL_FLAG_CREATE. The CSIDL can be passed to the following functions:

- **SHGetFolderLocation**, which retrieves the PIDL of a special folder.
- **SHGetFolderPath**, which retrieves the path of a file system special folder.

Note these two functions were introduced with version 5.0 of the shell and supersede the **SHGetSpecialFolderLocation** and **SHGetSpecialFolderPath** functions. To use **SHGetFolderPath** with earlier versions of the shell, you can include the redistributable DLL, *Shfolder.dll.*

An Example of How to Use CSIDLs and SHBrowseForFolder

The following sample function, PidlBrowse, illustrates how get a pointer to the shell allocator's **IMalloc** interface, use CSIDLs to retrieve a folder's PIDL, and use **SHBrowseForFolder** to have the user select a folder. It returns the PIDL and display name of the selected folder.

```
LPITEMIDLIST PidlBrowse(HWND hwnd, int nCSIDL, LPSTR pszDisplayName)
{
    LPITEMIDLIST pidlRoot = NULL;
    LPITEMIDLIST pidlSelected = NULL;
    BROWSEINFO bi = {0};
    LPMALLOC pMalloc = NULL;

    SHGetMalloc(&pMalloc);

    if(nCSIDL)
    {
        SHGetFolderLocation(hwnd, nCSIDL, NULL, NULL, &pidlRoot);
    }

    else
    {
        pidlRoot = NULL;
    }

    bi.hwndOwner = hwnd;
    bi.pidlRoot = pidlRoot;
    bi.pszDisplayName = pszDisplayName;
    bi.lpszTitle = "Choose a folder";
    bi.ulFlags = 0;
    bi.lpfn = NULL;
    bi.lParam = 0;

    pidlSelected = SHBrowseForFolder(&bi);

    if(pidlRoot)
    {
        pMalloc->Free(pidlRoot);
    }
    pMalloc->Release();
    return pidlSelected;
}
```

The calling application passes in a window handle, which is needed by **SHBrowseForFolder**. The *nCSIDL* parameter is an optional CSIDL that is used to specify a root folder. Only folders below the root folder in the hierarchy will be displayed. The illustration shown earlier was generated by calling this function with *nCSIDL* set to CSIDL_PROGRAM_FILES. The calling application also passes in a string buffer, *pszDisplayName*, to hold the display name of the selected folder when PidlBrowse returns.

PidlBrowse first calls **SHGetMalloc** to get a pointer to the shell's allocator. Although no PIDLs are allocated by the function itself, the **IMalloc** interface will be needed later to deallocate them. If the calling application specifies a root folder by passing in its CSIDL, PidlBrowse calls **SHGetFolderLocation** to retrieve the folder's PIDL. The function then assigns appropriate values to a **BROWSEINFO** structure, and passes it to **SHBrowseForFolder**.

After the user selects a folder, **SHBrowseForFolder** returns its PIDL. The folder's display name is returned in the **pszDisplayName** member of the **BROWSEINFO** structure, and is passed back to the calling application through the *pszDisplayName* parameter. Finally, PidlBrowse deallocates the root PIDL, releases the **IMalloc** interface, and returns the selected folder's PIDL to the calling application.

Getting Information About the Contents of a Folder

The *Getting a Folder's ID* section discussed two approaches to getting a namespace object's PIDL. One obvious question is: Once you have a PIDL, what can you do with it? A related question is: What if neither approach works, or is suitable for your application? The answer to both questions requires taking a closer look at how the namespace is implemented. The key is the **IShellFolder** interface.

Using the IShellFolder Interface

Earlier in this documentation, namespace folders were referred to as objects. Although, at that point, the term was used in a loose sense, it is actually true in a strict sense as well. Every namespace folder is represented by a Component Object Model (COM) object. Each folder object exposes a number of interfaces that can be used for a wide variety of tasks. Some interfaces that are optional may not be exposed by all folders. However, all folders must expose the fundamental interface, **IShellFolder**.

The first step in using a folder object is to get a pointer to its **IShellFolder** interface. In addition to providing access to the object's other interfaces through its **QueryInterface** method, **IShellFolder** exposes a group of methods that handle a number of common tasks, several of which are discussed in this section.

To get a pointer to a namespace object's **IShellFolder** interface, you must first call **SHGetDesktopFolder**. This function returns a pointer to the **IShellFolder** interface of the namespace root, the desktop. Once you have the desktop's **IShellFolder** interface, there a variety of ways to proceed.

If you already have the PIDL of the folder you are interested in—for instance, by calling **SHGetFolderLocation**—you can get its **IShellFolder** interface by calling the desktop's **IShellFolder::BindToObject** method. If you have the path of a file system object, you must first obtain its PIDL by calling the desktop's **IShellFolder::ParseDisplayName** method and then call **IShellFolder::BindToObject**. If neither of these approaches is applicable, you can use other **IShellFolder** methods to navigate the namespace. For more information, see *Navigating the Namespace.*

Enumerating the Contents of a Folder

The first thing you usually want to do with a folder is to find out what it contains. You must first call the folder's **IShellFolder::EnumObjects** method. The folder will create a standard OLE enumeration object and return its **IEnumIDList** interface. This interface exposes four standard methods—Clone, Next, Reset, and Skip—that can be used to enumerate the contents of the folder.

The basic procedure for enumerating a folder's contents is:

1. Call the folders **IShellFolder::EnumObjects** method to get a pointer to an enumeration object's **IEnumIDList** interface.
2. Pass an unallocated PIDL to **IEnumIDList::Next**. **Next** takes care of allocating the PIDL, but the application must deallocate it when it is no longer needed. When **Next** returns, the PIDL will contain just the object's item ID and the terminating NULLs. In other words, it is not a full PIDL, but is instead relative to the folder.
3. Repeat step 2 until **Next** returns S_FALSE to indicate that all items have been enumerated.
4. Call **IEnumIDList::Release** to release the enumeration object.

Note It is important to keep track of whether you are working with a full or relative PIDL. Some functions and methods will accept either, but others will only take one or the other.

The remaining three **IEnumIDList** methods (**Reset**, **Skip**, and **Clone**) are useful if you need to do repeated enumerations of the folder. They allow you to reset the enumeration, skip one or more objects, and make a copy of the enumeration object to preserve its state.

Determining Display Names and Other Properties

Once you have enumerated all the PIDLs that are contained by a folder, you can find out what sort of objects they represent. The **IShellFolder** interface provides a number of useful methods, two of which are discussed here. Other **IShellFolder** methods and other shell folder interfaces are discussed later.

One of the most useful properties is the object's display name. To get the display name of an object, pass its PIDL to **IShellFolder::GetDisplayNameOf**. Although the object can be located anywhere below the parent folder in the namespace, its PIDL must be relative to the folder.

IShellFolder::GetDisplayNameOf returns the display name as part of a **STRRET** structure. Because extracting the display name from a **STRRET** structure can be a little tricky, the shell provides two functions that do the job for you, **StrRetToStr** and **StrRetToBuf**. Both functions take a **STRRET** structure, and return the display name as a normal string. They differ only in how the string is allocated.

In addition to its display name, an object can have a number of attributes, such as whether it is a folder or whether it can be moved. You can retrieve an object's attributes by passing its PIDL to **IShellFolder::GetAttributesOf**. The complete list of attributes is quite large, so you should see the reference for details. Note that the PIDL that you pass to **GetAttributesOf** must be relative to the folder and can contain only one item ID. In particular, it will accept the PIDLs returned by **IEnumIDList::Next**. You can pass in an array of PIDLs, and **GetAttributesOf** will return those attributes that all objects in the array have in common.

If you have an object's fully-qualified path or PIDL, **SHGetFileInfo** provides a simple way to get information about an object that is sufficient for many purposes. **SHGetFileInfo** takes a fully-qualified path or PIDL, and returns a variety of information about the object including:

- The object's display name
- The object's attributes
- Handles to the object's icons
- A handle to the system image list
- The path of the file containing the object's icon

Getting a Pointer to a Subfolder's IShellFolder Interface

You can determine whether your folder contains any subfolders by calling **IShellFolder::GetAttributesOf** and checking to see if the SFGAO_FOLDER flag is set. If an object is a folder, you can bind to it, which provides you with a pointer to its **IShellFolder** interface.

To bind to a subfolder, call the parent folder's **IShellFolder::BindToObject** method. This method takes the subfolder's PIDL and returns a pointer to its **IShellFolder** interface. Once you have this pointer, you can use the **IShellFolder** methods to enumerate the subfolders contents, determine their properties, and so on.

Determining an Object's Parent Folder

If you have an object's PIDL, you may need a handle to one of the interfaces exposed by its parent folder. For example, if you want to determine the display name associated with a PIDL by using **IShellFolder::GetDisplayNameOf**, you must first get the **IShellFolder** interface of the object's parent. It is possible to do this with the techniques discussed in the previous sections. However, a much simpler approach is to use the shell function, **SHBindToParent**. This function takes the fully-qualified PIDL of an object and returns a specified interface pointer on the parent folder. Optionally, it also returns the item's PIDL relative to the parent for use in methods such as **IShellFolder::GetAttributesOf**.

The following sample console application gets the PIDL of the System special folder and returns its display name:

```
#include <shlobj.h>
#include <shlwapi.h>
#include <iostream.h>

int main()
{
    LPMALLOC pMalloc = NULL;
    IShellFolder *psfParent = NULL;
    LPITEMIDLIST pidlSystem = NULL;
    LPCITEMIDLIST pidlRelative = NULL;
    STRRET strDispName;
    TCHAR szDisplayName[MAX_PATH];
    HRESULT hr;

    hr = SHGetMalloc(&pMalloc);

    hr = SHGetFolderLocation(NULL, CSIDL_SYSTEM, NULL,
NULL, &pidlSystem);

    hr = SHBindToParent(pidlSystem, IID_IShellFolder,
(void **) &psfParent, &pidlRelative);

    if(SUCCEEDED(hr))
    {
        hr = psfParent->GetDisplayNameOf(pidlRelative,
SHGDN_NORMAL, &strDispName);
        hr = StrRetToBuf(&strDispName, pidlSystem,
szDisplayName, sizeof(szDisplayName));
        cout << "SHGDN_NORMAL - " <<szDisplayName << '\n';
    }

    psfParent->Release();
    pMalloc->Free(pidlSystem);
    pMalloc->Release();

    return 0;
}
```

The application first gets a pointer to the shell allocator's **Imalloc** interface and uses **SHGetFolderLocation** to get the System folder's PIDL. It then calls **SHBindToParent**, which returns a pointer to the parent folder's **IShellFolder** interface, and the System folder's PIDL relative to its parent. It then uses the parent folder's **IShellFolder::GetDisplayNameOf** method to get the display name of the System folder.

Because **GetDisplayNameOf** returns a **STRRET** structure, **StrRetToBuf** is used to convert the display name into a normal string. After displaying the display name, the interface pointers are released and the System PIDL freed. Note that you must not free the relative PIDL returned by **SHBindToParent**.

Navigating the Namespace

You now have all the essential elements needed to navigate anywhere in the namespace. The simplest way to start is to have your application call **SHGetDesktopFolder** to get the desktop's **IShellFolder** interface. Then, to navigate downward through the namespace, your application can follow these steps:

1. Enumerate the folder's contents.
2. Determine which objects are subfolders, and select one.
3. Bind to the subfolder to get its **IShellFolder** interface.

Repeat these steps as often as necessary to reach the target.

An Example of Namespace Navigation

The following piece of sample code is a simple console application that illustrates a number of the procedures discussed in the preceding sections. The application performs the following tasks:

1. Gets the Program Files folder's **IShellFolder** interface (*Using the IShellFolder Interface*).
2. Enumerates the contents of the folder (*Enumerating the Contents of a Folder*).
3. Determines all the display names and prints them (*Determining Display Names and Other Properties*).
4. Looks for a subfolder (Determining Display Names and Other Properties).
5. Binds to the first subfolder it finds (Getting a Pointer to a Subfolder's IShellFolder Interface).
6. Prints the display names of the objects in the subfolder.

```
#include <shlobj.h>
#include <shlwapi.h>
#include <iostream.h>

main()
{
    LPMALLOC pMalloc;
    LPITEMIDLIST pidlProgFiles = NULL;
    LPITEMIDLIST pidlItems = NULL;
    IShellFolder *psfFirstFolder = NULL;
    IShellFolder *psfDeskTop = NULL;
```

(continued)

(continued)

```
 IShellFolder *psfProgFiles = NULL;
    LPENUMIDLIST ppenum = NULL;
    ULONG celtFetched;
    HRESULT hr;
    STRRET strDispName;
    TCHAR pszDisplayName[MAX_PATH];
    ULONG uAttr;

    hr = SHGetMalloc(&pMalloc);

    hr = SHGetFolderLocation(NULL, CSIDL_PROGRAM_FILES,
NULL, NULL, &pidlProgFiles);

    hr = SHGetDesktopFolder(&psfDeskTop);

    hr = psfDeskTop->BindToObject(pidlProgFiles, NULL,
IID_IShellFolder, (LPVOID *) &psfProgFiles);
    hr = psfDeskTop->Release();

    hr = psfProgFiles->EnumObjects(NULL,SHCONTF_FOLDERS |
SHCONTF_NONFOLDERS, &ppenum);

    while( (hr = ppenum->Next(1,&pidlItems, &celtFetched)
!= S_FALSE) && (celtFetched) == 1)
    {
        psfProgFiles->GetDisplayNameOf(pidlItems,SHGDN_INFOLDER, &strDispName);
        StrRetToBuf(&strDispName, pidlItems,pszDisplayName, MAX_PATH);
        cout << pszDisplayName << '\n';
        if(!psfFirstFolder)
        {
            psfProgFiles->GetAttributesOf(1,
(LPCITEMIDLIST *) &pidlItems, &uAttr);
            if(uAttr & SFGAO_FOLDER)
            {
                hr = psfProgFiles->BindToObject(pidlItems,
NULL, IID_IShellFolder, (LPVOID *) &psfFirstFolder);
            }
        }
    }

    cout << "\n\n";

    ppenum->Release();
    hr = psfFirstFolder->EnumObjects(NULL,SHCONTF_FOLDERS |
SHCONTF_NONFOLDERS, &ppenum);

    while( (hr = ppenum->Next(1,&pidlItems, &celtFetched)
!= S_FALSE) && (celtFetched) == 1)
    {
        psfFirstFolder->GetDisplayNameOf(pidlItems,
SHGDN_INFOLDER, &strDispName);
        StrRetToBuf(&strDispName, pidlItems, pszDisplayName, MAX_PATH);
        cout << pszDisplayName << '\n';
    }

    ppenum->Release();
    pMalloc->Free(pidlProgFiles);
    pMalloc->Free(pidlItems);
    psfProgFiles->Release();
    psfFirstFolder->Release();

    return 0;
}
```

Launching Applications

Once your application has located a file object, the next step is often to act on it in some way. For instance, your application might want to launch another application that allows the user to modify a data file. If the file of interest is an executable, your application might want to simply launch it. This document discusses how to use **ShellExecute** or **ShellExecuteEx** to perform these tasks.

Using ShellExecute and ShellExecuteEx

To use **ShellExecute** or **ShellExecuteEx**, your application must specify the file or folder object that is to be acted on, and a *verb* that specifies the operation. For **ShellExecute**, assign these values to the appropriate parameters. For **ShellExecuteEx**, fill in the appropriate members of a **SHELLEXECUTEINFO** structure. There are also several other members or parameters that can be used to fine-tune the behavior of the two functions.

File and folder objects can be part of the file system or virtual objects, and they can be identified by either paths or PIDLs. The available verbs are essentially the items that you find on an object's context menu. As for context menus, the exact list of verbs available depends on the particular object. Commonly available verbs include:

Open	Launches an application. If the file is not executable, it launches the file's associated application.
Edit	Opens a file's editor.
Print	Prints a document file.
Display	Displays an object's properties.

Each verb corresponds to the command that would be used to launch the application from a console window. The **open** verb is a good example, as it is commonly supported. For .exe files, **open** simply launches the application. However, it is more commonly used to launch an application that operates on a particular file. For instance, by default, .txt files are opened by NotePad. The **open** verb for .txt file, thus, corresponds to something like the following command:

```
"%SystemRoot%\System32\NotePad.exe" "%1"
```

When you use **ShellExecute** or **ShellExecuteEx** to open a .txt file, NotePad.exe is launched with the specified file as its argument. Some commands can have additional arguments, such as flags, that can be added as needed to launch the application properly. For further discussion of context menus and verbs, see *Extending Context Menus*.

In general, trying to determine the list of available verbs for a particular file is somewhat complicated. In many cases, you can simply set the *lpVerb* parameter to NULL, which invokes the default command for the file class. This procedure is usually equivalent to setting *lpVerb* to "open", but some file classes may have a different default command. For further information, see *Extending Context Menus* and the ShellExecuteEx reference documentation.

Using ShellExecute to Launch the Find Dialog Box

When a user right-clicks a folder icon in windows Explorer, one of the menu items is "Find." If they select that item, the shell launches its Find utility. This utility displays a dialog box that can be used to search files for a specified text string. An application can programmatically launch the Find utility for a directory by calling **ShellExecute**, with "find" as the *lpVerb* parameter, and the directory path as the *lpFile* parameter. For instance, the following line of code launches the Find utility for the c:\MyPrograms directory.

```
ShellExecute(hwnd,"find","c:\\MyPrograms",NULL, NULL, 0);
```

An Example of How to Use ShellExecuteEx

The following sample console application illustrates the use of **ShellExecuteEx**.

```
#include <shlobj.h>
#include <shlwapi.h>

main()
{
    LPMALLOC pMalloc;
    LPITEMIDLIST pidlWinFiles = NULL;
    LPITEMIDLIST pidlItems = NULL;
    LPITEMIDLIST pidlFirstBmp = NULL;
    IShellFolder *psfWinFiles = NULL;
    IShellFolder *psfDeskTop = NULL;
    LPENUMIDLIST ppenum = NULL;
    STRRET strDispName;
    TCHAR pszParseName[MAX_PATH];
    ULONG celtFetched;
    SHELLEXECUTEINFO ShExecInfo;
    HRESULT hr;

    hr = SHGetMalloc(&pMalloc);

    hr = SHGetFolderLocation(NULL, CSIDL_WINDOWS, NULL, NULL, &pidlWinFiles);

    hr = SHGetDesktopFolder(&psfDeskTop);

    hr = psfDeskTop->BindToObject(pidlWinFiles, NULL,
IID_IShellFolder, (LPVOID *) &psfWinFiles);
    hr = psfDeskTop->Release();

    hr = psfWinFiles->EnumObjects(NULL,SHCONTF_FOLDERS |
SHCONTF_NONFOLDERS, &ppenum);

    while( (hr = ppenum->Next(1,&pidlItems, &celtFetched)
!= S_FALSE) && (celtFetched) == 1)
    {
        psfWinFiles->GetDisplayNameOf(pidlItems, SHGDN_FORPARSING, &strDispName);
        StrRetToBuf(&strDispName, pidlItems, pszParseName, MAX_PATH);
        if(StrRStrI(pszParseName, NULL, ".bmp"))
        {
            break;
        }

    }

    ppenum->Release();

    ShExecInfo.cbSize = sizeof(SHELLEXECUTEINFO);
    ShExecInfo.fMask = NULL;
    ShExecInfo.hwnd = NULL;
    ShExecInfo.lpVerb = NULL;
    ShExecInfo.lpFile = pszParseName;
    ShExecInfo.lpParameters = NULL;
    ShExecInfo.lpDirectory = NULL;
    ShExecInfo.nShow = SW_MAXIMIZE;
    ShExecInfo.hInstApp = NULL;

    ShellExecuteEx(&ShExecInfo);
```

(continued)

(continued)

```
pMalloc->Free(pidlWinFiles);
    pMalloc->Free(pidlItems);
    pMalloc->Free(pidlFirstBmp);
    psfWinFiles->Release();

    return 0;
}
```

The application first gets the PIDL of the Windows directory, and enumerates its contents until it finds the first .bmp file. Unlike the earlier example, **IShellFolder::GetDisplayNameOf** is used to get the file's parsing name instead of its display name. Because this is a file system folder, the parsing name is a fully qualified path, which is what is needed for **ShellExecuteEx**.

Once the first .bmp file has been located, appropriate values are assigned to the members of a **SHELLEXECUTEINFO** structure. The **lpFile** member is set to the parsing name of the file, and the **lpVerb** member to NULL, to begin the default operation. In this case, the default operation is "open." The structure is then passed to **ShellExecuteEx**, which launches MSPaint.exe to open the bitmap file. After the function returns, the PIDLs are freed and the Windows folder's **IShellFolder** interface is released.

Managing the File System

The shell provides a number of ways to manage file systems. The shell provides a function, **SHFileOperation**, that allows an application to programmatically move, copy, rename, and delete files. The shell also supports some additional file management capabilities:

- HTML documents can be *connected* to related files, such as graphics files or style sheets. When the document is moved or copied, the connected files are automatically moved or copied as well.
- For systems that are available to more than one user, files can be managed on a per-user basis. Users have easy access to their data files, but not to files belonging to other users.
- If document files are added or modified, they can be added to the shell's list of recent documents. When the user clicks the Documents command on the Start menu, a list of links to the documents appears.

This document discusses how these new file management technologies work. It then outlines how to use the shell to move, copy, rename, and delete files, and how to manage objects in the recycle bin.

Per-User File Management

The Microsoft Windows 2000 shell allows files to be associated with a particular user, so they remain hidden from other users. In terms of the file system, they are generally stored under the users profile folder, typically C:\Documents and Settings*Username*\ on Windows 2000 systems. This feature allows many individuals to use the same machine, while maintaining the privacy of their files from other users. Different users can have different programs available. It also provides a straightforward way for administrators and applications to store such things as initialization (.ini) or link (.lnk) files. Applications can thus preserve a different state for each user, and easily recover that particular state when needed. There is also a profile folder for storing information that is common to all users.

Because it is inconvenient to determine which user is logged in and where their files are located, the standard per-user folders are special folders, and are identified by a **CSIDL**. For instance, the CSIDL for the per-user Program Files folder is CSIDL_PROGRAMS. If your application calls **SHGetFolderLocation** or **SHGetFolderPath** with one of the per-user CSIDLs, the function will return the PIDL or path appropriate to the currently logged in user. If your application needs to get the path or PIDL of the profile folder, its CSIDL is **CSIDL_PROFILE**.

The My Documents and My Pictures Folders

One of the standard icons found on the desktop is My Documents. When you use the Windows Explorer to look at the namespace, the My Documents folder is immediately below the desktop. The My Pictures folder is a subfolder of My Documents. The purpose of these folders is to provide a default location for the current user's document and picture files.

There are actually separate My Documents and My Pictures file system folders for each user. For example, the location of a user's My Documents folder in the file system will be something like this: C:\Documents and Settings*Username*\My Documents. However, there is no need for the user to access this file system folder. They simply use My Documents, which is automatically mapped to the corresponding personal file system folder. Note that if multiple people use the same computer, this part of the file system can be locked by an administrator to prevent users from accessing files belonging to others. There is no way for a user to gain access to anyone else's files through My Documents.

There is usually no need for an application to know which user is logged in or where in the file system their My Documents folder is located. Instead, your application can get the PIDL of the desktop icon by calling the desktop's **IShellFolder::ParseDisplayName** method. The parsing name used to identify My Documents folder is not the file path, but rather "::{CLSID_MYDOCUMENTS}" where CLSID_XXX is the text form of the corresponding GUID. For example, to get the PIDL of My Documents, your application should use:

```
hr = psfDeskTop->ParseDisplayName(NULL, NULL,
L"::{450d8fba-ad25-11d0-98a8-0800361b1103}",
&chEaten, &pidlDocFiles, NULL);
```

Once your application has the My Documents PIDL, it can handle the PIDL as it would any other folder. The shell will automatically map changes in My Documents to the appropriate file system folder.

If your application needs access to the actual file system folders, the CSIDLs is CSIDL_PERSONAL for My Documents. Your application can pass the CSIDL to **SHGetFolderLocation** to get the PIDL of the current user's My Documents folder. Call **SHGetFolderPath** to get the path.

Connected Files

HTML documents often have a number of associated graphics files, a stylesheet file, several JScript files, and so on. When you move or copy the primary HTML document, you also usually want to move or copy its associated files to avoid breaking links. Unfortunately, there has been no easy way until now to determine which files are related to any given HTML document other than by analyzing their contents. To alleviate this problem, Windows 2000 provides a simple way to *connect* a primary HTML document to its group of associated files. If file connection is enabled, when the document is moved or copied, all its connected files go with it.

To create a group of connected files, the primary document must have an .htm or .html file name extension. Create a subfolder of the primary document's parent folder. The subfolder's name must be the name of the primary document, minus the extension, followed by "files." For instance, if the primary document is named MyDoc.htm, the subfolder must be named "MyDoc files". Any files that are placed in this subfolder will be connected to the primary document. If the primary document is moved or copied, the subfolder and its files will be moved or copied as well.

Note For some locales, you may need to replace "files" with its localized equivalent.

Whether or not file connection is enabled is controlled by a REG_DWORD value, **NoFileFolderConnection**, of the **HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer** registry key. By default, this value is not defined, and file connection is enabled. To disable file connection, add this value to the key, if necessary, and set it to one. To enable file connection again, set **NoFileFolderConnection** to zero.

Moving, Copying, Renaming, and Deleting Files

The namespace is not static, and applications commonly need to manage the file system by performing one of the following operations:

- Copying an object to another folder.
- Moving an object to another folder.
- Deleting an object.
- Renaming an object.

The move, copy, rename, and delete operations are all be performed with **SHFileOperation**. This function takes one or more source files, and produces corresponding destination files. In the case of the delete operation, the system attempts to put the deleted files in the Recycle Bin.

To use the function, you must fill in the members of a **SHFILEOPSTRUCT** structure, and pass it to **SHFileOperation**. The key members of the structure are **pFrom** and **pTo**.

The **pFrom** member contains one or more source file names. These names can be either fully qualified paths, or standard DOS wild cards such as "*.*". Although this member is declared as a null-terminated string, it is used as a buffer to hold multiple file names. Each file name must be terminated by a single NULL character. An additional NULL character must be appended to the end of the final name to indicate the end of **pFrom**.

The **pTo** member contains the names of one or more fully qualified destination names. They are packed into **pTo** in the same way as they are for **pFrom**. If **pTo** contains multiple names, you must also set the FOF_MULTIDESTFILES flag in the **fFlags** member. The usage of **pTo** depends on the operation as described here:

- For copy and move operations, if all the files are going to a single directory, **pTo** contains the fully qualified directory name. If the files are going to different destinations, **pTo** can also contain one fully qualified directory or file name for each source file. If a directory doesn't exist, the system will create it.
- For rename operations, **pTo** contains one fully qualified path for each source file in **pFrom**.
- For delete operations, **pTo** is not used.

Notifying the Shell

You should notify the shell of the change after using **SHFileOperation** to move, copy, rename, or delete files, or after any taking any other action that affects the namespace. Actions that should be accompanied by notification include:

- Adding or deleting files or folders.
- Moving, copying, or renaming files or folders.
- Changing a file association.
- Changing file attributes.
- Adding or removing drives or storage media.
- Creating or disabling a shared folder.
- Changing the system image list.

An application notifies the shell by calling **SHChangeNotify** with the details of what has changed. The shell can then update its image of the namespace to accurately reflect its new state.

An Example of Managing Files with SHFileOperation

The following sample console application illustrates the use of **SHFileOperation** to copy files from one directory to another. The source and destination directories, C:\My_Docs and C:\My_Docs2, are hard-coded into the application for simplicity.

```
#include <shlobj.h>
#include <shlwapi.h>

main()
{
    IShellFolder *psfDeskTop = NULL;
    IShellFolder *psfDocFiles = NULL;
    IMalloc *pMalloc = NULL;
    LPITEMIDLIST pidlDocFiles = NULL;
    LPITEMIDLIST pidlItems = NULL;
    IEnumIDList *ppenum = NULL;
    SHFILEOPSTRUCT sfo;
    STRRET strDispName;
    TCHAR szParseName[MAX_PATH];
    TCHAR szSourceFiles[256];
    int i;
    int iBufPos = 0;
    ULONG chEaten;
    ULONG celtFetched;
    HRESULT hr;

    pzSourceFiles[0] = '\0';
    hr = SHGetMalloc(&pMalloc);
    hr = SHGetDesktopFolder(&psfDeskTop);

    hr = psfDeskTop->ParseDisplayName(NULL, NULL,
L"c:\\My_Docs", &chEaten, &pidlDocFiles, NULL);
    hr = psfDeskTop->BindToObject(pidlDocFiles, NULL,
IID_IShellFolder, (LPVOID *) &psfDocFiles);
    hr = psfDeskTop->Release();

    hr = psfDocFiles->EnumObjects(NULL,SHCONTF_FOLDERS |
SHCONTF_NONFOLDERS, &ppenum);

    while( (hr = ppenum->Next(1,&pidlItems, &celtFetched)
== S_OK) && (celtFetched) == 1)
    {
        psfDocFiles->GetDisplayNameOf(pidlItems, SHGDN_FORPARSING, &strDispName);
        StrRetToBuf(&strDispName, pidlItems, szParseName, MAX_PATH);
        for(i=0;i<=lstrlen(szParseName); i++)
        {
            szSourceFiles[iBufPos++] = szParseName[i];
        }
    }
    ppenum->Release();

    szSourceFiles[iBufPos] = '\0';

    sfo.hwnd = NULL;
    sfo.wFunc = FO_COPY;
    sfo.pFrom = szSourceFiles;
    sfo.pTo = "c:\\My_Docs2\0";
    sfo.fFlags = FOF_SILENT | FOF_NOCONFIRMATION | FOF_NOCONFIRMMKDIR;

    hr = SHFileOperation(&sfo);

    SHChangeNotify(SHCNE_UPDATEDIR, SHCNF_PATH, (LPCVOID) "c:\My_Docs2", 0);

    pMalloc->Free(pidlDocFiles);
    pMalloc->Free(pidlItems);
    psfDocFiles->Release();

    return 0;
}
```

The application first gets a pointer to the desktop's **IShellFolder** interface. It then gets the source directory's PIDL by passing its fully qualified path to **IShellFolder::ParseDisplayName**. Note that **ParseDisplayName** requires the directory's path to be a Unicode string. The application then binds to the source directory and uses its **IShellFolder** interface to get an enumerator object's **IEnumIDList** interface.

As each file in the source directory is enumerated, **IShellFolder::GetDisplayNameOf** is used to get its name. The SHGDN_FORPARSING flag is set, which causes **GetDisplayNameOf** to return the file's fully qualified path. The file paths, including the terminating NULL characters, are concatenated into a single array, *szSourceFiles*. A second NULL character is appended to the final path to terminate the array properly.

Once the enumeration is complete, the application assigns values to a **SHFILEOPSTRUCT** structure. Note that the array assigned to **pTo** to specify the destination must also be terminated by a double NULL. In this case, it is simply included in the string that is assigned to **pTo**. Because this is a console application, the FOF_SILENT, FOF_NOCONFIRMATION, and FOF_NOCONFIRMMKDIR flags are set to suppress any dialog boxes that might appear. After **SHFileOperation** returns, **SHChangeNotify** is called to notify the shell of the change. Then the application performs the usual cleanup and returns.

Adding Files to the Shell's List of Recent Documents

The shell maintains a list of recently added or modified documents for each user. The user can display a list of links to these files by clicking Documents on the Start menu. As with My Documents, each user has a file system directory to hold the actual links. To get the PIDL of the current user's Recent directory, your application can call **SHGetFolderLocation** with CSIDL_RECENT, or call **SHGetFolderPath** to get its path.

Your application can enumerate the contents of the Recent folder using the techniques discussed earlier in this document. However, an application should not modify the contents of the folder as if it were a normal file system folder. If it does so, the shell's list of recent documents will not be updated properly, and the changes will not be reflected in the Start menu. Instead, to add a document link to the a user's Recent folder, your application can call **SHAddToRecentDocs**. The shell will add a link to the appropriate file system folder, as well as updating its list of recent documents and the Start menu. You can also use this function to clear the folder.

Managing Printers

The shell Application Programming Interface (API) provides functions that you can use to manage networked printers. If a file has the **print** verb associated with it, you can use the **ShellExecuteEx** command to print it.

Printer Management

You can manage printers on a system with the **SHInvokePrinterCommand** function. This function allows you to:

- Install printers.
- Open printers.
- Get printer properties.
- Create printer links.
- Print a test page.

Printing Files with ShellExecuteEx

If a file type has a print command associated with it, you can print the file by calling **ShellExecuteEx** with **print** as the verb. This command is often the same as that used for the **open** verb, with the addition of a flag to tell the application to print the file. For example, the default print command for .txt files is:

```
%SystemRoot%\system32\NOTEPAD.EXE /p %1
```

When you use **ShellExecuteEx** to print a .txt file, NotePad opens the file, prints it, and then closes, returning control to the application. The following sample function takes a fully qualified path, and uses **ShellExecuteEx** to print it, using the print command associated with its file name extension.

```
#include <shlobj.h>

HINSTANCE PrintFile(LPCTSTR pszFileName)
{
    SHELLEXECUTEINFO ShExecInfo;
    HINSTANCE hInst;

//Fill the SHELLEXECUTEINFO array.

    ShExecInfo.cbSize = sizeof(SHELLEXECUTEINFO);
    ShExecInfo.fMask = NULL;
    ShExecInfo.hwnd = NULL;
    ShExecInfo.lpVerb = "print";
    ShExecInfo.lpFile = pszFileName;
    ShExecInfo.lpParameters = NULL;
    ShExecInfo.lpDirectory = NULL;
    ShExecInfo.nShow = SW_MAXIMIZE;
    ShExecInfo.hInstApp = NULL;

    hInst = ShellExecuteEx(&ShExecInfo);

    return hInst;
}
```

Transferring Shell Objects with Drag-Drop and the Clipboard

Many applications allow users to transfer data to another application by dragging and dropping the data with the mouse, or by using the Clipboard. Among the many types of data that can be transferred are shell objects such as files or folders. Shell data transfer can take place between two applications, but users can also transfer shell data to or from the desktop or Microsoft Windows Explorer.

Although files are the most commonly transferred shell object, shell data transfer can involve any of the variety of objects found in the shell namespace. For instance, your application might need to transfer a file to a virtual folder such as the Recycle Bin, or accept an object from a third-party namespace extension. If you are implementing a namespace extension, it must be able to behave properly as a drop source and target.

This document discusses how applications can implement drag-drop and Clipboard data transfers with shell objects.

How Drag-Drop Works with Shell Objects

Applications often need to provide users with a way to transfer shell data. Some examples are:

- Dragging a file from Windows Explorer or the desktop and dropping it on an application.
- Copying a file to the Clipboard in Windows Explorer and pasting it into an application.
- Dragging a file from an application to the Recycle Bin.

For a detailed discussion of how to handle these and other scenarios, see *Handling Shell Data Transfer Scenarios*. This document focuses on the general principles behind shell data transfer.

Windows provides two standard ways for applications to transfer shell data:

- A user cuts or copies shell data, such as one or more files, to the Clipboard. The other application retrieves the data from the Clipboard.
- A user drags an icon that represents the data from the source application and drops the icon on a window owned by the target.

In both cases, the transferred data is contained in a *data object*. Data objects are COM objects that expose the **IDataObject** interface. Schematically, there are three essential steps that all shell data transfers must follow:

1. The source creates a data object that represents the data that is to be transferred.
2. The target receives a pointer to the data object's **IDataObject** interface.
3. The target calls the **IDataObject** interface to extract the data from it.

The difference between Clipboard and drag-drop data transfers lies primarily in how the **IDataObject** pointer gets from the source to the target.

Clipboard Data Transfers

The Clipboard is the simplest way to transfer shell data. The basic procedure is similar to standard Clipboard data transfers. However, because you are transferring a pointer to a data object, not the data itself, you must use the OLE clipboard API instead of the standard clipboard API. The following procedure outlines how to use the OLE clipboard API to transfer shell data with the Clipboard:

1. The data source creates a data object to contain the data.
2. The data source calls **OleSetClipboard**, which places a pointer to the data object's **IDataObject** interface on the Clipboard.
3. The target calls **OleGetClipboard** to retrieve the pointer to the data object's **IDataObject** interface.
4. The target extracts the data by calling the **IDataObject::GetData** method.
5. With some shell data transfers, the target might also need to call the data object's **IDataObject::SetData** method to provide feedback to the data object on the outcome of the data transfer. See *Handling Optimized Move Operations* for an example of this type of operation.

Drag-Drop Data Transfers

While somewhat more complex to implement, drag-drop data transfer has some significant advantages over the Clipboard:

- Drag-drop transfers can be done with a simple mouse movement, making operation more flexible and intuitive to use than the Clipboard.
- Drag-drop provides the user with a visual representation of the operation. The user can follow the icon as it moves from source to target.
- Drag-drop notifies the target when the data is available.

Drag-drop operations also use data objects to transfer data. However, the drop source must provide functionality beyond that required for Clipboard transfers:

- The drop source must also create an object that exposes an **IDropSource** interface. The system uses **IDropSource** to communicate with the source while the operation is in progress.
- The drag-drop data object is responsible for tracking cursor movement and displaying an icon to represent the data object.

Drop targets must also provide more functionality than is needed to handle Clipboard transfers:

- The drop target must expose an **IDropTarget** interface. When the cursor is over a target window, the system uses **IDropTarget** to provide the target with information such as the cursor position, and to notify it when the data is dropped.
- The drop target must register itself with the system by calling **RegisterDragDrop**. This function provides the system with the handle to a target window and a pointer to the target application's **IDropTarget** interface.

Note For drag-drop operations, your application must initialize COM with **OleInitialize**, not **CoInitialize**.

The following procedure outlines the essential steps that are typically used to transfer shell data with drag-drop:

1. The target calls **RegisterDragDrop** to give the system a pointer to its IDropTarget interface and register a window as a drop target.
2. When the user starts a drag-drop operation, the source creates a data object and initiates a *drag loop* by calling **DoDragDrop**.
3. When the cursor is over the target window, the system notifies the target by calling one of the target's **IDropTarget** methods. The system calls **IDropTarget::DragEnter** when the cursor enters the target window, and **IDropTarget::DragOver** as the cursor passes over the target window. Both methods provide the drop target with the current cursor position and the state of keyboard modifier keys such as CTRL or ESCAPE.

When the cursor leaves the target window, the system notifies the target by calling **IDropTarget::DragLeave**. When any of these methods return, the system calls the **IDropSource** interface to pass the return value to the source.

4. When the user releases the mouse button to drop the data, the system calls the target's **IDropTarget::Drop** method. Among the method's parameters is a pointer to the data object's **IDataObject** interface.
5. The target calls the data object's **IDataObject::GetData** method to extract the data.
6. With some shell data transfers, the target might also need to call the data object's **IDataObject::SetData** method to provide feedback to the source on the outcome of the data transfer.
7. When the target is finished with the data object, it returns from **IDropTarget::Drop**. The system returns the source's **DoDragDrop** call to notify the source that the data transfer is complete.
8. Depending on the particular data transfer scenario, the source might need to take additional action based on the value returned by **DoDragDrop** and the values that are passed to the data object by the target. For instance, when a file is moved, the source must check these values to determine whether it must delete the original file.
9. The source releases the data object.

While the procedures outlined above provide a good general model for shell data transfer, there are many different types of data that can be contained in a shell data object. There are also a number of different data transfer scenarios that your application might need to handle. Each data type and scenario requires a somewhat different approach to three key steps in the procedure:

- How a source constructs a data object to contain the shell data.
- How a target extracts shell data from the data object.
- How the source completes the data transfer operation.

The Shell Data Object provides a general discussion of how a source constructs a shell data object, and how that data object can be handled by the target. *Handling Shell Data Transfer Scenarios* discusses in detail how to handle a number of common shell data transfer scenarios.

The Shell Data Object

The data object is central to all shell data transfers. It is primarily a container to hold the transferred data. However, the target can also communicate with the data object to facilitate some specialized types of shell data transfer such as optimized moves. This documentation provides a general discussion of how shell data objects work, how they are constructed by a source, and how they are handled by a target. For a detailed discussion of how to use data objects to transfer different types of shell data, see *Handling Shell Data Transfer Scenarios*.

How Data Objects Work

Data objects are COM objects, created by the data source to transfer data to a target. They typically carry more than one item of data. There are two primary reasons for this practice:

- While almost any type of data can be transferred with a data object, the source typically does not know what kind of data the target can accept. For instance, the data might be a portion of a formatted text document. While the target might be able to handle complex formatting information, it might also only be able to accept ANSI text. For this reason, data objects often include the same data in several different formats. The target can then extract the data in a format that it can handle.
- Data objects can also contain auxiliary data items that are not versions of source data. This type of data item typically provides additional information about the data transfer operation. For instance, the shell uses auxiliary data items to indicate whether a file is to be copied or moved.

Clipboard Formats

Each item of data in a data object has an associated format, usually called a *clipboard format*. There are a number of standard clipboard formats, declared in Winuser.h, that correspond to commonly used types of data. Clipboard formats are integers, but they are normally referred to by their equivalent name, which has the form CF_*XXX*. For instance, the clipboard format for ANSI text is CF_TEXT.

Applications can extend the range of available clipboard formats by defining private formats. To define a private format, an application calls **RegisterClipBoardFormat** with a string that identifies the format. The unsigned integer that the function returns is a valid format value that can be used just like a standard clipboard format. However, both source and target must register the format in order to use it. With one exception—CF_HDROP—the clipboard formats used to transfer shell data are defined as private formats. They must be registered by the source and target before they can be used. For a description of the available shell clipboard formats, see *Shell Clipboard Formats*.

Although there are some exceptions, data objects normally contain only one item of data for each clipboard format they support. This one-to-one correlation between format and data allows the format value to be used as an identifier for the associated data item. In fact, when discussing the contents of a data object, a particular item of data is typically called a "format" and is referred to by its format name. For example, phrases such as "Extract the CF_TEXT format..." are typically used when discussing a data object's ANSI text data item.

When the target receives the pointer to the data object, it enumerates the available formats to determine what types of data are available. It then requests one or more of the available formats and extracts the data. The specific way that the target extracts shell data from a data object varies with the format; this is discussed in detail in *How a Target Handles a Shell Data Object*.

With simple clipboard data transfers, the data is placed in a global memory object. The address of that object is placed on the clipboard, along with its format. The clipboard format basically tells the target what kind of data it will find at the associated address. While simple clipboard transfers are easy to implement:

- Data objects provide a much more flexible way to transfer data.
- Data objects are better suited for transferring large amounts of data.
- Data objects must be used to transfer data with a drag-drop operation.

For these reasons, all shell data transfers use data objects. With data objects, clipboard formats are not used directly. Instead, data items are identified with a generalization of the clipboard format, a **FORMATETC** structure.

The FORMATETC Structure

The **FORMATETC** structure is essentially an extended version of a clipboard format. As used for shell data transfers, the **FORMATETC** structure has the following characteristics:

- A data item is still identified by its clipboard format, in the **cfFormat** member.
- Data transfer is not limited to global memory objects. The **tymed** member is used to indicate the data transfer mechanism contained in the associated **STGMEDIUM** structure. It is set to one of the **TYMED_XXX** values.
- The shell uses the **lindex** member with its **CFSTR_FILECONTENTS** format to allow a data object to contain more than one data item per format. See *Using the CFSTR_FILECONTENTS Format to Extract Data from a File* for a discussion of how to use this format.
- The **dwAspect** member is typically set to DVASPECT_CONTENT. However, there are three values defined in Shlobj.h that can be used for shell data transfer.

DVASPECT_COPY	Used with the CF_HDROP format to request a file path with the names shortened to the 8.3 format.
DVASPECT_LINK	Used to indicate that the format represents a shortcut to the data.
DVASPECT_SHORTNAME	Used to indicate that the format represents a copy of the data.

- The **ptd** member is not used for shell data transfers and is normally set to NULL.

The STGMEDIUM structure

The **STGMEDIUM** structure provides access to the data being transferred. Three data transfer mechanisms are supported for shell data:

- A global memory object.
- An **IStream** interface.
- An **IStorage** interface.

The **tymed** member of the **STGMEDIUM** structure is a **TYMED_XXX** value that identifies the data transfer mechanism. The second member is a pointer that is used by the target to extract the data. The pointer can be one of a variety of types, depending on the **tymed** value. The three **tymed** values that are used for shell data transfers are summarized in the following table, along with their corresponding **STGMEDIUM** member name.

tymed value	Member name	Description
TYMED_HGLOBAL	**hGlobal**	A pointer to a global memory object. This pointer type is typically used for transferring small amounts of data. For instance, the shell uses global memory objects to transfer short text strings such as file names or URLs.
TYMED_ISTREAM	**pstm**	A pointer to an **IStream** interface. This pointer type is preferred for most shell data transfers because it requires relatively little memory compared to TYMED_HGLOBAL. Also, the TYMED_ISTREAM data transfer mechanism does not require the source to store its data in any particular way.
TYMED_ISTORAGE	**pstg**	A pointer to an **IStorage** interface. The target calls the interface methods to extract the data. Like TYMED_ISTREAM, this pointer type requires relatively little memory. However, because TYMED_ISTORAGE is less flexible than TYMED_ISTREAM, it is not as commonly used.

How a Source Creates a Data Object

When a user initiates a shell data transfer, the source is responsible for creating a data object and loading it with data. The following procedure summarizes the process:

1. Call **RegisterClipBoardFormat** to obtain a valid clipboard format value for each shell format that will be included in the data object. Remember that **CF_HDROP** is already a valid clipboard format and does not need to be registered.
2. For each format to be transferred, either put the associated data into a global memory object or create an object that provides access to that data through an **IStream** or **IStorage** interface. The **IStream** and **IStorage** interfaces are created using standard COM techniques. For a discussion of how to handle global memory objects, see *How to Add a Global Memory Object to a Data Object.*
3. Create **FORMATETC** and **STGMEDIUM** structures for each format.
4. Instantiate a data object.

5. Load the data into the data object by calling the **IDataObject::SetData** method for each supported format and passing in the format's **FORMATETC** and **STGMEDIUM** structures.
6. With clipboard data transfers, call **OleSetClipboard** to place a pointer to the data object's **IDataObject** interface on the clipboard. For drag-drop transfers, initiate a *drag loop* by calling **DoDragDrop**. The **IDataObject** pointer will be passed to the drop target when the data is dropped, ending the drag loop.

The data object is now ready to be transferred to the target. For clipboard data transfers, the object is simply held until the target requests it by calling **OleGetClipboard**. For drag-drop data transfers, the data object is responsible for creating an icon to represent the data and moving it as the user moves the cursor. While the object is in the drag loop, the source receives status information through its **IDropSource** interface. For further discussion, see *Implementing IDropSource*.

The source receives no notification if the data object is retrieved from the clipboard by a target. When an object is dropped on a target by a drag-drop operation, the **DoDragDrop** function that was called to initiate the drag loop will return.

How to add a global memory object to a data object

Many of the shell data formats are in the form of a global memory object. Use the following procedure to create a format containing a global memory object and load it into the data object:

1. Create a **FORMATETC** structure. Set the **cfFormat** member to the appropriate clipboard format value and the **tymed** member to TYMED_HGLOBAL.
2. Create an **STGMEDIUM** structure. Set the **tymed** member to **TYMED_HGLOBAL.**
3. Create a global memory object by calling **GlobalAlloc** to allocate a suitably sized block of memory.
4. Assign the block of data to be transferred to the address returned by **GlobalAlloc.**
5. Assign the global memory object's address to the **hGlobal** member of the **STGMEDIUM** structure.
6. Load the format into the data object by calling **IDataObject::SetData** and passing in the **FORMATETC** and **STGMEDIUM** structures created in the previous steps.

The following sample function creates a global memory object containing a DWORD value and loads it into a data object. The *pdtobj* parameter is a pointer to the data object's **IDataObject** interface, *cf* is the clipboard format value, and *dw* is the data value.

```
STDAPI DataObj_SetDWORD(IDataObject *pdtobj, UINT cf, DWORD dw)
{
    FORMATETC fmte = {(CLIPFORMAT) cf, NULL, DVASPECT_CONTENT, -1,
        TYMED_HGLOBAL};
    STGMEDIUM medium;
    HRESULT hres = E_OUTOFMEMORY;
    DWORD *pdw = (DWORD *)GlobalAlloc(GPTR, sizeof(DWORD));
    if (pdw)
    {
        *pdw = dw;
        medium.tymed = TYMED_HGLOBAL;
        medium.hGlobal = pdw;
        medium.pUnkForRelease = NULL;

        hres = pdtobj->SetData(&fmte, &medium, TRUE);

        if (FAILED(hres))
            GlobalFree((HGLOBAL)pdw);
    }
    return hres;
}
```

Implementing IDataObject

IDataObject is a data object's primary interface. It must be implemented by all data objects. It is used by both source and target for a variety of purposes, including:

- Loading data into the data object.
- Extracting data from the data object.
- Determining what types of data are in the data object.
- Providing feedback to the data object on outcome of the data transfer.

IDataObject supports a number of methods. This document discusses how to implement the three most important methods for shell data objects, **SetData**, **EnumFormatEtc**, and **GetData**. For a discussion of the other methods, see the **IDataObject** reference.

The SetData method

The primary function of the **SetData** method is to allow the source to load data into the data object. For each format to be included, the source creates a **FORMATETC** structure to identify the format and an **STGMEDIUM** structure to hold a pointer to the data. The source then calls the object's **IDataObject::SetData** method and passes in the format's **FORMATETC** and **STGMEDIUM** structures. The method must store this information so that it is available when the target calls **GetData** to extract data from the object.

However, when transferring files, the shell often puts the information for each file to be transferred into a separate **CFSTR_FILECONTENTS** format. To distinguish the different files, the **lindex** member of each file's **FORMATETC** structure is set to an index value that identifies the particular file. Your **SetData** implementation must be capable of storing multiple CFSTR_FILECONTENTS formats that differ only by their **lindex** members.

While the cursor is over the target window, the target can use the drag-drop helper object to specify the drag image. The drag-drop helper object calls **SetData** to load private formats into the data object that are used for cross-process support. To support the drag-drop helper object, your **SetData** implementation must be able to accept and store arbitrary private formats.

After the data has been dropped, some types of shell data transfer require the target to call **SetData** to provide the data object with information about the outcome of the drop operation. For example, when moving files with an optimized move operation, the target normally deletes the original files, but it is not required to do so. The target informs the data object whether or not it deleted the files by calling **SetData** with a **CFSTR_LOGICALPERFORMEDDROPEFFECT** format. There are several other shell clipboard formats that are also used by the target to pass information to the data object. Your **SetData** implementation must be able to recognize these formats and respond appropriately. For further discussion, see *Handling Shell Data Transfer Scenarios*.

The EnumFormatEtc method

When the target receives a data object, it commonly calls **EnumFormatEtc** to determine what formats the object contains. The method creates an OLE enumeration object and returns a pointer to the object's **IEnumFORMATETC** interface. The target then uses the interface to enumerate the available formats.

An enumeration object for shell data is implemented in much the same way as for other types of data transfer, with one notable exception. Because data objects typically contain only one data item per format, they normally enumerate every format that is passed to

IDataObject::SetData. However, as discussed in the SetData method, shell data objects can contain multiple **CFSTR_FILECONTENTS** formats.

Because the purpose of **EnumFormatEtc** is to allow the target to determine what types of data are present, there is no need to enumerate more than one CFSTR_FILECONTENTS format. If the target needs to know how many of these formats the data object contains, the target can get that information from the accompanying **CFSTR_FILEDESCRIPTOR** format. For further discussion of how to implement **EnumFormatEtc**, see the method's reference documentation.

The GetData method

The target calls **GetData** to extract a particular data format. The target specifies the format by passing in the appropriate **FORMATETC** structure. **GetData** returns the format's **STGMEDIUM** structure.

The target can set the **tymed** member of the **FORMATETC** structure to a specific TYMED_*XXX* value to specify which data transfer mechanism it will use to extract the data. However, the target can also make a more generic request and let the data object decide. To ask the data object to select the data transfer mechanism, the target sets all the TYMED_*XXX* values that it supports. **GetData** selects one of these data transfer mechanisms and returns the appropriate **STGMEDIUM** structure. For instance, **tymed** is commonly set to TYMED_HGLOBAL|TYMED_ISTREAM|TYMED_ISTORAGE to request any of the three shell data transfer mechanisms.

Note Because there can be multiple **CFSTR_FILECONTENTS** formats, the **cfFormat** and **tymed** members of the **FORMATETC** structure are not sufficient to indicate which **STGMEDIUM** structure **GetData** should return. For the CFSTR_FILECONTENTS format, **GetData** must also examine the **FORMATETC** structure's **lIndex** member in order to return the correct **STGMEDIUM** structure.

The **CFSTR_INDRAGLOOP** format is placed in data objects to allow targets to check the status of the drag-drop loop while avoiding expensive rendering of the object's data. The format's data is a DWORD value that is set to a nonzero value if the data object is within a drag loop. The format's data value is set to zero if the data has been dropped. If a target requests this format and it has not been loaded by the source, **GetData** should respond as if the source had loaded the format with a value of zero.

While the cursor is over the target window, the target can use the drag-drop helper object to specify the drag image. The drag-drop helper object calls **SetData** to load private formats into the data object that are used for cross-process support. It later calls **GetData** to retrieve them. To support the drag-drop helper object, your **GetData** implementation must be able to return arbitrary private formats when they are requested.

Implementing IDropSource

The source must create an object that exposes an **IDropSource** interface. This interface allows the source to update the *drag image* that indicates the current position of the cursor, and provide feedback to the system on how to terminate a drag-drop operation. **IDropSource** has two methods: **GiveFeedback** and **QueryContinueDrag**.

The GiveFeedback method

While in the drag loop, a drop source is responsible for keeping track of the cursor position and displaying an appropriate drag image. However, in some cases you might want to change the appearance of the drag image when it is over the drop target's window.

When the cursor enters or leaves the target window and while it is moving over the target window, the system periodically calls the target's **IDropTarget** interface. The target responds with a **DROPEFFECT** value that is forwarded to the source through the **IDropSource::GiveFeedback** method. If appropriate, the source can modify the appearance of the cursor based on the DROPEFFECT value. For further details, see the **IDropSource::GiveFeedback** and **DoDragDrop** references.

The QueryContinueDrag method

This method is called if the mouse button or keyboard state changes while the data object is in the drag loop. It notifies the source whether the ESCAPE key has been pressed and provides the current state of the keyboard modifier keys, such as CONTROL or SHIFT. The **QueryContinueDrag** method's return value specifies one of three actions:

- S_OK. Continue the drag operation.
- DRAGDROP_S_DROP. Drop the data. The system will then call the target's **IDropTarget::Drop method.**
- DRAGDROP_S_CANCEL. Terminate the drag loop without dropping the data. This value is normally returned if the ESCAPE key was pressed.

For further discussion, see the **IDropSource::QueryContinueDrag** and **DoDragDrop** references.

How a Target Handles a Data Object

The target receives a data object when it either retrieves the data object from the clipboard or has it dropped on the target window by the user. The target can then extract data from the data object. If necessary, the target can also notify the data object of the outcome of the operation. Prior to a shell data transfer, a drop target must prepare itself for the operation:

1. The target must call **RegisterClipBoardFormat** to obtain a valid clipboard format value for all shell formats, other than **CF_HDROP**, that might be included in the data object. CF_HDROP is already a valid clipboard format and does not need to be registered.
2. To support a drag-drop operation, the target must implement an **IDropTarget** interface and register a target window. To register a target window, the target calls **RegisterDragDrop** and passes in the window's handle and the **IDropTarget** interface pointer.

For clipboard transfers, the target does not receive any notification that a data object has been placed on the clipboard. Typically, an application is notified that an object is on the clipboard by a user action, such as clicking the Paste button on the application's toolbar. The target then retrieves the data object's **IDataObject** pointer from the clipboard by calling **OleGetClipboard**. For drag-drop data transfers, the system uses the target's **IDropTarget** interface to provide the target with information about the progress of the data transfer:

- The system calls **IDropTarget::DragEnter** when the cursor enters the target window.
- The system periodically calls **IDropTarget::DragOver** as the cursor passes over the target window, to give the target the current cursor position.
- The system calls **IDropTarget::DragLeave** when the cursor leaves the target window.
- The system calls **IDropTarget::Drop** when the user drops the data object on the target window.

For a discussion of how to implement these methods, see *Implementing IDropTarget.*

When the data is dropped, **IDropTarget::Drop** provides the target with a pointer to the data object's **IDataObject** interface. The target then uses this interface to extract data from the data object.

Extracting Shell Data from a Data Object

Once a data object has been dropped or retrieved from the clipboard, the target can extract the data it needs. The first step in the extraction process is typically to enumerate the formats contained by the data object:

- Call **IDataObject::EnumFormatEtc**. The data object creates a standard OLE enumeration object and returns a pointer to its **IEnumFORMATETC** interface.
- Use the **IEnumFORMATETC** methods to enumerate the formats contained by the data object. This operation usually retrieves one **FORMATETC** structure for each format that the object contains. However, the enumeration object normally returns only a single **FORMATETC** structure for the **CFSTR_FILECONTENTS** format, regardless of how many such formats are contained by the data object.
- Select one or more formats to be extracted, and store their **FORMATETC** structures.

To retrieve a particular format, pass the associated **FORMATETC** structure to **IDataObject::GetData**. This method returns an **STGMEDIUM** structure that provides access to the data. To specify a particular data transfer mechanism, set the **tymed** value of the **FORMATETC** structure to the corresponding TYMED_*XXX* value. To ask the data object to select a data transfer mechanism, the target sets the TYMED_*XXX* values for every data transfer mechanism that the target can handle. The data object selects one of these data transfer mechanisms and returns the appropriate **STGMEDIUM** structure.

For most formats, the target can retrieve the data by passing the **FORMATETC** structure that it received when it enumerated the available formats. One exception to this rule is **CFSTR_FILECONTENTS**. Because a data object can contain multiple instances of this format, the **FORMATETC** structure returned by the enumerator might not correspond to the particular format you want to extract. In addition to specifying the **cfFormat** and **tymed** members, you must also set the **lIndex** member to the file's index value. For further discussion, see *Using the CFSTR_FILECONTENTS Format to Extract Data from a File.*

The data extraction process depends on the type of pointer contained by the returned **STGMEDIUM** structure. If the structure contains a pointer to an **IStream** or **IStorage** interface, use the interface methods to extract the data. The process of extracting data from a global memory object is discussed in the next section.

How to extract a global memory object from a data object

Many of the shell data formats are in the form of a global memory object. Use the following procedure to extract a format containing a global memory object from a data object and assign its data to a local variable:

1. Create a **FORMATETC** structure. Set the **cfFormat** member to the appropriate clipboard format value and the **tymed** member to TYMED_HGLOBAL.
2. Create an empty **STGMEDIUM** structure.
3. Call **IDataObject::GetData**, and pass in pointers to the **FORMATETC** and **STGMEDIUM** structures.
4. When **IDataObject::GetData** returns, the **STGMEDIUM** structure will contain a pointer to the global memory object that contains the data.
5. Assign the data to a local variable by calling **GlobalLock** and passing in the **hGlobal** member of the **STGMEDIUM** structure.
6. Call **GlobalUnlock** to release the lock on the global memory object.
7. Call **ReleaseStgMedium** to release the global memory object.

The following example shows how to extract a DWORD value stored as a global memory object from a data object. The *pdtobj* parameter is a pointer to the data object's **IDataObject** interface, *cf* is the clipboard format that identifies desired data, and *pdwOut* is used to return the data value.

```
STDAPI DataObj_GetDWORD(IDataObject *pdtobj, UINT cf, DWORD *pdwOut)
{
    STGMEDIUM medium;
    FORMATETC fmte = {(CLIPFORMAT) cf, NULL, DVASPECT_CONTENT, -1,
        TYMED_HGLOBAL};

    HRESULT hres = pdtobj->GetData(&fmte, &medium);

    if (SUCCEEDED(hres))
    {
        DWORD *pdw = (DWORD *)GlobalLock(medium.hGlobal);

        if (pdw)
        {
            *pdwOut = *pdw;
            GlobalUnlock(medium.hGlobal);
        }

        else
        {
            hres = E_UNEXPECTED;
        }

        ReleaseStgMedium(&medium);
    }
    return hres;
}
```

Implementing IDropTarget

The system uses the **IDropTarget** interface to communicate with the target while the cursor is over the target window. The target's responses are forwarded to the source through its **IDropSource** interface. Depending on the response, the source can modify the icon that represents the data. If the drop target needs to specify the data icon, it can do so by creating a drag-drop helper object.

With conventional drag-drop operations, the target informs the data object of the outcome of the operation by setting the *pdwEffect* parameter of **IDropTarget::Drop** to the appropriate **DROPEFFECT** value. With shell data objects, the target might also need to call **IDataObject::SetData**. For a discussion of how targets should respond for different data transfer scenarios, see *Handling Shell Data Transfer Scenarios*.

The following sections briefly discuss how to implement the **DragEnter**, **DragOver**, and **Drop** methods. For further details, see the reference documentation.

The DragEnter method

The system calls the **IDropTarget::DragEnter** method when the cursor enters the target window. Its parameters provide the target with the location of the cursor, the state of keyboard modifier keys such as the CTRL key, and a pointer to the data object's **IDataObject** interface. The target is responsible for using that interface to determine whether it can accept any of the formats contained by the data object. If it can, it normally leaves the value of *pdwEffect* unchanged. If it cannot accept any data from the data object, it sets the *pdwEffect* parameter to DROPEFFECT_NONE. The system passes the value of this parameter to the data object's **IDropSource** interface to allow it to display the appropriate drag image.

Targets should not use the **IDataObject::GetData** method to render shell data before it has been dropped. Fully rendering the object's data for each such occurrence might cause the drag cursor to stall. To avoid this problem, some shell objects contain a **CFSTR_INDRAGLOOP** format. By extracting this format, targets can check the status of the drag loop while avoiding expensive rendering of the object's data. The format's data value is a DWORD that is set to a nonzero value if the data object is within a drag loop. The format's data value is set to zero if the data has been dropped.

If the target can accept data from the data object, it should examine *grfKeyState* to determine whether any modifier keys have been pressed to modify the normal drop behavior. For instance, the default operation is typically a move, but depressing the CTRL key usually indicates a copy operation.

While the cursor is over the target window, the target can use the drag-drop helper object to replace the data object's drag image with its own. If so, **DragEnter** should call **IDropTargetHelper::DragEnter** to pass the information contained in the **DragEnter** parameters to the drag-drop helper object.

The DragOver method

As the cursor moves within the target window, the system periodically calls the **IDropTarget::DragOver** method. Its parameters provide the target with the location of the cursor and the state of keyboard modifier keys such as the CTRL key. **DragOver** has much the same responsibilities as **DragEnter**, and the implementations are usually very similar.

If the target is using the drag-drop helper object, **DragOver** should call **IDropTargetHelper::DragOver** to forward the information contained in the **DragOver** parameters to the drag-drop helper object.

The Drop method

The system calls the **IDropTarget::Drop** method to notify the target that the user has dropped the data, typically by releasing the mouse button. **Drop** has the same parameters as **DragEnter**. The target normally responds by extracting one or more formats from the data object. When finished, the target should set the *pdwEffect* parameter to a **DROPEFFECT** value that indicates the outcome of the operation. For some types of shell data transfer the target must also call **IDataObject::SetData** to pass a format with additional information on the outcome of the operation to the data object. For a detailed discussion, see *Handling Shell Data Transfer Scenarios*.

If the target is using the drag-drop helper object, **Drop** should call **IDropTargetHelper::Drop** to forward the information contained in the **DragOver** parameters to the drag-drop helper object.

Using the Drag-Drop Helper Object

The drag-drop helper object (CLSID_DragDropHelper) is exported by the shell to allow targets to specify the drag image while it is over the target window. To use the drag-drop

helper object, create an in-process server object by calling **CoCreateInstance** with a CLSID of CLSID_DragDropHelper. The drag-drop helper object exposes two interfaces that are used in the following way:

- The **IDragSourceHelper** interface allows the drop target to specify an icon to represent the data object.
- The **IDropTargetHelper** interface allows the drop target to inform the drag-drop helper object of the cursor location, and to show or hide the data icon.

Using the IDragSourceHelper Interface

The **IDragSourceHelper** interface is exposed by the drag-drop helper object to allow a drop target to provide the image that will be displayed while the cursor is over the target window. **IDragSourceHelper** provides two alternative ways to specify the bitmap to be used as a drag image:

- Drop targets that have a window can register a DI_GETDRAGIMAGE window message for it by initializing the drag-drop helper object with

IDragSourceHelper::InitializeFromWindow. When the target receives a DI_GETDRAGIMAGE message, the handler puts the drag image bitmap information in the **SHDRAGIMAGE** structure that is passed as the message's *lParam* value.

- Windowless drop targets specify a bitmap when they initialize the drag-drop helper object with **IDragSourceHelper::InitializeFromBitmap**.

Using the IDropTargetHelper Interface

This interface allows the drop target to notify the drag-drop helper object when the cursor enters or leaves the target. While the cursor is over the target window, **IDropTargetHelper** allows the target to give the drag-drop helper object the information that the target receives through its **IDropTarget** interface.

Four of the **IDropTargetHelper** methods, **IDropTargetHelper::DragEnter**, **IDropTargetHelper::DragLeave**, **IDropTargetHelper::DragOver**, and **IDropTargetHelper::Drop**, are associated with the **IDropTarget** method of the same name. To use the drag-drop helper object, each of the **IDropTarget** methods should call the corresponding **IDropTargetHelper** method to forward the information to the drag-drop helper object. The fifth **IDropTargetHelper** method, **IDropTargetHelper::Show**, notifies the drag-drop helper object to show or hide the drag image. This method is used when dragging over a target window in a low color-depth video mode. It allows the target to hide the drag image while it is painting the window.

Shell Clipboard Formats

The shell clipboard formats are used to identify the type of shell data being transferred. Most shell clipboard formats identify a type of data, such as a list of file names or PIDLs. However, some formats are used for communication between source and target. They can expedite the data transfer process by supporting shell operations such as *optimized move* and *delete-on-paste*. Shell data is always contained in a data object that uses a

FORMATETC structure as a more general way to characterize data. The structure's **cfFormat** member is set to the clipboard format for the particular item of data. The other members provide additional information, such as the data transfer mechanism. The data is contained in an accompanying **STGMEDIUM** structure.

Note Standard clipboard format identifiers have the form CF_*XXX*. A common example is CF_TEXT, which is used for transferring ANSI text data. These identifiers have predefined values and can be used directly with **FORMATETC** structures. With the exception of **CF_HDROP**, shell format identifiers have the form CFSTR_*XXX* and are not predefined. For simplicity, the CFSTR_*XXX* values are often referred to as *formats*. However, unlike predefined formats, they must be registered by both source and target before they can be used to transfer data. To register a shell format, include the Shlobj.h header file and pass the CFSTR_*XXX* format identifier to **RegisterClipBoardFormat**. This function returns a valid clipboard format value, which can then be used as the **cfFormat** member of a **FORMATETC** structure.

The shell clipboard formats are organized here into three groups, based on how they are used:

- Formats for Transferring File System Objects
- Formats for Transferring Virtual Objects
- Formats for Communication Between Source and Target

Formats for Transferring File System Objects

These formats are used to transfer one or more files or other shell objects.

- **CF_HDROP**
- **CFSTR_FILECONTENTS**
- **CFSTR_FILEDESCRIPTOR**
- **CFSTR_FILENAME**
- **CFSTR_FILENAMEMAP**
- **CFSTR_MOUNTEDVOLUME**
- **CFSTR_SHELLIDLIST**
- **CFSTR_SHELLIDLISTOFFSET**

CF_HDROP

This clipboard format is used when transferring the locations of a group of existing files. Unlike the other shell formats, it is predefined, so there is no need to call **RegisterClipBoardFormat**. The data consists of an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a **DROPFILES** structure as its **hGlobal** member.

The **pFiles** member of the **DROPFILES** structure contains an offset to a double NULL-terminated character array containing the file names. If you are extracting a CF_HDROP format from a data object, you can use **DragQueryFile** to extract individual file names from the global memory object. If you are creating a CF_HDROP format to place in a data object, you will need to construct the file name array.

The file name array consists of a series of strings, each containing one file's fully qualified path, including the terminating NULL character. An additional NULL character is appended to the final string to terminate the array. For example, if the files c:\temp1.txt and c:\temp2.txt are being transferred, the character array looks like this:

```
c:\temp1.txt'\0'c:\temp2.txt'\0''\0'
```

Note In this example, '\0' is used to represent the NULL character, not the literal characters that should be included.

If the object was copied to the clipboard as part of a drag-drop operation, the **pt** member of the **DROPFILES** structure contains the coordinates of the point where the object was dropped. You can use **DragQueryPoint** to extract the cursor coordinates.

If this format is present in a data object, an OLE drag loop simulates **WM_DROPFILES** functionality with non-OLE drop targets. This is important if your application is the source of a drag-drop operation on a Microsoft Windows 3.1 system.

Note This format supports both ANSI and Unicode. However, only ANSI file paths can be used with Windows 95 systems.

CFSTR_FILECONTENTS

This format identifier is used with the **CFSTR_FILEDESCRIPTOR** format to transfer data as if it were a file, regardless of how it is actually stored. The data consists of an **STGMEDIUM** structure that represents the contents of one file. The file is normally represented as a stream object, which avoids having to place the contents of the file in memory. In that case, the **tymed** member of the **STGMEDIUM** structure is set to TYMED_ISTREAM, and the file is represented by an **IStream** interface. The file can also be a storage or global memory object (TYMED_ISTORAGE or TYMED_HGLOBAL). The associated CFSTR_FILEDESCRIPTOR format contains a **FILEDESCRIPTOR** structure for each file that specifies the file's name and attributes.

The target treats the data associated with a CFSTR_FILECONTENTS format as if it were a file. When the target calls **IDataObject::GetData** to extract the data, it specifies a particular file by setting the **lindex** member of the **FORMATETC** structure to the zero-based index of the file's **FILEDESCRIPTOR** structure in the accompanying **CFSTR_FILEDESCRIPTOR** format. The target then uses the returned interface pointer or global memory handle to extract the data.

CFSTR_FILEDESCRIPTOR

This format identifier is used with the **CFSTR_FILECONTENTS** format to transfer data as a group of files. These two formats are the preferred way to transfer shell objects that are not stored as file-system files. For example, these formats can be used to transfer a group of e-mail messages as individual files, even though each e-mail is actually stored as a block of data in a database. The data consists of an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a **FILEGROUPDESCRIPTOR** structure that is followed by an array containing one **FILEDESCRIPTOR** structure for each file in the group. For each **FILEDESCRIPTOR** structure, there is a separate **CFSTR_FILECONTENTS** format that contains the contents of the file. To identify a particular file's CFSTR_FILECONTENTS format, set the **lIndex** value of the FORMATETC structure to the zero-based index of the file's **FILEDESCRIPTOR** structure.

The **CFSTR_FILEDESCRIPTOR** format is commonly used to transfer data as if it were a group of files, regardless of how it is actually stored. From the target's perspective, each **CFSTR_FILECONTENTS** format represents a single file and is treated accordingly. However, the source can store the data in any way it chooses. While a **CSFTR_FILECONTENTS** format might correspond to a single file, it could also, for example, represent data extracted by the source from a database or text document.

CFSTR_FILENAME

This format identifier is used to transfer a single file. The data consists of an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a single null-terminated string containing the file's fully qualified file path. This format has been superseded by **CF_HDROP**, but it is supported for backward compatibility with Windows 3.1 applications.

CFSTR_FILENAMEMAP

This format identifier is used when a group of files in **CF_HDROP** format is being renamed as well as transferred. The data consists of an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a double null-terminated character array. This array contains a new name for each file, in the same order that the files are listed in the accompanying CF_HDROP format. The format of the character array is the same as that used by CF_HDROP to list the transferred files.

CFSTR_MOUNTEDVOLUME

This format identifier is used to transfer a path on a mounted volume. It is similar to CF_HDROP, but it contains only a single path and can handle the longer path strings that might be needed to represent a path when the volume is mounted on a folder. The data consists of an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a single null-terminated string containing the fully qualified file path. The path string must end with a '\' character, followed by the terminating NULL.

Prior to Microsoft Windows 2000, volumes could be mounted only on drive letters. For Windows 2000 and later systems with an NTFS formatted drive, you can also mount volumes on empty folders. This feature allows a volume to be mounted without taking up a drive letter. The mounted volume can use any currently supported format, including FAT, FAT32, NTFS, and CDFS.

You can add pages to a Drive Properties property sheet by implementing a property sheet handler. If the volume is mounted on a drive letter, the shell passes path information to the handler with the **CF_HDROP** format. With Windows 2000 and later systems, the CF_HDROP format is used when a volume is mounted on a drive letter, just as with earlier systems. However, if a volume is mounted on a folder, the CSFTR_MOUNTEDVOLUME format identifier is used instead of CF_HDROP.

If only drive letters will be used to mount volumes, only CF_HDROP will be used, and existing property sheet handlers will work as they did with earlier systems. However, if you want your handler to display a page for volumes that are mounted on folders as well as drive letters, the handler must be able to understand both the CSFTR_MOUNTEDVOLUME and CF_HDROP formats.

CFSTR_SHELLIDLIST

This format identifier is used when transferring the locations of one or more existing namespace objects. It is used in much the same way as **CF_HDROP**, but it contains **PIDLs** instead of file system paths. Using PIDLS allows the CFSTR_SHELLIDLIST format to handle virtual objects as well as file system objects. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a **CIDA** structure.

The **aoffset** member of the **CIDA** structure is an array containing offsets to the beginning of the **ITEMIDLIST** structure for each PIDL that is being transferred. To extract a particular PIDL, first determine its index. Then, add the **aoffset** value that corresponds to that index to the address of the **CIDA** structure.

The first element of **aoffset** contains an offset to the fully qualified PIDL of a parent folder. If this PIDL is empty, the parent folder is the desktop. Each of the remaining elements of the array contains an offset to one of the PIDLs to be transferred. All of these PIDLs are relative to the PIDL of the parent folder.

The following two macros can be used to retrieve PIDLs from a **CIDA** structure. The first takes a pointer to the structure and retrieves the PIDL of the parent folder. The second takes a pointer to the structure and retrieves one of the other PIDLs, identified by its zero-based index.

```
#define GetPIDLFolder(pida) (LPCITEMIDLIST)(((LPBYTE)pida)
+(pida)->aoffset[0])
#define GetPIDLItem(pida, i) (LPCITEMIDLIST)(((LPBYTE)pida)
+(pida)->aoffset[i+1])
```

Note The value that is returned by these macros is a pointer to the PIDL's **ITEMIDLIST** structure. Since these structures vary in length, you must determine the end of the structure by walking through each of the **ITEMIDLIST** structure's **SHITEMID** structures until you reach the two-byte NULL that marks the end. For further discussion of PIDLs and the **ITEMIDLIST** structure, see *The Shell Namespace*.

CFSTR_SHELLIDLISTOFFSET

This format identifier is used with formats such as **CF_HDROP**, **CFSTR_SHELLIDLIST**, and **CFSTR_FILECONTENTS** to specify the position of a group of objects following a transfer. The data consists of an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to an array of **POINT** structures. The first structure specifies the screen coordinates, in pixels, of the upper-left corner of the rectangle that encloses the group. The remainder of the structures specify the locations of the individual objects relative to the group's position. They must be in the same order as that used to list the objects in the associated format.

Formats for Transferring Virtual Objects

The **CFSTR_SHELLIDLIST** format can be used to transfer both file system and virtual objects. However, there are also several specialized formats for transferring particular types of virtual objects.

- **CFSTR_NETRESOURCES**
- **CFSTR_PRINTERGROUP**
- **CFSTR_SHELLURL**

CFSTR_NETRESOURCES

This format identifier is used when transferring network resources, such as a domain or server. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a single NULL-terminated string that identifies the network resource. The drop target can then use the data with any of the **WNet API** functions, such as **WNetAddConnection**, to perform network operations on the object.

CFSTR_PRINTERGROUP

This format identifier is used when transferring the friendly names of printers. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a string in the same format as that used with **CF_HDROP**. However, the **pFiles** member of the **DROPFILES** structure contains one or more friendly names of printers instead of file paths.

CFSTR_SHELLURL

This format identifier is used when transferring a single URL. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a single NULL-terminated string that contains the URL. This format is equivalent to the CF_TEXT clipboard format, but it is useful for internet-related applications.

Formats for Communication Between Source and Target

Several format identifiers were introduced with Internet Explorer 4.0 to allow communication between source and target. These formats accompany the actual data and give applications a greater degree of control over move-copy-paste or drag-drop operations involving shell objects.

- CFSTR_INDRAGLOOP
- CFSTR_LOGICALPERFORMEDDROPEFFECT
- CFSTR_PASTESUCCEEDED
- CFSTR_PERFORMEDDROPEFFECT
- CFSTR_PREFERREDDROPEFFECT
- CFSTR_TARGETCLSID

CFSTR_INDRAGLOOP

This format identifier is used by a data object to indicate whether it is in a drag-drop loop. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a DWORD value. If the DWORD value is nonzero, the data object is within a drag-drop loop. If the value is set to zero, the data object is not within a drag-drop loop.

Some drop targets might call **IDataObject::GetData** and attempt to extract data while the object is still within the drag-drop loop. Fully rendering the object for each such occurrence might cause the drag cursor to stall. If the data object supports CFSTR_INDRAGLOOP, the target can instead use that format to check the status of the drag-drop loop and avoid expensive rendering of the object until it is actually dropped. The expensive-to-render formats should still be included in the **FORMATETC** enumerator and in calls to **IDataObject::QueryGetData**. If the data object does not set CFSTR_INDRAGLOOP, it should act as if the value is set to zero.

CFSTR_LOGICALPERFORMEDDROPEFFECT

Version 5.0. This format identifier allows a drop source to call the data object's **IDataObject::GetData** method to determine the outcome of a shell data transfer. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a DWORD containing a **DROPEFFECT** value.

The **CFSTR_PERFORMEDDROPEFFECT** format identifier was intended to allow the target to indicate to the data object what operation actually took place. However, the shell uses optimized moves for file system objects whenever possible. In that case, the shell normally sets the CFSTR_PERFORMEDDROPEFFECT value to DROPEFFECT_NONE, to indicate to the data object that the original data has been deleted. Thus, the source cannot use the CFSTR_PERFORMEDDROPEFFECT value to determine which operation has taken place. While most sources do not need this information, there are some exceptions. For instance, even though optimized moves eliminate the need for a source to delete any data, the source might still need to update a related database to indicate that the files have been moved or copied.

If a source needs to know which operation took place, it can call the data object's **IDataObject::GetData** method and request the CFSTR_LOGICALPERFORMEDDROPEFFECT format. This format essentially reflects what happens from the user's point of view after the operation is complete. If a new file is created and the original file is deleted, the user sees a move operation and the format's data value is set to DROPEFFECT_MOVE. If the original file is still there, the user sees a copy operation and the format's data value is set to DROPEFFECT_COPY. If a link was created, the format's data value will be DROPEFFECT_LINK.

CFSTR_PASTESUCCEEDED

This format identifier is used by the target to inform the data object, through its **IDataObject::SetData** method, that a delete-on-paste operation succeeded. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's

hGlobal member points to a DWORD containing a **DROPEFFECT** value. This format is used to notify the data object that it should complete the cut operation and delete the original data, if necessary. For more information, see *Delete-on-Paste Operations*.

CFSTR_PERFORMEDDROPEFFECT

This format identifier is used by the target to inform the data object through its **IDataObject::SetData** method of the outcome of a data transfer. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a DWORD set to the appropriate **DROPEFFECT** value, normally DROPEFFECT_MOVE or DROPEFFECT_COPY.

This format is normally used when the outcome of an operation can be either move or copy, such as in an optimized move or delete-on-paste operation. It provides a reliable way for the target to tell the data object what actually happened. It was introduced because the value of *pdwEffect* returned by **DoDragDrop** did not reliably indicate which operation had taken place. The CFSTR_PERFORMEDDROPEFFECT format is the reliable way to indicate that an unoptimized move has taken place.

CFSTR_PREFERREDDROPEFFECT

This format identifier is used by the source to specify whether its preferred method of data transfer is move or copy. A drop target requests this format by calling the data object's **IDataObject::GetData** method. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a DWORD value. This value is set to DROPEFFECT_MOVE if a move operation is preferred or DROPEFFECT_COPY if a copy operation is preferred.

This feature is used when a source can support either a move or copy operation. It uses the CFSTR_PREFERREDDROPEFFECT format to communicate its preference to the target. Because the target is not obligated to honor the request, the target must call the source's **IDataObject::SetData** method with a **CFSTR_PERFORMEDDROPEFFECT** format to tell the data object which operation was actually performed.

With a delete-on-paste operation, the CFSTR_PREFERREDDROPFORMAT format is used to tell the target whether the source did a cut or copy. With a drag-drop operation, you can use CFSTR_PREFERREDDROPFORMAT to specify the shell's action. If this format is not present, the shell performs a default action, based on context. For instance, if a user drags a file from one volume and drops it on another volume, the shell's default action is to copy the file. By including a CFSTR_PREFERREDDROPFORMAT format in the data object, you can override the default action and explicitly tell the shell to copy, move, or link the file. If the user chooses to drag with the right button, CFSTR_PREFERREDDROPFORMAT specifies the default command on the drag-drop context menu. The user is still free to choose other commands on the menu.

Before Internet Explorer 4.0, an application indicated that it was transferring shortcut file types by setting FD_LINKUI in the **dwFlags** member of the **FILEDESCRIPTOR** structure. Targets then had to use a potentially time-consuming call to **IDataObject::GetData** to find out if the FD_LINKUI flag was set. Now, the preferred way to indicate that shortcuts are being transferred is to use the CFSTR_PREFERREDDROPEFFECT format set to DROPEFFECT_LINK. However, for backward compatibility with older systems, sources should still set the FD_LINKUI flag.

CFSTR_TARGETCLSID

This format identifier is used by a target to provide its CLSID to the source. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to the CLSID GUID of the drop target.

This format is used primarily to allow objects to be deleted by dragging them to the Recycle Bin. When an object is dropped in the Recycle Bin, the source's **IDataObject::SetData** method is called with a CFSTR_TARGETCLSID format set to the Recycle Bin's CLSID (CLSID_RecycleBin). The source can then delete the original object.

Handling Shell Data Transfer Scenarios

The Shell Data Object document discussed the general approach that is used to transfer shell data with drag-drop or the Clipboard. However, to implement shell data transfer in your application, you must also understand how to apply these general principles and techniques to the variety of ways that shell data can be transferred. This document presents the following common shell data transfer scenarios and discusses how to implement each one in your application.

General Guidelines

Copying File Names from the Clipboard to an Application

Copying the Contents of a Dropped File into an Application

Handling Optimized Move Operations

Handling Delete-on-Paste Operations

Transfering Data to and from Virtual Folders

Dropping Files on the Recycle Bin

Creating and Importing Scrap Files

Extracting Data Asynchronously

Note Although each of these scenarios discusses a specific data transfer operation, many of them apply to a variety of related scenarios. For instance, the primary difference between most Clipboard and drag-drop transfers is in how the data object gets to the target. Once the target has a pointer to the data object's **IDataObject** interface, the procedures for extracting information are largely the same for both types of data transfer. However, some of the scenarios are limited to a specific type of operation. Refer to the individual scenario for details.

General Guidelines

Each of the following sections discusses a single, fairly specific data transfer scenario. However, data transfers are often more complex and might involve aspects of several scenarios. You typically do not know, in advance, which scenario you will actually need to handle. Here are a few general guidelines to keep in mind.

For data sources:

- The shell clipboard formats, with the exception of **CF_HDROP**, are not predefined. Each format you want to use must be registered by calling **RegisterClipboardFormat**.
- Include as many formats as you can support. You generally don't know where the data object will be dropped. This practice improves the odds that the data object will contain a format that the drop target can accept.
- Existing files should be offered with the CF_HDROP format.
- Offer file-like data with **CFSTR_FILECONTENTS/ CFSTR_FILEDESCRIPTOR** formats. This approach allows the target to create a file from a data object without needing to know anything about the underlying data storage. You should normally present the data as an **IStream** interface. This data transfer mechanism is more flexible than a global memory object and uses much less memory.
- Use standard feedback cursors.
- Support left and right drag.
- Use the data object itself from an embedded object. This approach allows your application to get any extra formats the data object has to offer and avoids creating an extra layer of containment. For instance, an embedded object from server A is dragged from server/container B and dropped on container C. C should create an embedded object of server A, not an embedded object of server B containing an embedded object of server A.
- Remember that the shell might use optimized moves or delete-on-paste operations when moving files. Your application should be able to recognize these operations and respond appropriately.

For data targets:

- The shell clipboard formats, with the exception of **CF_HDROP**, are not predefined. Each format you want to use must be registered by calling **RegisterClipboardFormat**.
- Implement and register an OLE drop target. Avoid using Microsoft Windows 3.1 targets or the **WM_DROPFILES** message, if possible.
- The formats contained by a data object vary, depending on where the object comes from. Since you generally don't know in advance where a data object comes from, don't assume that a particular format will be present. Enumerate the formats, and then select the best available format for your purposes.

- When a file is dragged from the desktop or Windows Explorer and dropped on an application's client area, the target application should first attempt to get a data format that it can handle directly, such as CF_TEXT or CF_EMBEDDEDOBJECT. If none of these are offered, the target should request CF_HDROP and call **OleCreateFromFile** for each file. Scrap files will always offer CF_EMBEDDEDOBJECT and CF_HDROP formats.
- Support right-drag. You can customize the drag context menu by creating a drag-drop handler
- If your application will accept existing files, it must be able to handle the **CF_HDROP** format.
- In general, applications that accept files should also handle the **CFSTR_FILECONTENTS/CFSTR_FILEDESCRIPTOR** formats. While files from the file system have the CF_HDROP format, files from providers such as namespace extensions generally use CFSTR_FILECONTENTS/ CFSTR_FILEDESCRIPTOR. Examples include Windows CE folders, FTP folders, Web folders, and CAB folders. The source normally implements an **IStream** interface to present data from their storage as a file.
- Remember that the shell might use optimized moves or delete-on-paste operations when moving files. Your application should be able to recognize these operations and respond appropriately.

Copying File Names from the Clipboard to an Application

Scenario: A user selects one or more files in Windows Explorer and copies them to the Clipboard. Your application extracts the file names and pastes them into the document.

This scenario could be used, for instance, to allow a user to create an HTML link by cutting and pasting the file to your application. Your application can then extract the file name from the data object and process it to create an anchor tag.

When a user selects a file in Windows Explorer and copies it to the Clipboard, the shell creates a data object. It then calls **OleSetClipboard** to place a pointer to the data object's **IDataObject** interface on the Clipboard.

When the user selects the Paste command from your application's menu or toolbar:

1. Call **OleGetClipboard** to retrieve the data object's IDataObject interface.
2. Call **IDataObject::EnumFormatEtc** to request an enumerator object.
3. Use the enumerator object's **IEnumFORMATETC** interface to enumerate the formats contained by the data object.

Extracting the File Names from the Data Object

The next step is to extract one or more file names from the data object and paste them into your application. Note that the procedure discussed in this section for extracting a file name from a data object applies equally well to drag-drop transfers.

The simplest way to get file names from a data object is the **CF_HDROP** format:

1. Call **IDataObject::GetData**. Set the **cfFormat** member of the **FORMATETC** structure to CF_HDROP and the **tymed** member to TYMED_HGLOBAL. The **dwAspect** member is normally set to DVASPECT_CONTENT. However, if you need to have the file's path in short (8.3) format, set **dwAspect** to DVASPECT_SHORT.
2. When **IDataObject::GetData** returns, the **hGlobal** member of the **STGMEDIUM** structure points to a global memory object that contains the data.
3. Create an HDROP variable and set it to the **hGlobal** member of the **STGMEDIUM** structure. The HDROP variable now points to a **DROPFILES** structure followed by a double NULL-terminated string containing the fully qualified file paths of the copied files.
4. Determine how many file paths are in the list by calling **DragQueryFile** with the *iFile* parameter set to 0xFFFFFFFF. The function returns the number of file paths in the list. The file path's zero-based index in this list is used in the next step to identify a particular path.
5. Extract the file paths from the global memory object by calling **DragQueryFile** once for each file, with *iFile* set to the file's index.
6. Process the file paths as needed and paste them into your application.

Copying the Contents of a Dropped File into an Application

Scenario: A user drags one or more files from Windows Explorer and drops them on your application's window. Your application extracts the content of the file and pastes it into the application.

This scenario uses drag-drop to transfer the files from Windows Explorer to the application. Prior to the operation, your application must:

1. Call **RegisterClipboard** to register any needed shell clipboard formats.
2. Call **RegisterDragDrop** to register a target window and your application's **IDropTarget** interface.

After the user initiates the operation by selecting one or more files and starting to drag them:

1. Windows Explorer creates a data object and loads the supported formats into it.
2. Windows Explorer calls **DoDragDrop** to initiate the drag loop.
3. When the drag image reaches your target window, the system notifies you by calling **IDropTarget::DragEnter**.
4. To determine what the data object contains, call the data object's **IDataObject::EnumFormatEtc** method. Use the enumerator object returned by the method to enumerate the formats contained by the data object. If your application does not want to accept any of these formats, return DROPEFFECT_NONE. For the purposes of this scenario, your application should ignore any data objects that do not contain formats used to transfer files, such as **CF_HDROP**.

5. When the user drops the data, the system calls **IDropTarget::Drop**.
6. Use the **IDataObject** interface to extract the contents of the files.

There are several different ways to extract the contents of a shell object from a data object. In general, you should attempt them in the following order:

- If the file contains a CF_TEXT format, the data is ANSI text. You can use the CF_TEXT format to extract the data, rather than opening the file itself.
- If the file contains a linked or embedded OLE object, the data object contains a CF_EMBEDDEDOBJECT format. Use standard OLE techniques to extract the data. Scrap files always contain a CF_EMBEDDEDOBJECT format.
- If the shell object is from the file system, the data object contains a **CF_HDROP** format with the names of the files. Extract the file name from CF_HDROP and call **OleCreateFromFile** to create a new linked or embedded object. For a discussion of how to retrieve a file name from a CF_HDROP format, see *Copying File Names from the Clipboard to an Application*.
- If the data object contains a **CFSTR_FILEDESCRIPTOR** format, you can extract a file's contents from the file's **CFSTR_FILECONTENTS** format. For a discussion of this procedure, see *Using the CFSTR_FILECONTENTS Format to Extract Data from a File*.
- Prior to shell version 4.71, an application indicated that it was transferring a shortcut file type by setting FD_LINKUI in the **dwFlags** member of the **FILEDESCRIPTOR** structure. For later versions of the shell, the preferred way to indicate that shortcuts are being transferred is to use the CFSTR_PREFERREDDROPEFFECT format set to DROPEFFECT_LINK. This approach is much more efficient than extracting the **FILEDESCRIPTOR** structure just to check a flag.

If the data extraction process will be lengthy, you might want to do the operation asynchronously on a background thread. Your primary thread can then proceed without unnecessary delays. For a discussion of how to handle asynchronous data extraction, see *Extracting Data Asynchronously*.

Using the CFSTR_FILECONTENTS Format to Extract Data from a File

The CFSTR_FILECONTENTS format provides a very flexible and powerful way to transfer the contents of a file. It is not even necessary for the data to be stored as a single file. All that is required for this format is that the data object present the data to the target as if it were a file. For instance, the actual data might be a section of a text document or a block of data extracted from a database. The target can treat the data as a file and doesn't need to know anything about the underlying storage mechanism.

Namespace extensions normally use CFSTR_FILECONTENTS to transfer data because it does not assume any particular storage mechanism. A namespace extension can use whatever storage mechanism is convenient, and use this format to present its objects to applications as if they were files.

The data transfer mechanism for CFSTR_FILECONTENTS is normally TYMED_ISTREAM. Transferring an **IStream** interface pointer requires much less memory than loading the data into a global memory object, and **IStream** is a more flexible way to represent data than **IStorage**.

A CFSTR_FILECONTENTS format is always accompanied by a CFSTR_FILEDESCRIPTOR format. You must examine the contents of this format first. If more than one file is being transferred, the data object will actually contain multiple CFSTR_FILECONTENTS formats, one for each file. The CFSTR_FILEDESCRIPTOR format contains the name and attributes of each file, and provides an index value for each file that is needed to extract a particular file's CFSTR_FILECONTENTS format.

To extract a **CFSTR_FILECONTENTS** format:

1. Extract the CFSTR_FILEDESCRIPTOR format. The **hGlobal** member of the returned **STGMEDIUM** structure points to a global memory object containing a **FILEGROUPDESCRIPTOR** structure followed by one or more **FILEDESCRIPTOR** structures. Each **FILEDESCRIPTOR** structure contains a description of a file that is contained by one of the accompanying **CFSTR_FILECONTENTS** formats.
2. Examine the **FILEDESCRIPTOR** structures to determine which one corresponds to the file you want to extract. The zero-based index of that **FILEDESCRIPTOR** structure is used to identify the file's CFSTR_FILECONTENTS format.
3. Call **IDataObject::GetData** with the **cfFormat** member of the **FORMATETC** structure set to the CFSTR_FILECONTENTS value and the **lIndex** member set to the index that you determined in the previous step. The **tymed** member is typically set to TYMED_HGLOBAL | TYMED_ISTREAM | TYMED_ISTORAGE. The data object can then choose its preferred data transfer mechanism.
4. The **STGMEDIUM** structure that **IDataObject::GetData** returns will contain a pointer to the file's data. Examine the **tymed** member of the structure to determine the data transfer mechanism.
5. If **tymed** is set to TYMED_ISTREAM or TYMED_ISTORAGE, use the interface to extract the data. If **tymed** is set to TYMED_HGLOBAL, the data is contained in a global memory object. For a discussion of how to extract data from a global memory object, see *How to Extract a Global Memory Object from a Data Object*.

Handling Optimized Move Operations

Scenario: A file is moved from the file system to a namespace extension using an optimized move.

In a conventional move operation, the target makes a copy of the data and the source deletes the original. This procedure can be inefficient because it requires two copies of the data. With large objects such as databases, a conventional move operation might not even be practical.

With an optimized move, the target uses its understanding of how the data is stored to handle the entire move operation. There is never a second copy of the data, and there is no need for the source to delete the original data. Shell data is well suited to optimized moves because the target can handle the entire operation using the shell API. A typical example is moving files. Once the target has the path of a file to be moved, it can use **SHFileOperation** to move it. There is no need for the source to delete the original file.

Note The shell normally uses an optimized move to move files. To handle shell data transfer properly, your application must be capable of detecting and handling an optimized move.

Optimized moves are handled in the following way:

1. The source calls **DoDragDrop** with the *dwEffect* parameter set to DROPEFFECT_MOVE to indicate that the source objects can be moved.
2. The target receives the DROPEFFECT_MOVE value through one of its **IDropTarget** methods, indicating that a move is allowed.
3. The target either copies the object (unoptimized move) or moves the object (optimized move).
4. The target then tells the source whether or not it needs to delete the original data.

An optimized move is the default operation, with the data deleted by the target. To inform the source that an optimized move was performed:

- The target sets the **pdwEffect* value it received through its **IDropTarget::Drop** method to some value other than DROPEFFECT_MOVE. It is typically set to either DROPEFFECT_NONE or DROPEFFECT_COPY. The value will be returned to the source by **DoDragDrop**.
- The target also calls the data object's **IDataObject::SetData** method and passes it a **CFSTR_PERFORMEDDROPEFFECT** format identifier set to DROPEFFECT_NONE. This method call is necessary because some drop targets might not set the *pdwEffect* parameter of **DoDragDrop** properly. The CFSTR_PERFORMEDDROPEFFECT format is the reliable way to indicate that an optimized move has taken place.

If the target did an unoptimized move, the data must be deleted by the source. To inform the source that an unoptimized move was performed:

- The target sets the *pdwEffect* value it received through its **IDropTarget::Drop** method to DROPEFFECT_MOVE. The value will be returned to the source by **DoDragDrop**.
- The target also calls the data object's **IDataObject::SetData** method and passes it a **CFSTR_PERFORMEDDROPEFFECT** format identifier set to DROPEFFECT_MOVE. This method call is necessary because some drop targets might not set the *pdwEffect* parameter of **DoDragDrop** properly. The CFSTR_PERFORMEDDROPEFFECT format is the reliable way to indicate that an unoptimized move has taken place.
    - The source inspects the two values that can be returned by the target. If both are set to DROPEFFECT_MOVE, it completes the unoptimized move by deleting the original data. Otherwise, the target did an optimized move and the original data has been deleted.

Handling Delete-on-Paste Operations

Scenario: One or more files are cut from a folder in Windows Explorer and pasted into a namespace extension. Windows Explorer leaves the files highlighted until it receives feedback on the outcome of the paste operation.

Traditionally, when a user cuts data it immediately disappears from view. This might not be efficient, and it can lead to usability problems if the user becomes concerned about what has happened to the data. An alternative approach is to use a delete-on-paste operation.

With a delete-on-paste operation, the selected data is not immediately removed from view. Instead, the source application marks it as selected, perhaps by changing the font or background color. After the target application has pasted the data, it notifies the source about the outcome of the operation. If the target performed an optimized move, the source can simply update its display. If the target performed a normal move, the source must also delete its copy of the data. If the paste fails, the source application restores the selected data to its original appearance.

Note The shell normally uses delete-on-paste when a cut/paste operation is used to move files. Delete-on-paste operations with shell objects normally use an optimized move to move the files. To handle shell data transfer properly, your application must be capable of detecting and handling delete-on-paste operations.

The essential requirement for delete-on-paste is that the target must report the outcome of the operation to the source. However, standard clipboard techniques cannot be used to implement delete-on-paste because they do not provide a way for the target to communicate with the source. Instead, the target application uses the data object's

IDataObject::SetData method to report the outcome to the data object. The data object can then communicate with the source through a private interface.

The basic procedure for a delete-on-paste operation is as follows:

1. The source marks the screen display of the selected data.
2. The source creates a data object. It indicates a cut operation by adding the CFSTR_PREFERREDDROPEFFECT format with a data value of DROPEFFECT_MOVE.
3. The source places the data object on the Clipboard using OleSetClipboard.
4. The target retrieves the data object from the Clipboard using OleGetClipboard.
5. The target extracts the CFSTR_PREFERREDDROPEFFECT data. If it is set to only DROPEFFECT_MOVE, the target can either do an optimized move or simply copy the data.
6. If the target does not do an optimized move, it calls the IDataObject::SetData method with the CFSTR_PERFORMEDDROPEFFECT format set to DROPEFFECT_MOVE.
7. When the paste is complete, the target calls the IDataObject::SetData method with the CFSTR_PASTESUCCEEDED format set to DROPEFFECT_MOVE.

8. When the source's IDataObject::SetData method is called with the CFSTR_PASTESUCCEEDED format set to DROPEFFECT_MOVE, it must check to see if it also received the CFSTR_PERFORMEDDROPEFFECT format set to DROPEFFECT_MOVE. If both formats are sent by the target, the source will have to delete the data. If only the CFSTR_PASTESUCCEEDED format is received, the source can simply remove the data from its display. If the transfer fails, the source updates the display to its original appearance.

Transfering Data to and from Virtual Folders

Scenario: A user drags an object from or drops it on a virtual folder.

Virtual folders contain objects that are generally not part of the file system. Some virtual folders, such as the Recycle Bin, can represent data that is stored on the hard drive but not as ordinary file system objects. Some can represent stored data that is on a remote system, such as a hand-held PC, or an FTP site. Others, such as the Printers folder, contain objects that do not represent stored data at all. While some virtual folders are part of the system, developers can also create and install custom virtual folders by implementing a namespace extension.

Regardless of the type of data or how it is stored, the folder and file objects that are contained by a virtual folder are presented by the shell as if they were normal files and folders. It is the responsibility of the virtual folder to take whatever data it contains and present it to the shell appropriately. This requirement means that virtual folders normally support drag-drop and clipboard data transfers.

There are thus two groups of developers who need to be concerned with data transfer to and from virtual folders:

- Developers whose applications need to accept data that is transferred from a virtual folder.
- Developers whose namespace extensions need to properly support data transfer.

Accepting Data from a Virtual Folder

Virtual folders can represent virtually any type of data and can store that data in any way they choose. Some virtual folders might actually contain normal file system files and folders. Others might, for instance, pack all their objects into a single document or database.

When a file system object is transferred to an application, the data object normally contains a **CF_HDROP** format with the object's fully qualified path. Your application can extract this string, and use the normal file system functions to open the file and extract its data. However, because virtual folders typically do not contain normal file system objects, they generally do not use CF_HDROP.

Instead of **CF_HDROP**, data is normally transferred from virtual folders with the **CFSTR_FILEDESCRIPTOR/CFSTR_FILECONTENTS** formats. The CFSTR_FILECONTENTS format has two advantages over CF_HDROP:

- No particular method of data storage is assumed.
- The format is more flexible. It supports three data transfer mechanisms: a global memory object, an **IStream** interface, or an **IStorage** interface.

Global memory objects are rarely used to transfer data to or from virtual objects because the data must be copied into memory in its entirety. Transferring an interface pointer requires almost no memory and is much more efficient. With very large files, an interface pointer might be the only practical data transfer mechanism. Typically, data is represented by an **IStream** pointer, because that interface is somewhat more flexible than **IStorage**. The target extracts the pointer from the data object and uses the interface methods to extract the data.

For further discussion of how to handle the CFSTR_FILEDESCRIPTOR/ CFSTR_FILECONTENTS formats, see *Using the CFSTR_FILECONTENTS Format to Extract Data from a File*.

Transferring Data to and from a NameSpace Extension

When you implement a namespace extension, you will normally want to support drag-drop capabilities. Follow the recommendations for drop sources and targets discussed in *General Guidelines*. In particular, a namespace extension must:

- Be able to handle the CFSTR_FILEDESCRIPTOR/ CFSTR_FILECONTENTS formats. These two formats are normally used to transfer objects to and from namespace extensions.
- Be able to handle optimized moves. The shell expects that shell objects will be moved with an optimized move.
- Be able to handle a delete-on-paste operation. The shell uses delete-on-paste when objects are moved from the shell with a cut/paste operation.
- Be able to handle data transfer through an **IStream** or **Istorage** interface. Data transfer to or from a virtual folder is normally handled by transferring one of these two interface pointers, typically an **IStream** pointer. The target then calls the interface methods to extract the data.
  - As a drop source, the namespace extension must extract the data from storage and pass it through this interface to the target.
  - As a drop target, a namespace extension must accept data from a source through this interface and store it properly.

For a more thorough discussion of how to implement a namespace extension, see *Namespace Extensions*.

Dropping Files on the Recycle Bin

Scenario: The user drops a file on the Recycle Bin. Your application or namespace extension deletes the original file.

The Recycle Bin is a virtual folder that is used as a repository for files that are no longer needed. As long as the Recycle Bin has not been emptied, the user can later recover the file and return it to the file system.

For the most part, transferring shell objects to the Recycle Bin works much like any other folder. However, when a user drops a file on the Recycle Bin, the source needs to delete the original, even if the feedback from the folder indicates a copy operation. Normally, a drop source has no way of knowing which folder its data object has been dropped on. However, for Windows 2000 and later systems, when a data object is dropped on the Recycle Bin, the shell will call the data object's **IDataObject::SetData** method with a **CFSTR_TARGETCLSID** format set to the Recycle Bin's CLSID (CLSID_RecycleBin). To handle the Recycle Bin case properly, your data object should be able to recognize this format and communicate the information to the source through a private interface.

Creating and Importing Scrap Files

Scenario: A user drags some data from an OLE application's data file and drops it on the desktop or Windows Explorer.

Windows allows users to drag an object from an OLE application's data file and drop it on the desktop or a file system folder. This operation creates a *scrap file*, which contains the data or a link to the data. The file name is taken from the short name registered for the CLSID of the object and the CF_TEXT data. For the shell to create a scrap file containing data, the application's **IDataObject** interface must support the CF_EMBEDSOURCE clipboard format. To create a file containing a link, **IDataObject** must support the CF_LINKSOURCE format.

There are also three optional features that an application can implement to support scrap files:

- Round-trip support
- Cached data formats
- Delayed rendering

Round-trip support

A *round trip* involves transferring a data object to another container and then back to the original document. For instance, a user could transfer a group of cells from a spreadsheet to the desktop, creating a scrap file with the data. If the user then transfers the scrap back to the spreadsheet, the data needs to be integrated into the document as it was before the original transfer.

When the shell creates the scrap file, it represents the data as an embedding object. When the scrap is transferred to another container, it is transferred as an embedding object, even if it is being returned to the original document. Your application is responsible for determining the data format contained in the scrap, and putting the data back into its native format if necessary.

To establish the format of the embedded object, determine its CLSID by retrieving the object's CF_OBJECTDESCRIPTOR format. If the CLSID indicates a data format that belongs to the application, it should transfer the native data instead of calling **OleCreateFromData**.

Cached data formats

When the shell creates a scrap file, it checks the registry for the list of available formats. By default, there are two formats available: CF_EMBEDSOURCE and CF_LINKSOURCE. However, there are a number of scenarios where applications might need to have scrap files in different formats:

- To allow scraps to be transferred to non-OLE containers, which cannot accepted embedded object formats.
- To allow suites of applications to communicate with a private format.
- To make round trips easier to handle.

Applications can add formats to the scrap by caching them in the registry. There are two types of cached formats:

- Priority cache formats. For these formats, the data is copied in its entirety into the scrap from the data object.
- Delay-rendered formats. For these formats, the data object is not copied to the scrap. Instead, rendering is delayed until a target requests the data. Delay-rendering is discussed in more detail in the next section.

To add a priority cache or delay-rendered format, create a **DataFormat** subkey under the **CLSID** key of the application that is the source of the data. Under that subkey, create a **PriorityCacheFormats** or **DelayRenderFormats** subkey. For each priority cache or delay-rendered format, create a numbered subkey starting with zero. Set the value of this key to either a string with the registered name of the format, or a #X value, where X represents the format number of a standard clipboard format.

The following sample shows cached formats for two applications. The MyProg1 application has the rich-text format as a priority cache format, and a private format "My Format" as a delay-rendered format. The MyProg2 application has the CF_BITMAP format ("#8") as a priority cache format.

```
HKEY_CLASSES_ROOT
   CLSID
      {GUID}="MyProg1"
         DataFormats
            PriorityCacheFormats
               0="Rich Text Format"
            DelayRenderFormats
               0 ="My Format"
      {GUID}="MyProg2"
         DataFormats
            PriorityCacheFormats
               0="#8"
```

Additional formats can be added by creating additional numbered subkeys.

Delayed Rendering

A delayed rendering format allows an application to create a scrap file but delay the expense of rendering the data until it is requested by a target. The **IDataObject** interface of a scrap will offer the delayed rendering formats to the target along with native and cached data. If the target requests a delayed rendering format, the shell will run the application and provide the data to the target from the active object.

Note Because delayed rendering is somewhat risky, it should be used with caution. It will not work if the server is not available, or on applications that are not OLE-enabled.

Dragging and Dropping Shell Objects Asynchronously

Scenario: A user transfers a large block of data from source to target. To avoid blocking both applications for a significant amount of time, the target extracts the data asynchronously.

Normally, drag-drop is a synchronous operation. In brief:

1. The drop source calls **DoDragDrop** and blocks its primary thread until the function returns. Blocking the primary thread normally blocks UI processing.
2. After the target's **IDropTarget::Drop** method is called, the target extracts the data from the data object on its primary thread. This procedure normally blocks the target's UI processing for the duration of the extraction process.
3. Once the data has been extracted, the target returns the **IDropTarget::Drop** call, the system returns **DoDragDrop**, and both threads can proceed.

In short, synchronous data transfer can block the primary threads of both applications for a significant amount of time. In particular, both threads must wait while the target extracts the data. For small amounts of data, the time required to extract data is small and synchronous data transfer works quite well. However, synchronously extracting large amounts of data can cause lengthy delays and interfere with the UI of both target and source.

The **IAsyncOperation** interface is an optional interface that can be implemented by a data object. It gives the drop target the ability to extract data from the data object asynchronously on a background thread. Once data extraction is handed off to the background thread, the primary threads of both applications are free to proceed.

How to use IASyncOperation

The essential purpose of **IAsyncOperation** is to allow the drop source and drop target to negotiate whether data can be extracted asynchronously. The following procedure outlines how the drop source uses the interface:

1. Create a data object that exposes **IAsyncOperation**.

2. Call **IAsyncOperation::SetAsyncMode** with *fDoOpAsync* set to VARIANT_TRUE to indicate that an asynchronous operation is supported.
3. After **DoDragDrop** returns, call **IAsyncOperation**::InOperation.
   - If **IAsyncOperation::InOperation** fails or returns VARIANT_FALSE, a normal synchronous data transfer has taken place and the data extraction process is finished. The source should do any cleanup that is required, and proceed.
   - If **IAsyncOperation::InOperation** returns VARIANT_TRUE, the data is being extracted asynchronously. Cleanup operations should be handled by **IAsyncOperation::EndOperation**.
4. Release the data object.
5. When the asynchronous data transfer is complete, the data object normally notifies the source through a private interface.

The following procedure outlines how the drop target uses the IAsyncOperation interface to extract data asynchronously:

1. When the system calls **IDropTarget::Drop**, call **IDataObject::QueryInterface** and request an **IAsyncOperation** interface (IID_IAsyncOperation) from the data object.
2. Call **IAsyncOperation::GetAsyncMode**. If the method returns VARIANT_TRUE, the data object supports asynchronous data extraction.
3. Create a separate thread to handle data extraction and call **IAsyncOperation::StartOperation**.
4. Return the **IDropTarget::Drop** call, as you would for a normal data-transfer operation. **DoDragDrop** will return and unblock the drop source. Do not call **IDataObject::SetData** to indicate the outcome of an optimized move or delete-on-paste operation. Wait until the operation is finished.
5. Extract the data on the background thread. The target's primary thread is unblocked and free to proceed.
6. If the data transfer was an optimized move or delete-on-paste operation, call **IDataObject::SetData** to indicate the outcome.
7. Notify the data object that extraction is finished by calling **IAsyncOperation::EndOperation**.

Extending the Shell

Creating a File Association

Files that contain a particular type of data commonly have the same *file name extension*. It is appended to the file name, and typically consists of a dot, followed by three alphanumeric characters. For example, ANSI text files commonly have a .txt file name extension.

Although it is customary, file name extensions are not restricted to three letters on systems that support long file names. On Microsoft Windows 95 and later systems, you can use any number of characters you like as long as the file name doesn't exceed 255 characters.

Note You can use multiple dots in a file name, but only those characters following the final dot will be recognized as a file name extension. Any other dots will be treated as part of the file name. Although file names can contain spaces, do not use spaces in file name extensions.

Defining a File Class

Files with a common file name extension can be defined as members of a *file class*. Defining a file class allows you to extend the shell by customizing the behavior of all files in the class. The Shell Basics section discusses those behaviors that can be customized by adding registry entries, including:

- Specifying the application used to open the file when it is double-clicked.
- Adding commands to the context menu.
- Specifying a custom icon.

For a greater degree of control over the behavior of a file class, you can write one or more *shell extension handlers*. For more information, see *Creating Shell Extension Handlers*.

To define a file class, first create a registry key for the extension, including the dot, under HKEY_CLASSES_ROOT. Set the key's value to the ProgID for the associated application. Next, create a second key under HKEY_CLASSES_ROOT for the application's ProgID. Set it to a REG_SZ value that describes the application. For example, to create a file class with a .myp extension and an associated application, MyProgram.exe with a ProgID of MyProgram.1, the registry entries would be:

```
HKEY_CLASSES_ROOT
    .myp=MyProgram.1
    ...
    MyProgram.1=MyProgram Application
```

A user can act on a member of the file class in a variety of ways, such as double-clicking or right-clicking the file in Windows Explorer. Once these two keys are in place, you can add subkeys to them to customize the behavior of the file class and its associated application. When a user acts on a member of the class, the shell's response will include the information contained in these keys.

Defining Attributes for a File Class

Assigning attributes to a file class allows you to control some aspects of its behavior. It also allows you to limit the extent to which the user can modify various aspects of the class, such as its icon or verbs, with the Folder Options property sheet. The attributes are defined as binary flags. To assign attributes to a file class, combine the selected attributes with a logical OR to form a single attribute value. Add an EditFlags REG_BINARY value to the class's ProgID key and set it to the attribute value. The following table lists the file class attributes, and their numerical values.

Flag	Value	Description
FTA_Exclude	0x00000001	Exclude the file class
FTA_Show	0x00000002	Show file classes, such as folders, that aren't associated with a filename extension.
FTA_HasExtension	0x00000004	The file class has a filename extension.
FTA_NoEdit	0x00000008	The registry entries associated with this file class cannot be edited. New entries cannot be added and existing entries cannot be modified or deleted.
FTA_NoRemove	0x00000010	The registry entries associated with this file class cannot be deleted.
FTA_NoNewVerb	0x00000020	No new verbs can be added to the file class.
FTA_NoEditVerb	0x00000040	Canonical verbs such as open and print cannot be modified or deleted.
FTA_NoRemoveVerb	0x00000080	Canonical verbs such as open and print cannot be deleted.
FTA_NoEditDesc	0x00000100	The description of the file class cannot be modified or deleted.
FTA_NoEditIcon	0x00000200	The icon assigned to the file class cannot be modified or deleted.
FTA_NoEditDflt	0x00000400	The default verb cannot be modified.
FTA_NoEditVerbCmd	0x00000800	The commands associated with verbs cannot be modified.
FTA_NoEditVerbExe	0x00001000	Verbs cannot be modified or deleted.
FTA_NoDDE	0x00002000	The DDE-related entries cannot be modified or deleted.
FTA_NoEditMIME	0x00008000	The content-type and default-extension entries cannot be modified or deleted.
FTA_OpenIsSafe	0x00010000	The file class's open verb can be safely invoked for downloaded files.

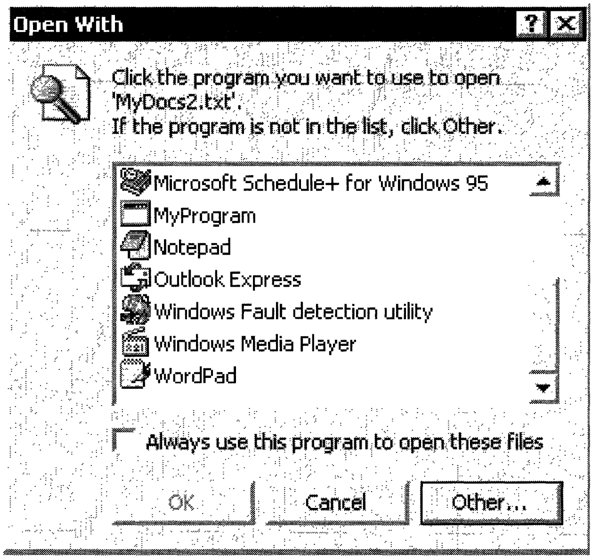
FTA_AlwaysUnsafe	0x00020000	Do not allow the "Never ask me" checkbox to be enabled. The user can override this attribute through the File Type dialog box.
FTA_AlwaysShowExt	0x00040000	Always show the file class's file name extension, even if the user has selected the "Hide Extensions" option.
FTA_NoRecentDocs	0x00100000	Don't add members of this file class to the Recent Documents folder.

The following example assigns the FTA_NoRemove and FTA_NoNewVerb attributes to the .myp file class.

```
HKEY_CLASSES_ROOT
    .myp=MyProgram.1
    ...
    MyProgram.1=MyProgram Application
        EditFlags=00 00 00 30
```

Excluding an Application from the Open With Dialog Box

The Open With dialog box, shown in Figure 6-4, is launched by default when the user double-clicks a file that is not a member of a file class. It is also usually one of the items that appears on the context menu that is displayed when the user right-clicks a file. The purpose of this dialog box is to allow the user to specify which application they want to use to open the file.



Open With

Click the program you want to use to open 'MyDocs2.txt'.
If the program is not in the list, click Other.

Microsoft Schedule+ for Windows 95
MyProgram
Notepad
Outlook Express
Windows Fault detection utility
Windows Media Player
WordPad
□ Always use this program to open these files
OK
Cancel
Other...

Figure 6-4: The Open With dialog box.

The applications that are listed on the Open With dialog are registered as subkeys of **HKEY_CLASSES_ROOT\Applications**. However, many applications should not be used to open files that are not members of their associated file class. The preferred way to exclude an application from the Open With dialog box is to add a NoOpenWith REG_SZ value name to the application's subkey. For example, the following sample registry entry excludes MyProgram.exe from the Open With dialog box.

```
HKEY_CLASSES_ROOT
    ...
    Applications
        ...
        MyProgram.exe
            NoOpenWith
```

An alternative way to exclude an application from the Open With dialog box is to append the application's file name to Windows Explorer's kill list. This list is a REG_SZ value of the **HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Explorer\FileAssociation** key named KillList. The kill list is a string consisting of the file names of the applications, separated by semicolons (;). The following example excludes MyProgram.exe from the Open With dialog box by appending it to Windows Explorer's kill list, along with rundll.exe and taskman.exe.

```
HKEY_LOCAL_MACHINE
    Software
        Microsoft
            Windows
                CurrentVersion
                    Explorer
                        FileAssociation
                            KillList=%1;rundll.exe;
taskman.exe;MyProgram.exe
```

Customizing Icons

Microsoft Windows supplies default icons for every item displayed on the desktop or in the Windows Explorer. Mass storage devices, such as disk drives, are also assigned a default icon. However, these icons often provide little insight to the user as to the contents of the file or what program is associated with it. You can assign a custom icon to a file system folder by creating a Desktop.ini file. This document discusses how use the registry to associate custom icons with file classes and drive letters.

Assigning a Custom Icon to a File Class

By default, all files are displayed on the desktop or in Windows Explorer with a default icon. For example, Figure 6-5 shows this icon used with MyDocs4.xyz.

All the files displayed in this screen shot contain ANSI text. The reason that the files with the .txt extension do not display the default icon is that .txt has been registered as a file class and assigned a custom icon.

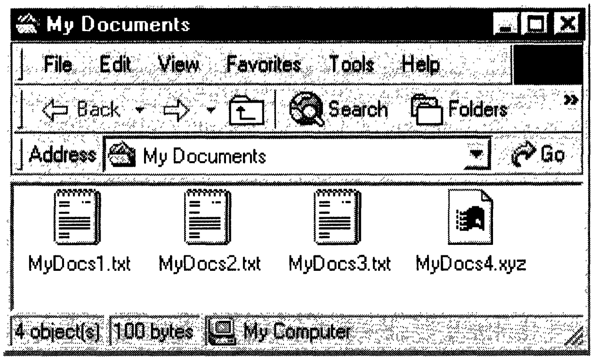


Figure 6-5: Default file icon.

Assigning a custom icon to a file class is a simple matter. Create a subkey, under the key for the application's ProgID, and name it DefaultIcon. Assign it a REG_SZ value containing the fully qualified path for the file with the icon. Any file containing an icon is acceptable, including .ico, .exe, and .dll files. If there is more than one icon in the file, the path should be followed by a comma, and then the index of the icon.

Figure 6-6 shows a custom icon that has been assigned to the .myp file class, which was also used in the example in *Creating a File Class*. The My Documents directory now looks like this:

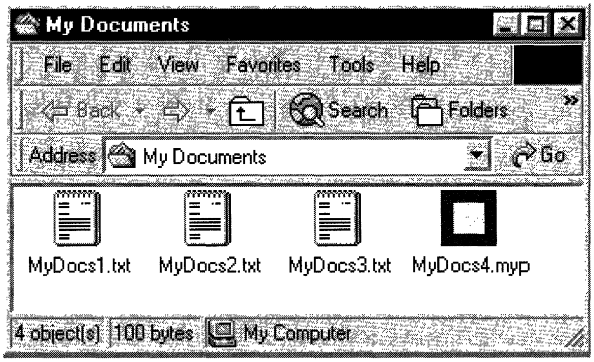


Figure 6-6: Custom file icon.

In this example, the icon is in the c:\MyDir\MyProgram.exe file, with an index of two. The registry entry that assigns the custom icon to all .myp files is:

```
HKEY_CLASSES_ROOT
    .myp=MyProgram.1


    ...
    MyProgram.1=MyProgram Application
        DefaultIcon=C:\MyDir\MyProgram.exe,2
```

Assigning a Custom Icon and Label to a Drive Letter

For shell versions 4.71 and later, you can use the registry to replace the standard drive icon with a custom icon. With versions 5.0 and later, you can also add a custom label. Custom drive icons and labels are normally used for removable mass storage devices, such as tape drives, to allow users to easily distinguish them from their system's hard and floppy drives.

To replace the standard drive icon with a custom icon, add a subkey named for the drive letter to **HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Explorer\DriveIcons\**. The drive letter should not be followed by a colon (:). Add a **DefaultIcon** subkey to the drive letter subkey and set its default value to a string containing the location of the icon. The first part of the string contains the fully-qualified path of the icon's file. If there is more than one icon in the file, the path is followed by a comma, and then the zero-based index of the icon. To add a custom label, add a **DefaultLabel** subkey to the drive letter subkey, and set its default value to a string containing the label.

The following example specifies a custom icon and label for the E: drive. The icon is in the C:\MyDir\MyDrive.exe file with a zero-based index of three.

```
HKEY_LOCAL_MACHINE
  Software
    Microsoft
      Windows
        CurrentVersion
          Explorer
            DriveIcons
              E
                DefaultIcon=C:\MyDir\MyDrive.exe,3
                DefaultLabel="MyDrive"
```

If you change the registry programmatically, you must also call SHChangeNotify to notify the shell to update its cache.

Extending Context Menus

Right clicking an object with Microsoft Windows 95 and later systems usually pops up a *context menu*. This menu contains a list of commands that the user can select to perform various actions on the object. This section is an introduction to context menus for file system objects.

Context Menus for File System Objects

When a user right-clicks an object, such as a file, that is displayed in Windows Explorer or on the desktop, a context menu appears with a list of commands. The user can then perform an action on the file, such as opening or deleting it, by selecting the appropriate command.

Because context menus are often used for file management, the shell provides a set of default commands, such as Cut and Copy, that will appear on the context menu for any file. Note that although Open With is a default command, it is not displayed for some standard file classes, such as .wav. Figure 6-7, a sample of the My Documents directory that was also used as an example in *Customizing Icons*, shows a default context menu that was displayed by right-clicking MyDocs4.xyz.

The reason that MyDocs4.xyz shows a default context menu is that it is not a member of a registered file class. On the other hand, .txt is a registered file class. If you right-click one of the .txt files, you will instead see a context menu with two additional commands in its upper section: Open and Print. (See Figure 6-8.)

Once a file class is registered, you can extend its context with additional commands. They are displayed above the default commands when any member of the class is right-clicked. Although most of the commands added in this way are common ones, such as Print or Open, you are free to add any command that a user might find helpful.

All that is required to extend the context menu for a file class is to create a registry entry for each command. A more sophisticated approach is to implement a *context menu handler*, which allows you extend the context menu for a file class on a file by file basis. For more information, see *Creating Context Menu Handlers*.

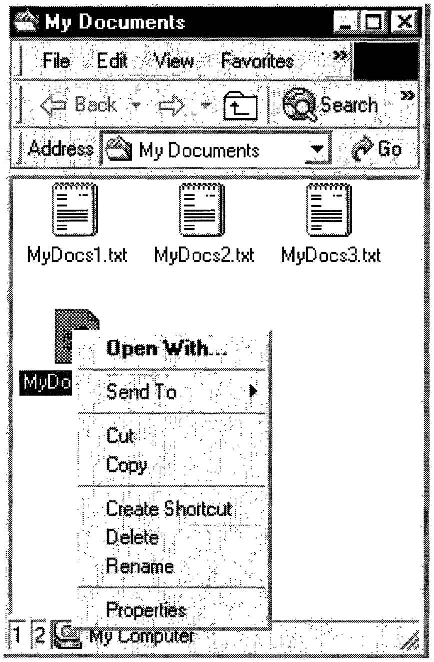


Figure 6-7: Default context menu.

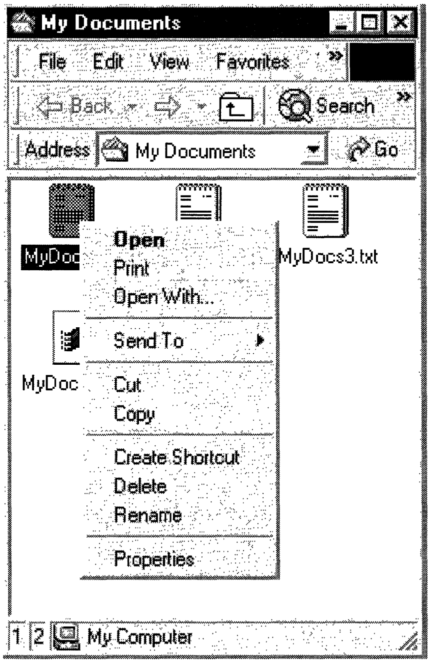


Figure 6-8: Context menu associated with a registered file class.

Verbs

Each command on the context menu is identified in the registry by its *verb*. These verbs are the same as those used by **ShellExecuteEx** when launching applications programmatically. For further information about the use of **ShellExecuteEx**, see the discussion in *Launching Applications*.

A verb is a simple text string that is used by the shell to identify the associated command. Each verb corresponds to the *command string* that would be used to launch the command in a console window or batch (.bat) file. For example, the **open** verb normally launches a program to open a file. Its command string typically looks something like this:

```
My Program.exe "%1"
```

Note If any element of the command string contains or might contain spaces, it must be enclosed in quotation marks. Otherwise, if the element contains a space, it will not parse correctly. For instance, "My Program.exe" will launch the application properly. If you use My Program.exe, the system will attempt to launch "My" with "Program.exe" as its first command line argument. You should always use quotation marks with arguments such as "%1" that are expanded to strings by the shell, because you cannot be certain that the string will not contain a space.

Verbs can also have a *display string* associated with them, which is displayed on the context menu instead of the verb string itself. For example, the display string for **openas** is "Open With...". Like normal menu strings, including an ampersand (&) in the display string allows keyboard selection of the command.

Canonical Verbs

In general, applications are responsible for providing localized display strings for the verbs they define. However, to provide a degree of language independence, the system defines a standard set of commonly used verbs called *canonical verbs.* A canonical verb can be used with any language and the system will automatically generate a properly localized display string. For instance, the **open** verb's display string will be set to "Open" on an English system, and "Öffnen" on a German system.

The canonical verbs include:

- **open**
- **print**
- **explore**
- **find**
- **openas**
- **properties**

The **printto** verb is also canonical, but never displayed. It allows the user to print a file by dragging it to a printer object.

Extended Verbs

When the user right-clicks an object, the context menu contains all the normal verbs. However, there may be commands that you wish to support, but not have displayed on every context menu. For example, you may have commands that are not commonly used, or intended for experienced users. For this reason, you can also define one or more *extended verbs.* These verbs are also character strings, and are essentially similar to normal verbs. They are distinguished from normal verbs by the way they are registered. To have access to the commands associated with extended verbs, the user must right-click an object while pressing the SHIFT key. They will then be displayed along with the normal verbs.

Extending the Context Menu for a File Class

The simplest way to extend the context menu for a file class is with the registry. To do this, add a **shell** subkey, below the key for the ProgID of the application associated with the **file class**. Optionally, you can define a *default verb* for the file class by making it the default value of the **shell** subkey.

The default verb is displayed first on the context menu. Its purpose is to provide the shell with a verb it can use when **ShellExecuteEx** is called, but no verb is specified. The shell does not necessarily select the default verb when **ShellExecuteEx** is used in this fashion. For shell versions 5.0 and later, found on Windows 2000 and later systems, the shell uses the first available verb from the following list. If none are available, the operation fails.

1. The **open** verb
2. The default verb
3. The first verb in the registry
4. The **openwith** verb

For shell versions prior to version 5.0, omit item three.

Below the shell subkey, create one subkey for each verb you wish to add. Each of these subkeys will have a REG_SZ value set to the verb's display string. You can omit the display string for canonical verbs because the system will automatically display a properly localized string. If you omit the display string for non-canonical verbs, the verb string will be displayed. For each verb subkey, create a **command** subkey, with the default value set to the command string.

Figure 6-9 shows a context menu for the .myp file class used in *Creating a File Association* and *Customizing Icons*. It now has **open**, **doit**, **print**, and **printto** verbs on its context menu, with **doit** as the default verb. The context menu will look like this:

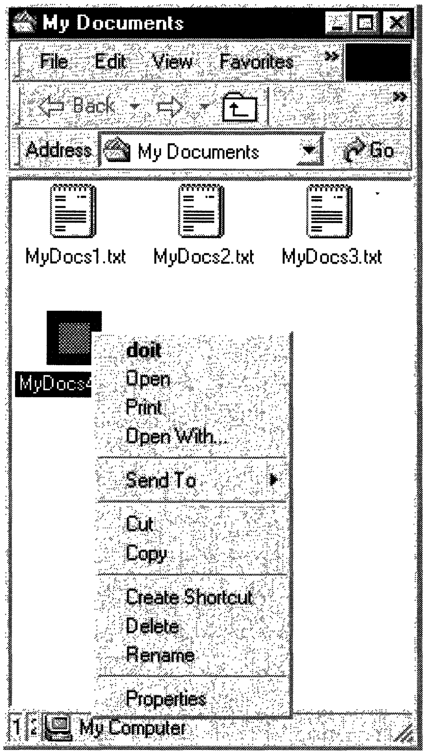


Figure 6-9: Context menu for the .myp file class.

The registry entries used to extend the context menu shown in the preceding illustration are:

```
HKEY_CLASSES_ROOT
    .myp=MyProgram.1
    ...
    MyProgram.1=MyProgram Application
        shell=doit
            open
                commandC:\MyDir\MyProgram.exe "%1"
            doit="&Do It"
                command=C:\MyDir\MyProgram.exe /d "%1"
            print
                command=C:\MyDir\MyProgram.exe /p "%1"
            printto
                command=C:\MyDir\MyProgram.exe /p "%1" "%2" %3 %4
```

Although the "Open With..." command is above the first separator, it is automatically created by the system and doesn't require a registry entry. The system will automatically create display names for the canonical verbs, **open** and **print**. Because **doit** is not a canonical verb, it is assigned a display name, "&Do It", which can be selected by pressing the 'd' key. The **printto** verb does not appear on the context menu, but including it allows the user to print files by dropping them on a printer icon. In this example, %1 represents the file name and %2 the printer name. You can ignore %3 and %4 for Windows 95 and later systems. For Windows 3.1 systems, %3 represents the driver name and %4 the port name. Unlike most similar arguments, the %3 and %4 arguments of **printto** should not be enclosed in quotation marks.

Defining Extended Verbs

You can also use the registry to define one or more extended verbs. The associated commands will only be displayed when the user right-clicks an object while also pressing the shift key. To define a verb as extended, simply add an "extended" REG_SZ value to the verb's subkey. The value should not have any data associated with it. The following sample registry entry shows the example from the previous section, with **doit** defined as an extended verb.

```
HKEY_CLASSES_ROOT
    .myp=MyProgram.1
    ...
    MyProgram.1=MyProgram Application
        shell=doit
            open
                command=C:\MyDir\MyProgram.exe "%1"
            doit="&Do It"
                extended
                command=C:\MyDir\MyProgram.exe /d "%1"
```

(continued)

(continued)

```
      print
        command=C:\MyDir\MyProgram.exe /p "%1"
      printto
        command=C:\MyDir\MyProgram.exe /p "%1" "%2" %3 %4
```

Associating Verbs with DDE Commands

Invoking a verb normally launches the application specified by the verb's **command** subkey. However, if your application supports DDE, you can instead have the shell initiate a DDE conversation.

To specify that invoking a verb should initiate a DDE conversation, add a **ddeexec** subkey to the verb's key. Set the default value of **ddeexec** to the DDE command string. The **ddeexec** key has three optional subkeys that provide some control over the DDE process.

- **application**. Set the default value of this subkey to the application name to be used to establish the DDE conversation. If there is no **application** subkey, the default value of the verb's **command** subkey is used as the application name.
- **topic**. Set the default value of this subkey to the topic name of the DDE conversation. If there is no **topic** subkey, System is used as the topic name.
- **ifexec**. Set the default value of this subkey to the DDE command to be used if DDE conversation cannot be initiated. When initiation fails, the application specified by the default value of the verb's **command** subkey is launched. If an **ifexec** key exists, its default value will then be used as the DDE command. If there is no **ifexec** subkey, the default value of the **ddeexec** key will used again as the DDE command.

The following example specifies that invoking the **open** verb for MyProgram.1 initiates a DDE conversation with a DDE command of Open("%1"), and an application name of MyProgram.

```
HKEY_CLASSES_ROOT
  ...
  MyProgram.1=MyProgram Application
    shell=doit
      open
        command=C:\MyDir\MyProgram.exe "%1"
        ddeexec=Open("%1")
          application=MyProgram
```

Extending the New Submenu

When a user opens the File menu in Windows Explorer, the first command is New. Selecting this command displays a submenu. By default, it contains two commands, Folder and Shortcut, that allow users to create subfolders and shortcuts. This submenu can be extended to include file creation commands for any file class.

To add a file-creation command to the New submenu, your application's files must have a file class associated with them. Include a **ShellNew** subkey, under the file extension key. When the File menu's New command is selected, the shell will add it to the New submenu. The command's display string will be the descriptive string that is assigned to the program's ProgID.

Assign one or more data values to the ShellNew subkey to specify the file creation method. The available values follow:

Value	Description
Command	Executes an application. This is an REG_SZ value specifying the path of the application to be executed. For example, you could set it to launch a wizard.
Data	Creates a file containing specified data. **Data** can be either a REG_SZ or REG_BINARY value with the file's data. **Data** is ignored if either **NullFile** or **FileName** are specified.
FileName	Creates a file that is a copy of a specified file. **FileName** is a REG_SZ value, set to the fully qualified path of the file to be copied.
NullFile	Creates an empty file. **NullFile** is not assigned a value. If **NullFile** is specified, **Data** and **FileName** are ignored.

Figure 6-10 shows the New submenu for the .myp file class used as an example in *Creating a File Association* and *Customizing Icons*. It now has a command, "MyProgram Application". When a user selects MyProgram Application from the New submenu, the shell will create a file named "New MyProgram Application.myp" and pass it to MyProgram.exe.

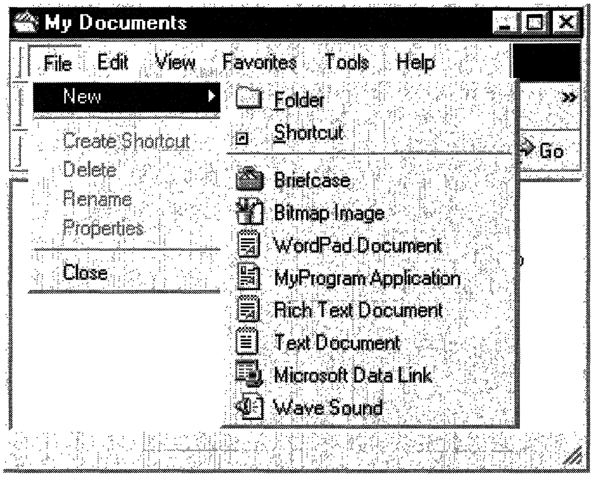


Figure 6-10: The New submenu for the .myp file class.

The registry entry is now:

```
HKEY_CLASSES_ROOT
    .myp=MyProgram.1
        ShellNew=NULLFILE
    ...
    MyProgram.1=MyProgram Application
        DefaultIcon=C:\MyDir\MyProgram.exe,2
        shell=doit
            open
                command=C:\MyDir\MyProgram.exe "%1"
            doit="&Do It"
                command=C:\MyDir\MyProgram.exe /d "%1"
            print
                command=C:\MyDir\MyProgram.exe /p "%1"
            printto
                command=C:\MyDir\MyProgram.exe /p "%1"
"%2" %3 %4
```

Customizing Folders with Desktop.ini

File system folders are commonly displayed with a standard icon and set of properties, which specify, for instance, whether or not the folder is shared. You can customize the appearance and behavior of an individual folder in two ways:

- Create a Desktop.ini file for the folder
- Create a Folder.htt template for the folder

Folders can be displayed in either Classic or Web style. For a detailed discussion of these styles, see *Web View*. The Desktop.ini file, discussed here, applies to both styles. It allows you to assign a custom icon to a folder and control its behavior in a limited way.

To customize the folder's style beyond what is possible with the Desktop.ini file, you must create a custom Folder.htt template for the folder. This file will affect only the appearance of the folder when the Web style is selected. If this template is not present in the folder, the shell uses a default template. For further discussion of .htt templates, see *The Web View Template*.

Using Desktop.ini Files

Folders are normally displayed with the standard folder icon. The most common use of the Desktop.ini file is to assign a custom icon to a folder. This icon will be displayed in Classic style as well as Web style, and it will appear next to the folder's name anywhere the name appears. You can also use Desktop.ini to create an *infotip* that displays information about the folder and controls some aspects of the folder's behavior, such as whether it can be shared.

Use the following procedure to customize a folder's style with Desktop.ini:

- Use **PathMakeSystemFolder** to make the folder a system folder. You can also make a folder a system folder from the command line by using **attrib +s** *FolderName.*
- Create a Desktop.ini file for the folder. You should mark it as *hidden* and *read-only* to protect it from being modified.

Creating a Desktop.ini File

The Desktop.ini file is a text file that allows you to specify how a file system folder will be viewed. There are three sections in the file. The first two, [ExtShellFolderViews] and [{5984FFE0-28D4-11CF-AE66-08002B2E1262}] are necessary only if you want to use a custom Folder.htt template. If you omit them, the system will use the default template. To use a custom Folder.htt template, you must include these two sections into the Desktop.ini file exactly as they are shown:

```
[ExtShellFolderViews]
Default={5984FFE0-28D4-11CF-AE66-08002B2E1262}
{5984FFE0-28D4-11CF-AE66-08002B2E1262}={5984FFE0-28D4-11CF
-AE66-08002B2E1262}

[{5984FFE0-28D4-11CF-AE66-08002B2E1262}]
PersistMoniker=file://Folder.htt
```

The third section, [.ShellClassInfo], allows you to customize the folder's view by assigning values to several entries:

Entry	Value
ConfirmFileOp	Set this entry to 0 to avoid a "You Are Deleting a System Folder" warning when deleting or moving the folder.
NoSharing	Set this entry to 1 to prevent the folder from being shared.
IconFile	If you want to specify a custom icon for the folder, set this entry to the icon's file name. The .ico file extension is preferred, but it is also possible to specify .bmp files, or .exe and .dll files that contain icons. If you use a relative path, the icon will be available to people who view the folder over the network. You must also set the IconIndex entry.
IconIndex	Set this entry to specify the index for a custom icon. If the file assigned to IconFile only contains a single icon, set IconIndex to 0.
InfoTip	Set this entry to an informational text string. It will be displayed as an infotip when the cursor hovers over the folder. If the user clicks on the folder in a Web view, the information text will be displayed in the folder's information block, below the standard information.

The following illustrations are of the Music folder with a custom Desktop.ini file. The folder now:

- Has a custom icon.
- Does not display a "You Are Deleting a System Folder" warning if the folder is moved or deleted.
- Cannot be shared.
- Displays informational text when the cursor hovers over the folder.

The folder options in Figures 6-11 through 6-13 have been set to show hidden files, so that Desktop.ini is visible. A folder needs only its own Folder.htt template if it does not use the default template. The Web style view of the folder looks like this:

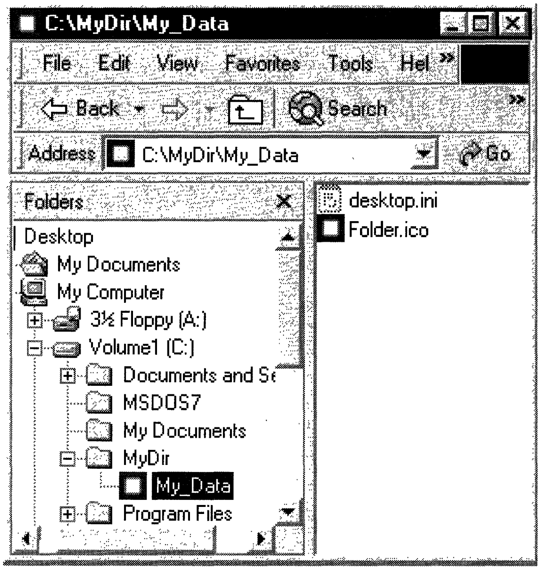
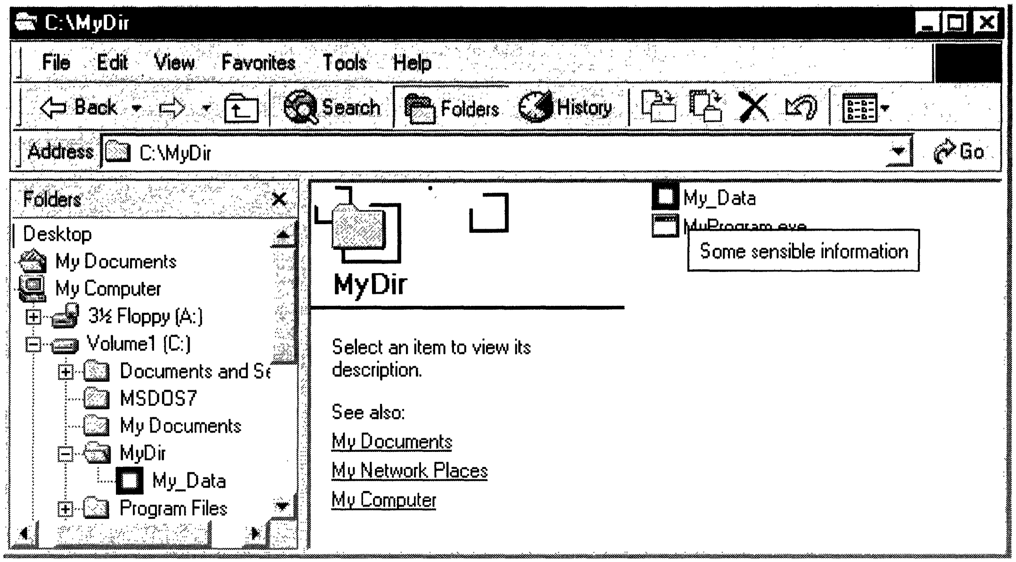


Figure 6-11: Web view of the Music folder.

When the cursor hovers over the folder, the infotip is displayed.



Some sensible information

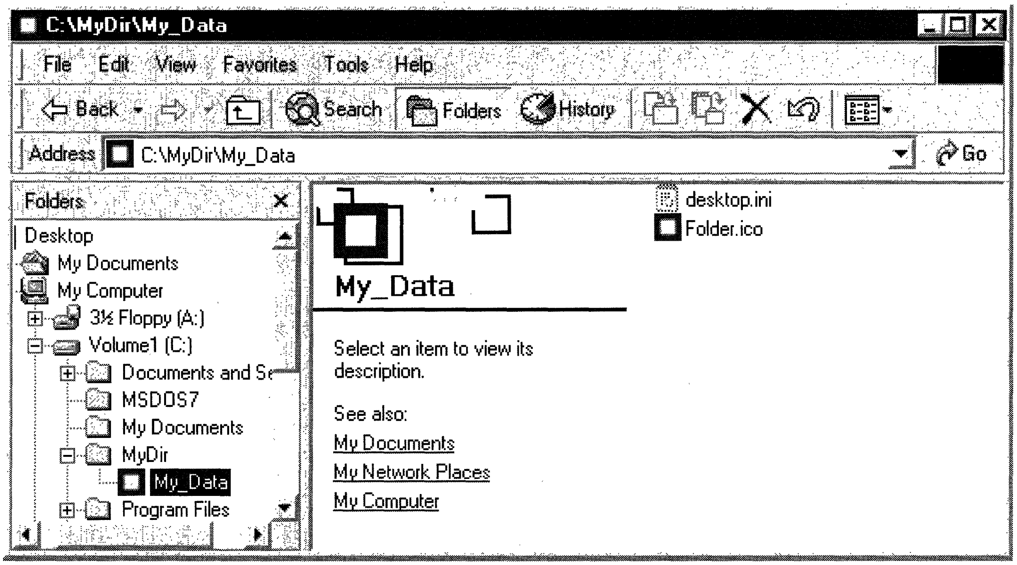
Select an item to view its description.

See also:
My Documents
My Network Places
My Computer

Figure 6-12: Infotip displayed over the folder.

The Classic style of this folder is similar, with the custom icon replacing the folder icon everywhere the folder name appears.

The following desktop.ini file was used to customize the Music folder, as seen in Figures 6-11 through 6-13. For instructional purposes, it includes the optional sections that are needed if you want to use a custom Folder.htt template.



Select an item to view its description.

See also:
My Documents
My Network Places
My Computer

Figure 6-13: Custom icon in use throughout the dialog box.

```
[ExtShellFolderViews]
Default={5984FFE0-28D4-11CF-AE66-08002B2E1262}
{5984FFE0-28D4-11CF-AE66-08002B2E1262}={5984FFE0-28D4-11CF-AE66-08002B2E1262}

[{5984FFE0-28D4-11CF-AE66-08002B2E1262}]
PersistMoniker=file://Folder.htt

[.ShellClassInfo]
ConfirmFileOp=0
NoSharing=1
IconFile=Folder.ico
IconIndex=0
InfoTip=Some sensible information.
```

Creating an AutoPlay-Enabled CD-ROM Application

Microsoft AutoPlay is a feature of the Microsoft Windows operating system. It automates the procedures for installing and configuring products designed for Windows-based platforms that are distributed on CD-ROMs. When users insert an AutoPlay-enabled compact disc into their CD-ROM drive, AutoPlay automatically runs an application on the CD-ROM that installs, configures, or runs the selected product. If you want your CD-ROM product to display the Microsoft Windows 95 logo, it must be enabled for AutoPlay.

AutoPlay can be used to install and run CD-ROM applications written for MS-DOS, Windows 3.0 and Windows 3.1, and all 32-bit versions of Windows. Although AutoPlay is most commonly used for Windows applications, it can also be used to install, configure, or run MS-DOS–based applications in a Windows MS-DOS session. You can configure each MS-DOS–based application with its own unique icon, Config.sys file, and Autoexec.bat file. Windows creates the correct configuration files for the MS-DOS–based application. The startup application then starts the MS-DOS–based application in a window.

There are two essential requirements that a system must meet in order for AutoPlay to work.

- The system must be running Windows 95, Windows NT 4.0, or later versions. MS-DOS, Windows versions 3.1 and earlier, and Windows NT versions 3.51 and earlier do not support AutoPlay. You can use AutoPlay-enabled CD-ROMs in such systems. However, the AutoPlay features will be ignored, and the CD-ROMs will behave as ordinary CD-ROMs.
- The CD-ROM drive must have 32-bit device drivers that detect when a user inserts a compact disc and notify the system. Device drivers for MS-DOS or 16-bit versions of Windows do not have this feature.

The following sections discuss how to implement an AutoPlay-enabled CD-ROM application.

Creating an AutoPlay-Enabled Application

Creating an AutoPlay-enabled CD-ROM application is a straightforward procedure. You simply include two essential files:

- An Autorun.inf file
- A startup application

When a user inserts a disc into a CD-ROM drive on a AutoPlay-compatible computer, the system immediately checks to see if the disc has a personal computer file system. If it does, the system searches for a file named Autorun.inf. This file specifies setup application that will be run, along with a variety of optional settings. The startup application typically installs, uninstalls, configures, and perhaps runs the application.

Creating an Autorun.inf File

Autorun.inf is a text file located in the root directory of the CD-ROM that contains your application. Its primary function is to provide the system with the name and location of the application's startup program that will be run when the disc is inserted. The Autorun.inf file can also contain optional information including:

- The name of a file that contains an icon that will represent your application's CD-ROM drive. This icon will be displayed by Microsoft Windows Explorer in place of the standard drive icon.
- Additional commands for the context menu that is displayed when the user right-clicks the CD-ROM icon. You can also specify the default command that is run when the user double-clicks the icon.

Autorun.inf files are similar to .ini files. They consist of one or more sections, each headed by a name enclosed in square brackets. Each section contains a series of commands that will be run by the shell when the disc is inserted. There are two sections that are currently defined for Autorun.inf files:

- The **[autorun]** section contains the default AutoPlay commands. All Autorun.inf files must have an **[autorun]** section.
- An optional **[autorun.alpha]** section can be included for Microsoft Windows NT 4.0 and later systems running on RISC-based computers. When a disc is inserted in a CD-ROM drive on a RISC-based system, the shell will run the commands in this section instead of those in the **[autorun]** section.

Note The shell checks for an architecture-specific section first. If it does not find one, it uses the information in the **[autorun]** section. After the shell finds a section, it ignores all others, so each section must be self-contained.

Each section contains a series of commands that determine how the Autorun operation takes place. There are five commands available:

Command	Description
defaulticon	Specifies the default icon for the application.
icon	Specifies the path and file name of an application-specific icon for the CD-ROM drive.
open	Specifies the path and file name of the startup application.
shell	Defines the default command in the CD-ROM's context menu.
shell\verb	Adds commands to the CD-ROM's context menu.

The following is an example of a simple Autorun.inf file. It specifies Filename.exe as the startup application. The second icon in Filename.exe will represent the CD-ROM drive instead of the standard drive icon.

```
[autorun]
open=Filename.exe
icon=Filename.exe,1
```

This Autorun.inf sample runs different startup applications depending on the type of computer:

```
[autorun]
open=Filename_x86.exe
icon=IconFile.ico

[autorun.alpha]
open=Filename_RISC.exe
icon=IconFile.ico
```

Tips for Writing AutoPlay Startup Applications

There are essentially no constraints on how to write an AutoPlay startup application. You can implement it to do whatever you find necessary to install, uninstall, configure, or run your application. However, the following tips provide some guidelines to implementing an effective AutoPlay startup application.

- Users should receive feedback as soon as possible after they insert an AutoPlay compact disc into the CD-ROM drive. Startup applications should thus be small programs that load quickly. They should clearly identify the application and provide an easy way to cancel the operation.

- Typically, the initial part of the startup application presents users with a user interface, such as a dialog box, asking them how they would like to proceed. Check to see if the program is already installed. If not, the next step will probably be the setup procedure. The startup application can take advantage of the time the user spends reading the dialog box by starting another thread to begin loading the setup code. By the time the user clicks **OK**, your setup program will already be partly if not fully loaded. This approach significantly reduces the user's perception of the amount of time it takes to load your application.
- If the application has already been installed, the user probably inserted the disk to run the application. As with the setup case, you can start another thread to begin loading application code to shorten the waiting time for the user.
- Hard disk space may be a limited resource on many systems. Here are a few hints for minimizing hard disk usage:
  - Keep the number of files that must be on the hard disk to a minimum. They should be restricted to files that are essential to running the program or that would take an unacceptably large amount of time to read from the CD-ROM.
  - In many cases, installing nonessential files on the hard disk is not necessary, but may provide benefits such as increased performance. Give the user the option of deciding how to make the tradeoff between the costs and benefits of hard disk storage.
  - Provide a way to uninstall any components that were placed on the hard disk. For more information about uninstalling an application, see *Installing Applications*.
  - If your application caches data, give the user some control over it. Include options in the startup application such as setting a limit on the maximum amount of cached data that will be stored on the hard disk, or having the application discard any cached data when it terminates.

Autorun.inf Commands

This document is a reference for the commands that can be used in an Autorun.inf file.

icon

The **icon** command specifies an icon which represents the AutoPlay-enabled drive in the Windows user interface.

```
icon=iconfilename[,index]
```

Parameters

iconfilename

Name of an .ico, .bmp, .exe, or .dll file containing the icon information. If a file contains more than one icon, you must also specify zero-based index of the icon.

Remarks

The icon represents the AutoPlay-enabled drive in the Microsoft Windows user interface. For instance, in Windows Explorer, the drive will have this icon instead of the standard drive icon. The icon's file must be in the same directory as the file specified by the open command.

The following example specifies the second icon in the MyProg.exe file.

```
icon=MyProg.exe,1
```

label

The **label** command specifies a text label to represent the AutoPlay-enabled drive in the Windows user interface.

```
label=LabelText
```

Parameters

LabelText
A text string containing the label. It may contain spaces.

Remarks

The label represents the AutoPlay-enabled drive in the Microsoft Windows user interface. If an icon is also specified, the label will be displayed below it.

The following example specifies "My Drive Label" as the drive's label.

```
label=My Drive Label
```

open

The **open** command specifies the path and file name of the application that AutoPlay runs when a user inserts a disc in the drive.

```
open=[exepath\]exefile [param1 [param2] ...]
```

Parameters

exefile
Fully qualified path of an executable file that will be run when the compact disc is inserted. If only a file name is specified, it must be in drive's root directory. To locate the file in a subdirectory, you must specify a path. You can also include one or more command-line parameters to be passed to the startup application.

Remarks

See *Tips for Writing AutoPlay Startup Applications* for further discussion of startup applications.

shellexecute

The **shellexecute** command specifies an application or data file that AutoPlay will use to call **ShellExecuteEx**.

```
shellexecute=[filepath\]filename[param1, [param2]...]
```

Parameters

filepath
A string containing the fully-qualified path of the directory that contains the data or executable file. If no path is specified, the file must be in the drive's root directory.

filename
A string containing the file's name. If it is an executable file, it will be launched. If it is a data file, it must be a member of a file class. **ShellExecuteEx** will launch the default command associated with the file class.

paramx
Any additional parameters that should be passed to **ShellExecuteEx**.

Remarks

This command is similar to open, but it allows you to use file association information to run the application.

shell

The **shell** command specifies a default command for the drive's context menu.

```
shell=verb
```

Parameters

verb
The verb that corresponds to the command. The verb and its associated command must be defined in the Autorun.inf file with a shell/verb command.

Remarks

When a user right-clicks the drive icon, a context menu will appear. If an Autorun.inf file is present, the default context menu command is taken from it. This command is also executed when the user double-clicks the drive's icon.

To specify the default context menu command, first define its verb, command string, and menu text with shell/verb. Then use shell to make it the default context menu command. Otherwise, the default menu item text will be "AutoPlay", which will launch the application specified by the open command.

shell\verb

The **shell\verb** command adds a custom command to the drive's context menu.

```
shell\verb\command=Filename.exe
shell\verb=MenuText
```

Parameters

verb
The command's verb. The **shell*verb*\command** command associates the verb with an executable file. Verbs must not contain embedded spaces. By default, *verb* is the text that is displayed in the context menu.

Filename.exe
The path and file name of the application that performs the command.

MenuText
This parameter specifies the text that is displayed in the context menu. If it is omitted, *verb* is displayed. *MenuText* can be mixed-case and may contain spaces. You can set a shortcut key for the menu item by putting an ampersand (&) in front of the letter.

Remarks

When a user right-clicks the drive icon, a context menu will appear. Adding **shell/verb** commands to the drive's Autorun.inf file allows you to add commands to this context menu.

There are two parts to this command, which must be on separate lines. The first part is **shell/*verb*/command**, and is required. It associates a string, called a *verb*, with the application to be launched when the command is run. The second part is the **shell/*verb*** command, and is optional. It can be included to specify the text that is displayed in the context menu.

To specify a default context menu command, define the verb with **shell/verb**, and make it the default command with shell.

The following sample Autorun.inf fragment associates the *readit* verb with the command string "Notepad abc\readme.txt". The menu text is "Read Me", and 'M' is defined as the item's shortcut key. When the user selects this command, the drive's abc\readme.txt file will be opened with Notepad.

```
shell\readit\command=notepad abc\readme.txt
shell\readit=Read &Me
```

Enabling and Disabling AutoPlay

There are many situations where AutoPlay may need to be temporarily or persistently disabled. For example, AutoPlay might interfere with the operation of a running application and need to be disabled for the duration. The system provides several ways to disable AutoPlay.

Suppressing AutoPlay Programmatically

There are a variety of situations where AutoPlay may need to be suppressed programmatically. Two examples are:

- Your application has a setup program that requires the user to insert another disc that may contain an Autorun.inf file.
- During the operation of your application, the user may need to insert another disc that may contain an Autorun.inf file.

In either case, you will normally not want to launch another application while the original is in progress.

Users can manually suppress AutoPlay by holding down the SHIFT key when they insert the CD-ROM. However, it is usually preferable to handle this operation programmatically rather than depending on the user.

With systems that have shell version 4.70 and later, Microsoft Windows sends a "QueryCancelAutoPlay" message to the foreground window. Your application can respond to this message to suppress AutoPlay. This approach is used by system utilities such as the Open common dialog box to disable AutoPlay. You will not get a "QueryCancelAutoPlay" message with versions of Windows 95 that do not have the Internet Explorer 4.0 integrated shell installed.

The following code fragments illustrate how to set up and handle this message. Your application must be running in the foreground window. First, register "QueryCancelAutoPlay" as a Windows message:

```
uMessage = RegisterWindowMessage
(TEXT("QueryCancelAutoPlay"));
```

Your application's window must be in the foreground to receive this message. The message handler should return TRUE to cancel AutoPlay and FALSE to enable it. The following code fragment illustrates how to use this message to disable AutoPlay.

```
UINT g_uQueryCancelAutoPlay = 0;

LRESULT WndProc(HWND hwnd, UINT uMsg,  WPARAM wParam, LPARAM lParam)
{
    switch (uMsg) {
    ...
        default:
        if (!g_uQueryCancelAutoPlay)
        {
            g_uQueryCancelAutoPlay = RegisterWindowMessage
(TEXT("QueryCancelAutoPlay"));
        }
        if (uMsg && uMsg == g_uQueryCancelAutoPlay)
        {
            return TRUE;      // cancel auto-play
        }
    }
}
```

If your application is using a dialog box and needs to respond to a "QueryCancelAutoPlay" message, it cannot simply return TRUE or FALSE. Instead, call SetWindowLong with *nIndex* set to DWL_MSGRESULT. Set the *dwNewLong* parameter to TRUE to cancel AutoPlay and FALSE to enable it. For example, the following sample dialog procedure cancels AutoPlay when it receives a "QueryCancelAutoPlay" message.

```
UINT g_uQueryCancelAutoPlay = 0;

BOOL DialogProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch (uMsg) {
    ...
    default:
        if (!g_uQueryCancelAutoPlay)
        {
            g_uQueryCancelAutoPlay = RegisterWindowMessage
(TEXT("QueryCancelAutoPlay"));
        }
        if (uMsg == g_uQueryCancelAutoPlay)
        {
            SetWindowLong(hDlg, DWL_MSGRESULT, TRUE);
            return 1;
        }
    }
}
```

Using the Registry to Disable AutoPlay

There are two registry values that can be used to persistently disable AutoPlay: NoDriveAutoRun and NoDriveTypeAutoRun. The first value disables AutoPlay for specified drive letters and the second disables AutoPlay for a class of drives. If either of these values is set to disable AutoPlay for a particular device, it will be disabled.

Note The NoDriveAutoRun and NoDriveTypeAutoRun values should only be modified by system administrators to change the value for the entire system for testing or administrative purposes. Applications should not modify these values, as there is no way to reliably restore them to their original values.

The NoDriveAutoRun value disables AutoPlay for specified drive letters. It is a **REG_DWORD** data value, found under the **HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer** key. The first bit of the value corresponds to A:, the second to B:, and so on. To disable AutoPlay for one or more drive letters, set the corresponding bits. For example, to disable the A: and C: drives, set NoDriveAutoRun to 0x00000005.

The NoDriveTypeAutoRun value disables AutoPlay for a class of drives. It is a **REG_DWORD** or 4-byte **REG_BINARY** data value, found under the **HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer** key. By setting the bits of this value's first byte, different drives can be excluded from working with AutoPlay.

The following table gives the bits and bitmask constants, that can be set in the first byte of NoDriveTypeAutoRun to disable AutoPlay for a particular drive type. For Microsoft Windows NT and Windows 2000, you must restart Windows Explorer before the changes take effect.

Bit number	Bitmask constant	Description
0x04	DRIVE_REMOVEABLE	Disk can be removed from drive (such as a floppy disk).
0x08	DRIVE_FIXED	Disk cannot be removed from drive (a hard disk).
0x10	DRIVE_REMOTE	Network drive.
0x20	DRIVE_CDROM	CD-ROM drive.
0x40	DRIVE_RAMDISK	RAM disk.

AutoPlay for Other Types of Storage Media

AutoPlay is primarily intended for public distribution of applications on CD-ROM and DVD-ROM. However, it is often useful to enable AutoPlay on other types of removable storage media. This feature is typically used simplify the debugging of AutoRun.inf files. AutoPlay only works on removable storage devices when the following criteria are met:

- The device must have AutoPlay-compatible drivers. To be AutoPlay-compatible, a driver must notify the system that a disk has been inserted by sending a **WM_DEVICECHANGE** message.
- The root directory of the inserted media must contain an Autorun.inf file.
- The device must not have AutoPlay disabled through the registry.
- The foreground application has not suppressed AutoPlay.

Note This feature should not be used to distribute applications on floppy disks. Because implementing AutoPlay on a floppy disk provides an easy way to spread computer viruses, users should be suspicious of any publicly distributed floppy disk that contains an Autorun.inf file.

Normally, AutoPlay starts automatically, but it can also be started manually. If the device meets the criteria listed above, the drive letter's context menu will include an **AutoPlay** command. To run AutoPlay manually, either right-click the drive icon and select **AutoPlay** from the context menu or double-click the drive icon. If the drivers are not AutoPlay-compatible, the context menu will not have an **AutoPlay** item and AutoPlay can not be started.

AutoPlay-compatible drivers are provided with some floppy disk drives, as well as some other types of removable media such as Compact Flash cards. AutoPlay also works with network drives that are mapped to a drive letter with Windows Explorer or mounted with the Microsoft Management Console (MMC). As with mounted hardware, a mounted network drive must have an Autorun.inf file in its root directory, and must not be disabled through the registry.

Note Please see the companion DVD at the back of *Volume 1: Base Services* for information about advanced shell topics.

CHAPTER 7

Shell Interfaces

Shell Interface Overview

As mentioned previously, a large number of standard interfaces are defined by the shell and declared in shlobj.h. Most fall into one of three general categories:

- Interfaces that are exposed by folders and available for use by applications.
- Interfaces that are exposed by folders but used only by the shell. These interfaces generally are not used by applications, but must be implemented by namespace extensions.
- Interfaces that are used by shell extensions, such as shell extension handlers or band objects. These interfaces are covered in the discussion of the associated shell extension.

This chapter provides comprehensive information for Shell Interfaces. The standard structure of the sections in this chapter is that the name of the interfaces is presented as a header (such as the first section, **IACList**), followed by the interface's associated methods (such as **IACList::Expand**). Some interfaces have only one interface, others have many; regardless, the presentation of the interface overview, followed by its methods, is consistent throughout the chapter. Hopefully, you'll find this presentation of Shell Interfaces intuitive, and it'll help you get your Shell programming information quickly and easily—because that is the intent behind this chapter's structure.

If you have feedback about this structure, or suggestions on how to make it more intuitive or more accessible, you can send me an e-mail message at *winprs@microsoft.com.* While I can't guarantee I'll answer (nor can I provide technical or usage support), you can be assured that I'll read it and consider your suggestions.

Shell Interfaces

IACList

The **IACList** interface is designed to improve the efficiency of autocompletion when the candidate strings are organized in a hierarchy.

Autocompletion normally requires three components:

- The autocompletion client. This client is a window, such as a dialog box, that hosts the edit control.

- The autocompletion object (CLSID_AutoComplete). This object is provided by the system, and handles the user interface, parsing, and background thread management.
- The autocompletion list object. This object is responsible for providing lists of candidate strings to the autocompletion object.

A simple autocompletion list object needs to export only **IEnumString** in addition to **IUnknown**. When the user starts to enter characters in the edit box, the autocompletion object calls the list object's **IEnumString** interface to enumerate the list of strings that can be used to complete the partial string. The list object is responsible for maintaining a namespace and deciding which of those strings are relevant.

The simplest approach a list object can take is to return every string in its namespace every time the autocompletion object makes a request. For a discussion of how to implement this type of list object, see **IAutoComplete**. However, this approach is practical only if the namespace is relatively small. When large numbers of strings are involved, the list object needs to restrict itself to a small subset of the namespace.

The **IACList** interface is exported by autocompletion list objects to help them choose a sensible subset of strings from a hierarchically organized namespace. With a large namespace, this procedure substantially increases the efficiency of autocompletion. The basic procedure is:

1. The AutoComplete object calls the list object's **IEnumString** interface. The list object returns the names of the top-level items in the hierarchy. For instance, if the namespace consists of every file and folder on the C: drive, the list object returns the fully qualified paths of the folders and files contained by the C:\ directory.
2. Users continue to type until they enter a delimiter. The '\' and '/' characters are recognized as delimiters by the autocompletion object.
3. The autocompletion object calls the list object's **IACList::Expand** method and passes it the current partial string.
4. The autocomplete object then calls the list object's **IEnumString** interface again to request a new list of strings. If the partial string matches one of the top-level items in the namespace, the list object returns the names of the items that fall immediately under the selected item. For instance, if the user has entered "C:\Program Files\", the list object returns the names of the files and folders contained in that directory. If the name passed to **IACList::Expand** does not match any top-level item, the list object can stop returning strings until the autocomplete object calls **IACList::Expand** with a string that is in the list object's namespace.
5. The process continues until the user selects a string, typically by pressing the ENTER key.

When to Implement

This interface is implemented by autocompletion list objects that maintain a hierarchically organized namespace. It is essential for acceptable performance if the namespace is large.

When to Use

Applications normally do not use this interface.

Methods

IACList exposes the following method in addition to **IUnknown**:

Expand	Requests that the autocomplete object expand a string.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IACList::Expand

Requests that the autocompletion client generate candidate strings associated with a specified item in its namespace.

```
HRESULT Expand(
    LPCOLESTR pszExpand
);
```

Parameters

pszExpand

[in] NULL-terminated string to be expanded by the autocomplete object.

Return Values

Returns S_OK if successful, or an OLE error code otherwise.

Remarks

The autocomplete object calls this method when a delimiter is entered in the edit control. If the string pointed to by *pszExpand* matches an item in the autocompletion client's namespace, the client generates strings for those items that fall immediately under *pszExpand* in its namespace hierarchy. The client returns those strings next time the autocompletion object calls the client's **IEnumString** interface.

Assume, for example, that the client's namespace consists of all the files and folders on the C: drive, and *pszExpand* is set to "C:\Program Files\". The client should generate a list of strings corresponding to the fully qualified paths of the files and subfolders of "C:\Program Files\".

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IACList

IACList2

The **IACList2** interface extends the **IACList** interface to allow clients of an autocomplete object to retrieve and set option flags.

When to Implement

Autocompletion clients implement this interface to allow the autocomplete object to retrieve and set options. The options are basically a request that the client generate a list with the names of all the files and subfolders contained by one or more specified folders. The autocomplete object then calls the client's **IEnumString** interface to request the strings.

When to Use

This interface normally is not used by applications.

Methods

IACList2 exposes the following methods in addition to **IUnknown**:

GetOptions	Retrieves the current autocomplete options.
SetOptions	Sets the current autocomplete options.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IACList2::GetOptions

Retrieves the current autocomplete options.

```
HRESULT GetOptions(
    DWORD *pdwFlag
);
```

Parameters

pdwFlag

[out] Address of a value that will hold the current option flag when the method returns. It can be a combination of the following values:

ACLO_CURRENTDIR	Enumerate the current working directory.
ACLO_DESKTOP	Enumerate the Desktop folder.
ACLO_FAVORITES	Enumerate the Favorites folder.
ACLO_MYCOMPUTER	Enumerate the My Computer folder.
ACLO_NONE	Do not enumerate anything.

Return Values

Returns S_OK if successful, or an OLE error code otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IACList2

IACList2::SetOptions

Sets the current autocomplete options.

```
HRESULT SetOptions(
    DWORD dwFlag
);
```

Parameters

dwFlag

[in] New option flags. They are used to ask the client to include the names of the files and subfolders of the specified folders the next time its **IEnumString** interface is called. It can be one or more of the following flags:

ACLO_CURRENTDIR	Enumerate the current working directory.
ACLO_DESKTOP	Enumerate the Desktop folder.
ACLO_FAVORITES	Enumerate the Favorites folder.
ACLO_MYCOMPUTER	Enumerate the My Computer folder.
ACLO_NONE	Do not enumerate anything.

Return Values

Returns S_OK if successful, or an OLE error code otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IACList2

IActiveDesktop Interface

Allows a client program to manage the desktop items and wallpaper on a local computer.

Methods

AddDesktopItem	Adds a desktop item.
AddDesktopItemWithUI	Adds a desktop item to the Active Desktop using a user interface.
AddUrl	Adds the desktop item associated with the specified URL.
ApplyChanges	Applies changes to the Active Desktop.
GenerateDesktopItemHtml	Generates a generic HTML page containing the given desktop item.
GetDesktopItem	Retrieves the specified desktop item.
GetDesktopItemByID	Retrieves the desktop item that matches the given identification.

GetDesktopItemBySource	Retrieves a desktop item using its source URL.
GetDesktopItemCount	Retrieves a count of the desktop items.
GetDesktopItemOptions	Retrieves the desktop item's options.
GetPattern	Retrieves the pattern being used currently.
GetWallpaper	Retrieves the wallpaper being used currently.
GetWallpaperOptions	Retrieves the wallpaper options.
ModifyDesktopItem	Modifies the desktop item.
RemoveDesktopItem	Removes the specified desktop item from the desktop.
SetDesktopItemOptions	Sets the item's options.
SetPattern	Sets the ActiveDesktop pattern.
SetWallpaper	Sets the wallpaper for the Active Desktop.
SetWallpaperOptions	Sets the wallpaper options.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IActiveDesktop::AddDesktopItem Method

Adds a desktop item.

```
HRESULT AddDesktopItem(
    LPCOMPONENT pcomp,
    DWORD dwReserved
);
```

Parameters

pcomp
Address of the **COMPONENT** structure associated with the item to be added.

dwReserved
Reserved. Must be set to zero.

Return Value

Returns one of the following values:

E_FAIL	Failed to add the desktop item or an instance of the desktop item already exists on the Active Desktop.
E_INVAILDARG	One or more of the parameters were invalid.
S_OK	Desktop item has been added successfully.

Remarks

The desktop item is added to the desktop, but it does not save it to the registry. The client application must call **IActiveDesktop::ApplyChanges** separately to update the registry.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

Active Desktop Interface (overview), **IActiveDesktop**

IActiveDesktop::AddDesktopItemWithUI Method

Adds a desktop item to the Active Desktop after displaying user interfaces that confirm the addition of the desktop item, verify security zone permissions, and ask if the user wants to create a subscription.

```
HRESULT AddDesktopItemWithUI(
    HWND hwnd,
    LPCOMPONENT pcomp,
    DWORD dwFlags
);
```

Parameters

hwnd
Handle of the parent window. If NULL, the desktop item will be added without displaying any user interface.

pcomp
Address of the **COMPONENT** structure containing the details of the desktop item to be added.

dwFlags

Unsigned long integer value that contains the flags that control how the desktop item is added. Can be one of the following values:

Flag	Description
DTI_ADDUI_DEFAULT	Do default action. Identical to using zero.
DTI_ADDUI_DISPSUBWIZARD	Activate the subscription wizard to allow the user to subscribe to this desktop item.
DTI_ADDUI_POSITIONITEM	Instructs the system to look at the **COMPPOS** structure passed to the **cpPos** member of the **COMPONENT** structure to make sure that the values are within reasonable limits. This value was added for Microsoft Internet Explorer 5.0.

Return Value

Returns one of the following values:

E_FAIL	Failed to add the desktop item or an instance of the desktop item already exists on the Active Desktop.
E_INVAILDARG	One or more of the parameters were invalid.
S_OK	If the ADDURL_SILENT flag has been set, the desktop item either has been added successfully or already exists on the Active Desktop. Otherwise, the desktop item has been added successfully.

Remarks

This method creates a second instance of the Active Desktop to add the desktop item, so the desktop item will not appear in the current instance. The application must call the **IUnknown::Release** method on this **IActiveDesktop** interface and use the **CoCreateInstance** function to get the **Active Desktop** object with the newly added component.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

Active Desktop Interface Overview, IActiveDesktop

IActiveDesktop::AddUrl Method

Adds the desktop item associated with the specified URL.

```
HRESULT AddUrl(
    HWND hwnd,
    LPCWSTR pszSource,
    LPCOMPONENT pcomp,
    DWORD dwFlags
);
```

Parameters

hwnd

Handle to the parent window for the user interface.

pszSource

Address of a string value that contains the URL of the desktop item.

pcomp

Address of the **COMPONENT** structure containing the details of the desktop item to be added.

dwFlags

Unsigned long integer value that controls this method. Can be set to ADDURL_SILENT to add a desktop item without displaying any user interfaces.

Return Value

Returns one of the following values:

E_FAIL	Failed to add the desktop item or an instance of the desktop item already exists on the Active Desktop.
E_INVAILDARG	One or more of the parameters were invalid.
S_OK	If the ADDURL_SILENT flag has been set, the desktop item either has been added successfully or already exists on the Active Desktop. Otherwise, the desktop item has been added successfully.

Remarks

By default, this method will display some user interface and then add the desktop item to the Active Desktop. Like **IActiveDesktop::AddDesktopItem**, the client application must call **IActiveDesktop::ApplyChanges** to have the changes saved to the registry.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

Active Desktop Interface Overview, **IActiveDesktop**

IActiveDesktop::ApplyChanges

Applies changes to the Active Desktop and saves them in the registry.

```
HRESULT ApplyChanges(
    DWORD dwFlags
);
```

Parameters

dwFlags

Unsigned long integer value that contains the changes to be applied. Can be one of the following values:

AD_APPLY_ALL

AD_APPLY_BUFFERED_REFRESH

AD_APPLY_DYNAMICREFRESH

AD_APPLY_FORCE

AD_APPLY_HTMLGEN

AD_APPLY_REFRESH

AD_APPLY_SAVE

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IActiveDesktop::GenerateDesktopItemHtml

Generates a generic HTML page containing the given desktop item.

```
HRESULT GenerateDesktopItemHtml(
    LPCWSTR pwszFileName,
    LPCOMPONENT pcomp,
    DWORD dwReserved
);
```

Parameters

pwszFileName
String value containing the name to store the HTML file under.

pcomp
Address of the **COMPONENT** structure of the desktop item to insert in the HTML page.

dwReserved
Reserved. Must be set to zero.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IActiveDesktop::GetDesktopItem

Retrieves the specified desktop item.

```
HRESULT GetDesktopItem(
    int nComponent,
    LPCOMPONENT pcomp,
    DWORD dwReserved
);
```

Parameters

nComponent
Unsigned long integer value that contains the desktop item's index. The index values start at zero. Use **IActiveDesktop::GetDesktopItemCount** to get a count on the total number of desktop items.

pcomp

Address of the **COMPONENT** structure of the retrieved desktop item.

dwReserved

Reserved. Must be set to zero.

Remarks

The index values will change as desktop items are added and removed from the Active Desktop. Applications cannot assume that an index value always will be associated with a particular desktop item.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with
Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0
or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IActiveDesktop::GetDesktopItemByID

Retrieves the desktop item that matches the given identification.

```
HRESULT GetComponentByID(
    DWORD dwID,
    LPCOMPONENT pcomp,
    DWORD dwReserved
);
```

Parameters

dwID

Unsigned long integer value that contains the desktop item's identification.

pcomp

Address of the **COMPONENT** structure of the retrieved desktop item.

dwReserved

Reserved. Must be set to zero.

Remarks

The desktop item's identification is returned in the **dwID** member of the **COMPONENT** structure returned from the **IActiveDesktop::GetDesktopItem** method. This identification is valid only until the **IActiveDesktop::ApplyChanges** method is called.

Applications that need to get the same desktop item consistently should enumerate the desktop items using the **IActiveDesktop::GetDesktopItem** and **IActiveDesktop::GetDesktopItemCount** methods.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IActiveDesktop::GetDesktopItemBySource

Retrieves a desktop item using its source URL.

```
HRESULT GetDesktopItemBySource (
    LPCWSTR pszSource,
    LPCOMPONENT pcomp,
    DWORD dwReserved
);
```

Parameters

pszSource
String value containing the source URL of the desktop item.

pcomp
Address of the **COMPONENT** structure that will be used to store the details about the desktop item. The size of the structure must be initialized in order for it to work properly.

dwReserved
Reserved. Must be set to zero.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IActiveDesktop::GetPattern

Retrieves the pattern being used currently.

```
HRESULT GetPattern(
    LPWSTR pwszPattern,
    UINT cchPattern,
    DWORD dwReserved
);
```

Parameters

pwszPattern

Address of a string value that contains a string of decimals whose bit pattern represents a picture. Each decimal represents the on/off state of the 8 pixels in that row.

cchPattern

Unsigned long integer value that contains the size of the string.

dwReserved

Reserved. Must be set to zero.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IActiveDesktop::GetDesktopItemCount

Retrieves a count of the desktop items.

```
HRESULT GetDesktopItemCount(
    LPINT lpiCount,
    DWORD dwReserved
);
```

Parameters

lpiCount

Address of an integer value that contains the count.

dwReserved

Reserved. Must be set to zero.

Remarks

The *lpiCount* value can be used to enumerate all desktop items. The index values start at zero and go to *lpiCount* minus one.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IActiveDesktop::GetDesktopItemOptions

Retrieves the desktop item's options.

```
HRESULT GetDesktopItemOptions(
    LPCOMPONENTSOPT pco
    DWORD dwReserved
);
```

Parameters

pco
Address of the **COMPONENTSOPT** structure containing the options that are set currently.

dwReserved
Reserved. Must be set to zero.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IActiveDesktop::GetWallpaper

Retrieves the wallpaper being used currently.

```
HRESULT GetWallpaper(
    LPWSTR pwszWallpaper,
    UINT cchWallpaper,
    DWORD dwReserved
);
```

Parameters

pwszWallpaper
String value that contains the file name of the wallpaper.

cchWallpaper
Unsigned integer value that contains size of the string.

dwReserved
Reserved. Must be set to zero.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IActiveDesktop::GetWallpaperOptions

Retrieves the wallpaper options.

```
HRESULT GetWallpaperOptions(
    LPWALLPAPEROPT pwpo,
    DWORD dwReserved
);
```

Parameters

pwpo
Address of a **WALLPAPEROPT** structure containing the options set currently.

dwReserved
Reserved. Must be set to zero.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IActiveDesktop::ModifyDesktopItem

Modifies the desktop item.

```
HRESULT ModifyDesktopItem(
    LPCCOMPONENT pcomp,
    DWORD dwFlags
);
```

Parameters

pcomp

Address of the **COMPONENT** structure containing the modifications. The desktop item associated with the **wszSource** member of the structure will be modified.

dwFlags

Unsigned long integer value containing the flags used for the modification. This can be one of the following values:

- **COMP_ELEM_ALL**
- **COMP_ELEM_CHECKED**
- **COMP_ELEM_CURITEMSTATE**
- **COMP_ELEM_FRIENDLYNAME**
- **COMP_ELEM_NOSCROLL**
- **COMP_ELEM_ORIGINAL_CSI**
- **COMP_ELEM_POS_LEFT**
- **COMP_ELEM_POS_TOP**
- **COMP_ELEM_POS_ZINDEX**
- **COMP_ELEM_RESTORED_CSI**
- **COMP_ELEM_SIZE_HEIGHT**
- **COMP_ELEM_SIZE_WIDTH**
- **COMP_ELEM_TYPE**

Remarks

The client application must call **IActiveDesktop::ApplyChanges** separately to update the registry. For example, to change the friendly name, first call this function with either COMP_ELEM_FRIENDLYNAME or COMP_ELEM_ALL in the *dwFlags* member of **COMPONENT**. Then call **IActiveDesktop::ApplyChanges**.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IActiveDesktop::RemoveDesktopItem

Removes the specified desktop item from the desktop.

```
HRESULT RemoveDesktopItem(
    LPCCOMPONENT pcomp,
    DWORD dwReserved
);
```

Parameters

pcomp
Address of the **COMPONENT** structure that specifies the item to be removed. The desktop item associated with the **wszSource** member of the structure will be removed.

dwReserved
Reserved. Must be set to zero.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IActiveDesktop::SetDesktopItemOptions

Sets the item's options.

```
HRESULT SetDesktopItemOptions(
    LPCCOMPONENTSOPT pcomp,
    DWORD dwReserved
);
```

Parameters

pcomp
Address of the **COMPONENTSOPT** structure that contains the options to set.

dwReserved
Reserved. Must be set to zero.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IActiveDesktop::SetPattern

Sets the Active Desktop pattern.

```
HRESULT SetPattern(
    LPCWSTR pwszPattern,
    DWORD dwReserved
);
```

Parameters

pwszPattern
Address of a string value that contains a string of decimals whose bit pattern represents a picture. Each decimal represents the on/off state of the 8 pixels in that row.

dwReserved
Reserved. Must be set to zero.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).

Windows CE: Unsupported.
Header: Declared in shlobj.h.

IActiveDesktop::SetWallpaper

Sets the wallpaper for the Active Desktop.

```
HRESULT SetWallpaper(
    LPCWSTR pwszWallpaper,
    DWORD dwReserved
);
```

Parameters

pwszWallpaper
String value containing the file name of the wallpaper to be set.

dwReserved
Reserved. Must be set to zero.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IActiveDesktop::SetWallpaperOptions

Sets the wallpaper options.

```
HRESULT SetWallpaperOptions(
    LPCWALLPAPEROPT pwpo,
    DWORD dwReserved
);
```

Parameters

pwpo
Address of the **WALLPAPEROPT** structure containing the options to be set.

dwReserved
Reserved. Must be set to zero.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IAsyncOperation

Allows interfaces that are normally synchronous to function asynchronously.

This interface is exported primarily by the data objects used with drag-drop and Clipboard operations. Normally, such operations are synchronous. However, if data rendering will be time-consuming, **IASyncOperation** can be used to allow data extraction to take place on a background thread.

When to Implement

IAsyncOperation is an optional interface that is implemented by a data object. It allows the drop target to negotiate with the drop source to extract data from the data object asynchronously.

When to Use

Drop sources and targets use this interface when they wish to have a lengthy data extraction processes handled by a background thread.

Methods

IAsyncOperation exposes the following methods in addition to **IUnknown**:

EndOperation Notifies the data object that that asynchronous data extraction has ended.

GetAsyncMode Called by a drop target to determine whether the data object supports asynchronous data extraction.

InOperation Called by the drop source to determine whether the target is extracting data asynchronously.

SetAsyncMode Called by a drop source to specify whether the data object supports asynchronous data extraction.

StartOperation Called by a drop target to indicate that asynchronous data extraction is starting.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IASyncOperation::EndOperation

Notifies the data object that that asynchronous data extraction has ended.

```
HRESULT EndOperation(
    HRESULT hResult,
    IBindCtx *pbcReserved,
    DWORD dwEffects
);
```

Parameters

hResult

[in] An HRESULT value that indicates the outcome of the data extraction. Set to S_OK, if successful, or an OLE error code otherwise.

pbcReserved

[in] Reserved. Set to NULL.

dwEffects

[in] A **DROPEFFECT** value that indicates the result of an optimized move. This should be the same value that would be passed to the data object as a CFSTR_PERFORMMEDDROPEFFECT format with a normal data extraction operation.

Return Values

Returns S_OK if successful or an OLE error value otherwise.

Remarks

EndOperation retrieves the **IAsyncOperation** pointer stored by **IAsyncOperation::SetAsyncMode**, and passes its parameter values to that interface's **IAsyncOperation::EndOperation** method. **EndOperation** then releases the **IAsyncOperation** pointer.

EndOperation is also responsible for any clean-up operations that are needed. When finished, **EndOperation** should notify the drop source through a private interface.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IASyncOperation

IASyncOperation::GetAsyncMode

Called by a drop target to determine whether the data object supports asynchronous data extraction.

```
HRESULT GetAsyncMode(
    BOOL *pfIsOpAsync
);
```

Parameters

pfIsOpAsync

[out] A Boolean value that is set to VARIANT_TRUE to indicate that an asynchronous operation is supported, VARIANT_FALSE otherwise.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

The purpose of this method is to give the drop target the value of the **IAsyncOperation::SetAsyncMode** method's *fDoOpAsync* parameter. This parameter is set to VARIANT_FALSE, by default. If the data object supports asynchronous data extraction, it must call **IAsyncOperation::SetAsyncMode** and set *fDoOpAsync* to VARIANT_TRUE.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IASyncOperation

IASyncOperation::InOperation

Called by the drop source to determine whether the target is extracting data asynchronously.

```
HRESULT InOperation(
    BOOL *pfInAsyncOp
);
```

Parameters

pfInAsyncOp

[out] Set to VARIANT_TRUE if data extraction is being handled asynchronously, or VARIANT_FALSE otherwise.

Return Values

Returns S_OK if successful or an OLE error value otherwise.

Remarks

This method is called by the drop source after **DoDragDrop** returns. *pfInAsyncOp* should be set to VARIANT_TRUE only if the drop target has called **IAsyncOperation::StartOperation**.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IASyncOperation

IASyncOperation::SetAsyncMode

Called by a drop source to specify whether the data object supports asynchronous data extraction.

```
HRESULT SetAsyncMode(
    BOOL fDoOpAsync
);
```

Parameters

fDoOpAsync

[in] A Boolean value that is set to VARIANT_TRUE to indicate that an asynchronous operation is supported, VARIANT_FALSE otherwise. It's default value is VARIANT_FALSE.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

This method is called by the drop source to indicate that the data object supports asynchronous data extraction. Store the *fDoOpAsync* for future use by **IAsyncOperation::GetAsyncMode**. The drop target determines whether asynchronous data extraction is supported by calling **IAsyncOperation::GetAsyncMode** to retrieve the *fDoOpAsync* value.

If *fDoOpAsync* is set to VARIANT_TRUE, **SetAsyncMode** must call **IAsyncOperation::AddRef**, and store the interface pointer for use by **IAsyncOperation::EndOperation**.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IASyncOperation

IASyncOperation::StartOperation

Called by a drop target to indicate that asynchronous data extraction is starting.

```
HRESULT StartOperation(
    IBindCtx *pbcReserved
);
```

Parameters

pbcReserved

[in]Reserved. Set this value to NULL.

Return Values

Returns S_OK if successful or an OLE error value otherwise.

Remarks

The drop target calls this method to notify the data object that asynchronous data extraction is starting. The method should store this information, so that it can be returned by **IAsyncOperation::InOperation**. Once **StartOperation** has been called, the drop target returns the **IDropTarget::Drop** call as it would for normal synchronous data extraction.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IASyncOperation

IAutoComplete

The **IAutoComplete** interface is exposed by the autocomplete object (CLSID_AutoComplete). It allows applications to initialize, enable, and disable the object.

Autocompletion expands strings that have been entered partially in an edit control into complete strings. For example, when a user starts to enter a URL in the Address edit control that is embedded in the Internet Explorer toolbar, autocompletion expands the string into one or more complete URLs that are consistent with the existing partial string. A partial URL string such as “mic” might be expanded to “http://www.microsoft.com” or “http://www.microsoft.com/windows”. Autocompletion is used typically with edit controls or with controls that have an embedded edit control such as the comboboxex control.

Autocompletion has two modes for displaying the completed string. The modes are independent, so you can enable either or both. To specify the mode, call **IAutoComplete2::SetOptions**.

- In *autoappend* mode, autocompletion appends the remainder of the most likely candidate string to the existing characters, highlighting the appended characters. The edit control behaves as if the user had entered the entire string manually and then highlighted the appended characters. If the user continues to enter characters, they are added to the existing partial string. If the user adds a character that is identical to the next highlighted character, the highlighting for that character will be turned off. The remaining characters will still be highlighted. If the user adds a character that does not match the next highlighted character, autocompletion will attempt to generate a new candidate string based on the larger partial string. It will append the remainder of the new candidate string to the current partial string, as before. If no candidate string can be found, only the typed characters will appear and the edit box will behave as it would without autocompletion. This process continues until the user accepts a string.

- In *autosuggest* mode, autocompletion displays a drop-down list, with one or more suggested complete strings, beneath the edit control. The user can select one of the suggested strings, usually by clicking it with the mouse, or continue typing. As typing progresses, the drop-down list may be modified, based on the current partial string. If you set the ACO_SEARCH flag in the *dwFlag* parameter of **IAutoComplete2::SetOptions**, a "Search for "XXX"" item will be added to the bottom of the drop-down list. It will be displayed even if there are no suggested strings. "XXX" will be set to the current partial string and will be updated as the user continues to type. If the user selects "Search for "XXX"", your application should launch a search engine to assist them.

The simplest way to implement autocompletion is to call **SHAutoComplete**. When this function is called for a system edit control, the control will autocomplete partially entered file-system paths or URLs. To enable autocompletion for other types of strings, or to have more control over how autocompletion works, you can use the underlying autocomplete object directly.

When to Implement

This interface normally is not implemented by applications. It is exposed by the shell's autocomplete object and used by applications.

When to Use

Use the **IAutoComplete** interface of the autocomplete object to initialize the object, and to enable or disable autocompletion.

To implement autocompletion for an edit control using the autocomplete object:

1. Implement a string list COM object that exports an **IEnumString** interface. This string list object is responsible for providing the list of strings that the autocomplete object will use as candidates for completed strings.
2. Create an instance of the autocomplete object with **CoCreateInstance**. Request a pointer to its **IAutoComplete** interface.
3. Call **IAutoComplete::Init**. Set the *hwndEdit* parameter to the window handle of the edit control. If the edit control is embedded in another control, you must get the handle to the edit control itself. For example, to get a handle to the edit control embedded in a **comboboxex control**, send a **CBEM_GETEDITCONTROL** message. Set the *punkACL* parameter of **IAutoComplete::Init** to the **IUnknown** pointer of the string list object.

4. If you do not want to use the default options, get a pointer to the autocomplete object's **IAutoComplete2** interface. Call **IAutoComplete2::SetOptions** to set the desired options.
5. The autocomplete object uses the **IUnknown** pointer of the string list object, passed as *punkACL* in step 4, to get a pointer to that object's **IEnumString** interface. The autocomplete object then calls that interface to generate its list of candidate strings. It selects strings from that list that are an acceptable match to the partial string in the control. In autoappend mode, the characters needed to complete the string are appended to the partial string and highlighted. In autosuggest mode, a drop-down box with a list of one or more possible strings is displayed below the edit control.
6. If the user accepts an autocompleted string, the edit control behaves as if the string had been entered manually.

Autocompletion is enabled by default. Applications only need to call **IAutoComplete::Enable** to disable autocompletion, or to reenable it if it has been disabled.

IAutoComplete exposes the following methods in addition to **IUnknown**:

Methods	Description
Enable	Enables or disables autocompletion.
Init	Initializes the autocomplete object.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in Shldisp.h.

+ See Also

IAutoComplete2

IAutoComplete::Enable

Enables or disables autocompletion.

```
HRESULT Enable(
    BOOL fEnable
);
```

Parameters

fEnable

[in] Value that is set to TRUE to enable autocompletion, or to FALSE to disable it.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

Autocompletion is enabled by default. Applications only need to call this method to disable autocompletion, or to reenable it if it has been disabled.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in Shldisp.h.

+ See Also

IAutoComplete

IAutoComplete::Init

Initializes the autocomplete object.

```
HRESULT Init(
    HWND hwndEdit,
    IUnknown *punkACL,
    LPCOLESTR pwszRegKeyPath,
    LPCOLESTR pwszQuickComplete
);
```

Parameters

hwndEdit

[in] Window handle for the system edit control that is to have autocompletion enabled.

punkACL

[in] Pointer to the **IUnknown** interface of the string list object that is responsible for generating candidates for the completed string. The object must expose an **IEnumString** interface.

pwszRegKeyPath

[in] Optional NULL-terminated Unicode string that gives the registry path, including the value name, where the format string is stored as a REG_SZ value. The autocomplete

object first looks for the path under **HKEY_CURRENT_USER**. If it fails, it then tries **HKEY_LOCAL_MACHINE**. For a discussion of the format string, see the definition of *pwszQuickComplete*.

pwszQuickComplete

[in] String that specifies the format to be used if the user enters some text and presses CTRL-ENTER. The autocomplete object treats *pwszQuickComplete* as a **sprintf** format string, and the text in the edit box as its associated argument, to produce a new string. For example, set *pwszQuickComplete* to "http://www.%s.com/". When a user enters "MyURL" into the edit box and presses CTRL-ENTER, the text in the edit box is updated to "http://www.MyURL.com/".

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in Shldisp.h.

+ See Also

IAutoComplete

IAutoComplete2

The **IAutoComplete2** interface extends **IAutoComplete**. It allows clients of the autocomplete object to retrieve and set a number of options that control how autocompletion operates.

When to Implement

This interface normally is not implemented by applications. It is exposed by the shell's autocomplete object and used by applications.

When to Use

Use this interface when you need to retrieve or set autocomplete options. The list of available options is given in the method references.

IAutoComplete2 exposes the following methods in addition to **IUnknown**:

Methods	Description
GetOptions	Retrieves the current autocomplete options.
SetOptions	Sets the current autocomplete options.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in Shldisp.h.

IAutoComplete2::GetOptions

Retrieves the current autocomplete options.

```
HRESULT GetOptions(
    DWORD *pdwFlag
);
```

Parameters

pdwFlag

[out] Flags that indicate the options that are set currently. This can be a combination of one or more of the following flags:

Flag	Description
ACO_AUTOAPPEND	Enable autoappend.
ACO_AUTOSUGGEST	Enable the autosuggest drop-down list.
ACO_FILTERPREFIXES	Do not match common prefixes, such as "www.", "http://", and so on.
ACO_NONE	No autocomplete.
ACO_RTLREADING	Normal windows display text left-to-right (LTR). Windows can be *mirrored* to display languages such as Hebrew or Arabic that read right-to-left (RTL). Normally, a control's text is displayed in the same direction as the text in its parent window. If ACO_RTLREADING is set, the text reads in the opposite direction from the text in the parent window.
ACO_SEARCH	Add a search item to the list of completed strings. Selecting this item launches a search engine.
ACF_UPDOWNKEYDROPSLIST	Use the UP ARROW and DOWN ARROW keys to display the autosuggest drop-down list.
ACO_USETAB	Use the TAB key to select an item from the drop-down list.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in Shldisp.h.

See Also

IAutoComplete2, **IAutoComplete2::SetOptions**

IAutoComplete2::SetOptions

Sets the current autocomplete options.

```
HRESULT SetOptions(
    DWORD dwFlag
);
```

Parameters

dwFlag

[in] Flags that allow an application to specify autocomplete options. This can be a combination of one or more of the following flags:

Flag	Description
ACO_AUTOAPPEND	Enable autoappend.
ACO_AUTOSUGGEST	Enable the autosuggest drop-down list.
ACO_FILTERPREFIXES	Do not match common prefixes, such as "www.", "http://", and so on.
ACO_NONE	No autocompletion.
ACO_RTLREADING	Normal windows display text from left to right (LTR). Windows can be *mirrored* to display languages such as Hebrew or Arabic that read from right to left (RTL). Normally, a control's text is displayed in the same direction as the text in its parent window. If ACO_RTLREADING is set, the text is read in the opposite direction from the text in the parent window.

(continued)

(continued)

Flag	Description
ACO_SEARCH	Add a search item to the dropdown list of completed strings. If this is item is selected, you should launch a search engine to assist the user.
ACF_UPDOWNKEYDROPSLIST	Use the UP ARROW and DOWN ARROW keys to display the autosuggest drop-down list.
ACO_USETAB	Use the TAB key to select an item from the drop-down list. This flag is disabled by default.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

The TAB key is disabled by default because it is used normally to move from control to control, not within a control. If you set the ACO_USETAB flag in *dwFlag*, users can move to a string in the drop-down list by pressing the TAB key. If the drop-down list is closed, the TAB key allows the user to move from control to control, as usual. The user can close the drop-down list by pressing the ESCAPE key.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in Shldisp.h.

+ See Also

IAutoComplete2

IColumnProvider

The Windows Explorer Details view normally displays several standard columns. Each column lists information, such as the file size or type, for each file in the current folder. There can also be a number of columns that the user can choose to display. When the user right-clicks one of the column headers, a list of the available columns is displayed in a dialog box. By creating a column provider object that exports the **IColumnProvider** interface, you can add custom columns to that dialog box for display by Windows Explorer. For example, a collection of files that contain music could use a column provider to display columns listing the artist and piece contained by each file.

A column provider is a global object that is called every time Windows Explorer displays the Details view. Windows Explorer queries all registered column providers for their column characteristics. If the user has selected one of the column provider's columns, Windows Explorer queries the column provider for the associated data for each file in the folder. It then displays all the selected columns.

Typically, column providers are used to display one or more custom columns for a particular file class. When a column provider receives a request for data, it provides it if the file is a member of its supported class. Otherwise, it ignores the request by returning S_FALSE.

Columns are identified by an **SHCOLUMNID** structure that contains an FMTID/PID pair. If possible, use existing FMTIDs and PIDs. If a folder contains files from more than one file class, the data from different classes can be merged into the same column. For instance, the Author PID from the summary information property set can be used for a wide variety of purposes. If you use a custom **SHCOLUMNID** structure, the column will display data only for those files that are members of the supported class. If the folder contains other files, their entries will be blank.

When to Implement

Implement an object that exports this interface when you want to have one or more custom columns displayed in the Windows Explorer Details view. Windows Explorer calls the interface methods to request the information it needs to display the column. The procedure used by Windows Explorer is:

1. Call **IColumnProvider::Initialize** to specify the folder to be displayed.
2. Call **IColumnProvider::GetColumnInfo** to get the column's characteristics.
3. If the column has been selected by the user, call **IColumnProvider::GetItemData** for each file in the folder to get the data that belongs in the file's column entry.

In addition to normal COM registration, the column provider object must also be registered with Windows Explorer. To do so, add a subkey named with the string form of the object's GUID to the **HKEY_CLASSES_ROOT\Folder\shellex\ColumnHandlers** key.

When to Use

This interface is called by Windows Explorer. It normally is not used by applications.

IColumnProvider exposes the following methods in addition to **IUnknown**:

Methods	Description
GetColumnInfo	Requests information about a column.
GetItemData	Requests column data for a specified file.
Initialize	Initializes the interface.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IColumnProvider::GetColumnInfo

Requests information about a column.

```
HRESULT GetColumnInfo(
    DWORD dwIndex,
    SHCOLUMNINFO *psci
);
```

Parameters

dwIndex
[in] Column's zero-based index. It is an arbitrary value that is used to enumerate columns.

psci
[out] Pointer to an **SHCOLUMNINFO** structure to hold the column information.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

This method is called to assign an index to the column and to ask for details on what kind of information the column will contain.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IColumnProvider

IColumnProvider::GetItemData

Requests column data for a specified file.

```
HRESULT GetItemData(
    LPCSHCOLUMNID pscid,
    LPCSHCOLUMNDATA pscd,
    VARIANT *pvarData
);
```

Parameters

pscid

[in] **SHCOLUMNID** structure that identifies the column.

pscd

[in] **SHCOLUMNDATA** structure that specifies the file.

pvarData

[out] Pointer to a VARIANT with the data for the file specified by *pscd* that belongs in the column specified by *pscid*. Set this value if the file is a member of the class supported by the column provider.

Return Values

Returns S_OK if file data is returned, S_FALSE if the file is not supported by the column provider and no data is returned, or an OLE error value otherwise.

Remarks

This method is called to get the data for a file to be displayed in the specified column. It should be thread-safe.

This method is called for every file that Windows Explorer displays, even though many of them will not be supported by a particular column provider. To improve performance, first check the **pwszExt** member of the structure pointed to by *pscd* to see if it has a file name extension that is supported by the column provider. If not, avoid unnecessary processing by immediately returning S_FALSE.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IColumnProvider

IColumnProvider::Initialize

Initializes an **IColumnProvider** interface.

```
HRESULT Initialize(
    LPCSHCOLUMNINIT psci
);
```

Parameters

psci

[in] **SHCOLUMNINIT** structure with initialization information, including the folder whose contents are to be displayed.

Return Value

Returns S_OK if successful, or an OLE error value otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

ICommDlgBrowser

The **ICommDlgBrowser** interface is exposed by the common file dialog boxes to be used when they host a shell browser. If supported, **ICommDlgBrowser** allows a shell view to handle several cases that need to behave differently in a dialog box than in a normal shell view. You obtain an **ICommDlgBrowser** interface pointer by calling **QueryInterface** on the **IShellBrowser** object.

When to Implement

This interface is implemented only by the common file dialog boxes.

When to Use

Use **ICommDlgBrowser** when you need to provide special behavior while hosted inside the common dialog boxes.

ICommDlgBrowser methods	Description
IncludeObject	Allows the common dialog to filter objects that the view displays.
OnDefaultCommand	Called when a user double-clicks in the view or presses the ENTER key.
OnStateChange	Called after a change of state has occurred in a common dialog box.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

ICommDlgBrowser::IncludeObject

Allows the common dialog to filter objects that the view displays.

```
HRESULT IncludeObject(
    IShellView *ppshv,
    LPCITEMIDLIST pidl
);
```

Parameters

ppshv
A pointer to the view's **IShellView** interface.

pidl
Pointer to an item identifier list (PIDL) that is relative to the folder.

Return Values

The browser should return S_OK to include the object in the view, or S_FALSE to hide it.

Remarks

This method is called by the **IEnumIDList** implementation when hosted in the file dialog boxes. The enumerator should call this method to let the common dialog box filter out objects that should not be displayed. Typically, the file dialog boxes will get the display text of the item, and filter by the extension.

Note to Callers Call before returning an object in the shell folder's IDLIST enumerator.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

ICommDlgBrowser

ICommDlgBrowser::OnDefaultCommand

Called when a user double-clicks in the view or presses the ENTER key.

```
HRESULT OnDefaultCommand(
    IShellView *ppshv
);
```

Parameters

ppshv
A pointer to the view's **IShellView** interface.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

Remarks

The browser should return S_OK if it has processed the action or S_FALSE to let the view perform the default action.

Note to Callers This method allows the default command to be handled by the common dialog box instead of the view.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

ICommDlgBrowser

ICommDlgBrowser::OnStateChange

Called after a state, identified by the *uChange* parameter, has changed in the **IShellView** interface.

```
HRESULT OnStateChange(
    IShellView *ppshv,
    ULONG uChange
);
```

Parameters

ppshv
A pointer to the view's **IShellView** interface.

uChange
Change in the selection state. This parameter can be one of the following values:

Value	Description
CDBOSC_KILLFOCUS	The view has lost the focus.
CDBOSC_RENAME	An item has been renamed.
CDBOSC_SELCHANGE	The selection has changed.
CDBOSC_SETFOCUS	The focus has been set to the view.

Remarks

This method is used to let the common file dialog boxes track the state of the view and change its user interface as needed.

Note to Callers When items in the view are selected, or when the view loses the focus, it needs to call this method to notify the common dialog box that either the view state or selection state is changing.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

ICommDlgBrowser

ICommDlgBrowser2

The **ICommDlgBrowser2** interface extends the capabilities of **ICommDlgBrowser**. It is used by the common file dialog boxes when they host a shell browser. A pointer to **ICommDlgBrowser2** can be obtained by calling **QueryInterface** on the **IShellBrowser** object.

When to Implement

This interface is implemented only by common file dialog boxes.

When to Use

Use **ICommDlgBrowser2** when your shell view is hosted inside a common dialog box.

ICommDlgBrowser2 implements all the **ICommDlgBrowser** methods, as well as **IUnknown**. The following methods are specific to **ICommDlgBrowser2**:

Method	Description
GetDefaultMenuText	Called by the shell view to get the default context menu text.
GetViewFlags	Called by the shell view to ask the common dialog box hosting it if all files, including system and hidden files, should be displayed.
Notify	Called by a shell view to notify the common dialog box hosting it that an event has occurred.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

ICommDlgBrowser2::GetDefaultMenuText

Called by the shell view to get the default context menu text.

```
HRESULT GetDefaultMenuText(
    IShellView *pshv,
    WCHAR pszText,
    int cchMax
);
```

Parameters

pshv
Pointer to the **IShellView** interface of the hosted view.

pszText
Buffer that is used by the shell browser to return the default context menu text.

cchMax
Size of the *pszText* buffer. It should be at least the maximum allowable path length (MAX_PATH) in size.

Return Values

Returns S_OK if a new default context menu text was returned in *pshv*. If S_FALSE is returned, use the normal default text. Otherwise, returns a standard COM error value.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

ICommDlgBrowser2

ICommDlgBrowser2::GetViewFlags

Called by the shell view to ask the common dialog box hosting it if all files, including system and hidden files, should be displayed.

```
HRESULT GetViewFlags(
    DWORD *pdwFlags
);
```

Parameters

pdwFlags

Pointer to a flag value that indicates which files should be shown. If it is set to CDB2GVF_SHOWALLFILES, all files, including hidden and system files, should be shown. Otherwise, it is set to NULL.

Return Values

Returns S_OK if successful, or a standard COM error value otherwise.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

ICommDlgBrowser2

ICommDlgBrowser2::Notify

Called by a shell view to notify the common dialog box hosting it that an event has occurred.

```
HRESULT Notify(
    IShellView *pshv,
    DWORD dwNotifyType
);
```

Parameters

pshv

Pointer to the **IShellView** interface of the hosted view.

dwNotifyType

Flag that can take one of two values:

CDB2N_CONTEXTMENU_DONE	Indicates that the context menu is no longer displayed.
CDB2N_CONTEXTMENU_START	Indicates that the context menu is about to be displayed.

Return Values

Returns S_OK if successful, or a standard COM error value otherwise.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

ICommDlgBrowser2

IContextMenu

The **IContextMenu** interface is called by the shell to either create or merge a context menu associated with a shell object.

When to Implement

Implement **IContextMenu** in the following situations:

- Shell extension handlers implement this interface to add items dynamically to a shell object's context menu.
- Namespace extensions implement this interface to specify their object's context menus.

See *Creating Context Menu Handlers* for a detailed discussion of how to implement **IContextMenu**.

When to Use

Applications use **IContextMenu** to get information about the items in an objects context menu, and to invoke the associated commands. To get an object's **IContextMenu** interface, an application must call the object's **IShellFolder::GetUIObjectOf** method.

Methods

GetCommandString Retrieves a command's language-independent name or its Help text. The language-independent name is also referred to as a *verb*.

InvokeCommand Carries out the command associated with a context menu item.

QueryContextMenu Adds commands to a context menu.

Remarks

Shell extension handlers that export this interface must also export **IShellExtInit**. See *Creating Shell Extension Handlers* for details.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IContextMenu::GetCommandString

Retrieves a context menu command's language-independent name or its Help text.

```
HRESULT GetCommandString(
    UINT idCmd,
    UINT uFlags,
    UINT *pwReserved,
    LPSTR pszName,
    UINT cchMax
);
```

Parameters

idCmd
Menu command identifier offset.

uFlags

Flags specifying the information to return. It can have one of the following values:

GCS_HELPTEXTA	Sets *pszName* to an ANSI string containing the help text for the command.
GCS_HELPTEXTW	Sets *pszName* to a Unicode string containing the help text for the command.
GCS_VALIDATEA	Returns S_OK if the menu item exists, S_FALSE otherwise.
GCS_VALIDATEW	Returns S_OK if the menu item exists, S_FALSE otherwise.
GCS_VERBA	Sets *pszName* to an ANSI string containing the language-independent command name for the menu item.
GCS_VERBW	Sets *pszName* to a Unicode string containing the language-independent command name for the menu item.

pwReserved

Reserved. Applications must specify NULL when calling this method, and handlers must ignore this parameter when called.

pszName

Address of the buffer to receive the null-terminated string being retrieved.

cchMax

Size of the buffer to receive the null-terminated string.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

Remarks

The language-independent command name, or *verb* is a name that can be passed to the **IContextMenu::InvokeCommand** method to activate a command by an application. The Help text is a description of the command that Windows Explorer displays in its status bar. It should be reasonably short (under 40 characters).

Even though *pszName* is declared as an LPSTR, you must cast it to LPWSTR and return a Unicode string if *uFlags* is set to either GCS_HELPTEXTW or GCS_VERBW.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IContextMenu

IContextMenu::InvokeCommand

Carries out the command associated with a context menu item.

```
HRESULT InvokeCommand(
    LPCMINVOKECOMMANDINFO pici
);
```

Parameters

pici

Pointer to a **CMINVOKECOMMANDINFO** or **CMINVOKECOMMANDINFOEX** structure containing information about the command. See the Remarks for further details.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

Remarks

The **IContextMenu** interface is exported by several shell extension handlers and namespace extensions to add commands to context menus. When the user selects one of the commands that the handler or namespace extension added to a context menu, the shell will call its **InvokeCommand** method. The command may be specified by its menu identifier offset, defined when **IContextMenu::QueryContextMenu** was called, or by its associated verb. An application can invoke this method directly by obtaining an a pointer to an object's **IContextMenu** interface. It also can invoke it indirectly by calling **ShellExecute** or **ShellExecuteEx**, and specifying a verb that is supported by the namespace extension or handler.

Although the *pici* parameter is declared in shlobj.h as a **CMINVOKECOMMANDINFO** structure, in practice it often points to a **CMINVOKECOMMANDINFOEX** structure. This structure is an extended version of **CMINVOKECOMMANDINFO**, and has several additional members that allow Unicode strings to be passed.

Note to Users You can pass either structure to **IContextMenu::InvokeCommand**. Either will work for ANSI strings, but you must use a **CMINVOKECOMMANDINFOEX** structure for Unicode strings.

Note to Implementers Check the **cbSize** member of *pici* to determine which structure was passed in. If it is a **CMINVOKECOMMANDINFOEX** structure, and the **fMask** member has the CMIC_MASK_UNICODE flag set, you must cast *pici* to **CMINVOKECOMMANDINFOEX** in order to use the Unicode information contained in the last five members of the structure.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IContextMenu

IContextMenu::QueryContextMenu

Adds commands to a context menu.

```
HRESULT QueryContextMenu(
    HMENU hmenu,
    UINT indexMenu,
    UINT idCmdFirst,
    UINT idCmdLast,
    UINT uFlags
);
```

Parameters

hmenu

Handle to the menu. The handler should specify this handle when adding menu items.

indexMenu

Zero-based position at which to insert the first menu item.

idCmdFirst

Minimum value that the handler can specify for a menu item identifier.

idCmdLast

Maximum value that the handler can specify for a menu item identifier.

uFlags

Optional flags specifying how the context menu can be changed. This parameter can be set to any combination of the following values:

CMF_CANRENAME

This flag is set if the calling application supports renaming of items. A context menu extension or drag-and-drop handler should ignore this flag. A namespace extension should add a rename item to the menu if applicable.

CMF_DEFAULTONLY

This flag is set when the user is activating the default action, typically by double-clicking. This flag provides a hint for the context menu extension to add nothing if it does not modify the default item in the menu. A context menu extension or drag-and-drop handler should not add any menu items if this value is specified. A namespace extension should add only the default item (if any).

CMF_EXPLORE
This flag is set when Windows Explorer's tree window is present.

CMF_EXTENDEDVERBS
This flag is set when the calling application wants extended verbs. Normal verbs are displayed when the user right-clicks an object. To display extended verbs, the user must right-click while pressing the SHIFT key.

CMF_INCLUDESTATIC
This flag is set when a static menu is being constructed. Only the browser should use this flag. All other context menu extensions should ignore this flag.

CMF_NODEFAULT
This flag is set if no item in the menu should be the default item. A drag-and-drop handler should ignore this flag. A namespace extension should not set any of the menu items to the default.

CMF_NORMAL
Indicates normal operation. A context menu extension, namespace extension, or drag-and-drop handler can add all menu items.

CMF_NOVERBS
This flag is set for items displayed in the "Send To:" menu. Context menu handlers should ignore this value.

CMF_VERBSONLY
This flag is set if the context menu is for a shortcut object. Context menu handlers should ignore this value.

The remaining bits of the low-order word are reserved by the system. The high-order word may be used for context-specific communications. The CMF_RESERVED value can be used to mask out the low-order word.

Return Values

If successful, returns an HRESULT value that has its severity value set to SEVERITY_SUCCESS and its code value set to the largest command identifier that was assigned, plus one. Otherwise, returns an OLE error code.

Remarks

This method should call either **InsertMenu** or **InsertMenuItem** to insert its menu items into the menu specified by *hMenu*. The *indexMenu* parameter holds the index that should be used for the first menu item. The identifier of each menu item must fall within the range defined by *idCmdFirst* and *idCmdLast*.

A common practice is to set the first command identifier to *idCmdFirst* (an offset of zero), and increment the offset for each additional command by one. This practice ensures that you do not exceed *idCmdLast* and preserves the range of identifiers that are available for use by other handlers. You should store the offsets for future reference because they may be used to identify the command in subsequent calls to **IContextMenu::GetCommandString** and **IContextMenu::InvokeCommand**.

If the shell subsequently calls another context menu handler, it will use the code value of the returned HRESULT to set *idCmdFirst* when it calls that handler's **QueryContextMenu** method.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IContextMenu

IContextMenu2

The **IContextMenu2** interface is used to either create or merge a context menu associated with a certain object when the menu involves owner-drawn menu items.

When to Implement

Implement **IContextMenu2** if your namespace extension or context menu handler needs to process one or more of the following messages:

- **WM_INITMENUPOPUP**
- **WM_DRAWITEM**
- **WM_MEASUREITEM**

These messages are forwarded to **IContextMenu2**—through the **HandleMenuMsg** method—only if a **QueryInterface** call for an **IContextMenu2** interface pointer is successful, indicating that the object supports this interface.

When to Use

Applications normally do not call this interface directly.

Methods

IContextMenu2 is derived from **IContextMenu**. The following method is specific to **IContextMenu2**:

HandleMenuMsg	Handles messages related to drawing owner-drawn menu items.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IContextMenu2::HandleMenuMsg

Allows client objects of the **IContextMenu** interface to handle messages associated with owner-drawn menu items.

```
HRESULT HandleMenuMsg(
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam
);
```

Parameters

uMsg

Message to be processed. If it is WM_INITPOPUP, WM_DRAWITEM, WM_MENUCHAR,or WM_MEASUREITEM, the client object being called may provide owner-drawn menu items.

wParam

Additional message information. The value of this parameter depends on the value of the *uMsg* parameter.

lParam

Additional message information. The value of this parameter depends on the value of the *uMsg* parameter.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

Remarks

HandleMenuMsg is called when a client of **IContextMenu** determines that the **IContextMenu2** interface is supported and receives one of the messages specified in the description of the *uMsg* parameter.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IContextMenu3

The **IContextMenu3** interface is used to create or merge a context menu associated with a certain object when the menu implementation needs to process the **WM_MENUCHAR** message.

When to Implement

Implement **IContextMenu3** if your context menu extension needs to process the WM_MENUCHAR message.

These messages are forwarded to **IContextMenu3**—through the **HandleMenuMsg2** method—only if a **QueryInterface** call for an **IContextMenu3** interface pointer is successful, indicating that the object supports this interface.

When to Use

You do not call this interface directly. **IContextMenu3** is used by the operating system only when it has confirmed that your application is aware of this interface.

IContextMenu3 is derived from **IContextMenu2**. The following method is specific to **IContextMenu3**:

IContextMenu3 method	Description
HandleMenuMsg2	Called when the window that owns the menu receives a **WM_MENUCHAR** message.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).

Windows CE: Unsupported.
Header: Declared in shlobj.h.

IContextMenu3::HandleMenuMsg2

Allows context menu handlers to process the **WM_MENUCHAR** message.

```
HRESULT HandleMenuMsg2(
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam,
    LRESULT *plResult
);
```

Parameters

uMsg
Message to be processed. At the present time, this method is called only for **WM_MENUCHAR**.

wParam
Additional message information. The value of this parameter depends on the value of the *uMsg* parameter.

lParam
Additional message information. The value of this parameter depends on the value of the *uMsg* parameter.

plResult
Address of an LRESULT value that the owner of the menu will return from the message.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

Remarks

HandleMenuMsg2 is called when a client of **IContextMenu** determines that the **IContextMenu3** interface is supported and receives one of the messages specified in the description of the *uMsg* parameter.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).

Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IContextMenu2

ICopyHook

ICopyHook is a COM-based interface used to create a *copy hook handler.* A copy hook handler is a shell extension that determines if a shell folder or printer object can be moved, copied, renamed, or deleted. The shell calls the **ICopyHook::CopyCallback** method prior to performing one of these operations.

The copy hook handler, which is an OLE in-process server (a DLL), does not perform the task itself, but it does approve or disapprove the action. If the shell receives approval from the copy hook handler, it performs the file system operation. Copy hook handlers are not informed about the success of an operation, so they cannot monitor actions taken on folder objects unless **FindFirstChangeNotification** is used.

A folder object can have multiple copy hook handlers. For example, even if the shell already has a copy hook handler registered for a particular folder object, you can still register one of your own. If two or more copy hook handlers are registered for an object, the shell calls each of them before performing one of the specified file-system operations.

The shell initializes **ICopyHook** directly, without using the **IShellExtInit** interface first.

CopyCallback returns an integer value that indicates whether or not the shell should perform the operation. The shell will call each copy hook handler registered for a folder object until all the handlers have been called or until one of them has returned a value other than IDYES. The handler returns IDYES to specify that the operation should be performed, or IDNO or IDCANCEL to specify that the operation should be discontinued.

When to Implement

Implement a copy hook handler when you want to be able to control when, or if, these file system operations are performed on a given object. You might want to use a copy hook handler on shared folders, for example.

When to Use

You do not call this shell extension directly. **ICopyHook::CopyCallback** is called by the shell prior to moving, copying, deleting, or renaming a shell folder object.

Method	Description
CopyCallback	Determines whether a move, copy, delete, or rename operation on a folder object should be allowed or disallowed.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

ICopyHook::CopyCallback

Determines whether the shell will be allowed to move, copy, delete, or rename a folder or printer object.

```
UINT CopyCallback(
    HWND hwnd,
    UINT wFunc,
    UINT wFlags,
    LPCSTR pszSrcFile,
    DWORD dwSrcAttribs,
    LPCSTR pszDestFile,
    DWORD dwDestAttribs
);
```

Parameters

hwnd

Handle to the window that the copy hook handler should use as the parent for any user interface elements the handler may need to display. If FOF_SILENT is specified, the method should ignore this parameter.

wFunc

Operation to perform. This parameter can be one of the values listed under the **wFunc** member of the **SHFILEOPSTRUCT** structure.

wFlags

Flags that control the operation. This parameter can be one or more of the values listed under the **fFlags** member of the **SHFILEOPSTRUCT** structure.

pszSrcFile

Address of a string that contains the name of the source folder.

dwSrcAttribs

Attributes of the source folder. This parameter can be a combination of any of the file attribute flags (FILE_ATTRIBUTE_*) defined in the Windows header files.

pszDestFile

Address of a string that contains the name of the destination folder.

dwDestAttribs

Attributes of the destination folder. This parameter can be a combination of any of the file attribute flags (FILE_ATTRIBUTE_*) defined in the Windows header files.

Return Values

Returns an integer value that indicates whether or not the shell should perform the operation. It can be one of the following:

Value	Description
IDCANCEL	Prevents the current operation and cancels any pending operations.
IDNO	Prevents the operation on this folder but continues with any other operations that have been approved (for example, a batch copy operation).
IDYES	Allows the operation.

Remarks

The shell calls each copy hook handler registered for a folder or printer object until all the handlers have been called, or until one of them returns IDNO or IDCANCEL.

Copy hook handlers for folders are registered under **HKEY_CLASSES_ROOT\Directory\Shellex\CopyHookHandlers\***your_copyhook*\\{*copy hook CLSID value*}. Copy hook handlers for printers are registered under **HKEY_CLASSES_ROOT\Printers\Shellex\CopyHookHandlers\***your_copyhook*\\{*copyhook CLSID value*}.

When this method is called, the shell initializes the **ICopyHook** interface directly without using an **IShellExtInit** interface first.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

SHFileOperation

ICurrentWorkingDirectory

The **ICurrentWorkingDirectory** interface allows a client to retrieve or set an object's current working directory.

When to Implement

Implement this interface if your object allows clients to retrieve or set the current working directory.

When to Use

Use this interface to retrieve or set the working directory of the object that exports it.

Methods

ICurrentWorkingDirectory exposes the following methods in addition to **IUnknown**:

GetDirectory	Retrieves the current working directory.
SetDirectory	Sets the current working directory.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

ICurrentWorkingDirectory::GetDirectory

Retrieves the current working directory.

```
HRESULT GetDirectory(
    LPWSTR pwzPath,
    DWORD cchSize
);
```

Parameters

pwzPath
[out] Address of a buffer. On return, it will hold a NULL-terminated Unicode string with the current working directory's fully qualified path.

cchSize
[in] Size of the buffer in Unicode characters, including the terminating NULL character.

Return Values

Returns S_OK if successful, or an OLE error code otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

ICurrentWorkingDirectory

ICurrentWorkingDirectory::SetDirectory

Sets the current working directory.

```
HRESULT SetDirectory(
    LPCWSTR pwzPath
);
```

Parameters

pwzPath

[in] Address of a NULL-terminated Unicode string with the fully qualified path of the new working directory.

Return Values

Returns S_OK if successful, or an OLE error code otherwise.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

ICurrentWorkingDirectory

IDeskBand

IDeskBand is used to obtain information about a band object. See *Band Objects* for more information about band objects.

When to Implement

Implement **IDeskBand** if you are implementing a band object.

When to Use

You do not call this interface directly. **IDeskBand** is used by the shell or the browser to obtain display information for a band object.

IDeskBand is derived from **IDockingWindow**. The following method is specific to **IDeskBand**:

IDeskBand method	Description
GetBandInfo	Retrieves the information for a band object.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IDeskBand::GetBandInfo

Retrieves the information for a band object.

```
HRESULT GetBandInfo(
    DWORD dwBandID,
    DWORD dwViewMode,
    DESKBANDINFO* pdbi
);
```

Parameters

dwBandID

Identifier of the band. The container assigns this identifier. The band object can keep this value if it is required.

dwViewMode

View mode of the band object. This will be one of the following values:

DBIF_VIEWMODE_NORMAL	The band object is being displayed in a horizontal band.
DBIF_VIEWMODE_VERTICAL	The band object is being displayed in a vertical band.
DBIF_VIEWMODE_FLOATING	The band object is being displayed in a floating band.
DBIF_VIEWMODE_TRANSPARENT	The band object is being displayed in a transparent band.

pdbi

Address of a **DESKBANDINFO** structure that receives the band information for the object. The **dwMask** member of this structure indicates what information is being requested.

Return Value

Returns NOERROR if successful, or an OLE-defined error code otherwise.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IDockingWindow

The **IDockingWindow** interface is implemented by window objects that can be docked within the border space of a Windows Explorer window.

When to Implement

You implement **IDockingWindow** when you want to display a window inside a browser frame. This normally is used for user interface windows, such as toolbars.

When to Use

You normally do not use the **IDockingWindow** interface. The shell browser uses this interface to support docked windows inside the browser frame.

IDockingWindow is derived from **IOleWindow**. The following are the methods specific to **IDockingWindow**:

IDockingWindow method	Description
CloseDW	Notifies the docking window object that it is about to be removed.
ResizeBorderDW	Notifies the docking window object that the frame's border space has changed.
ShowDW	Instructs the docking window object to show or hide itself.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IDockingWindow::CloseDW

Notifies the docking window object that it is about to be removed from the frame. The docking window object should save any persistent information at this time.

```
HRESULT CloseDW(
    DWORD dwReserved
);
```

Parameters

dwReserved
Reserved. This parameter should always be zero.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IDockingWindow, **IDockingWindowFrame**, **IDockingWindowSite**

IDockingWindow::ResizeBorderDW

Notifies the docking window object that the frame's border space has changed. In response to this method, the **IDockingWindow** implementation must call **IDockingWindowSite::SetBorderSpaceDW**, even if no border space is required or a change is not necessary.

```
HRESULT ResizeBorderDW(
    LPCRECT prcBorder,
    IUnknown* punkToolbarSite,
    BOOL fReserved
);
```

Parameters

prcBorder

Address of a **RECT** structure that contains the frame's available border space.

punkToolbarSite

Address of the site's **IUnknown** interface. The docking window object should call the **QueryInterface** method for this interface, requesting IID_IShellToolbarSite, and use that interface to negotiate its border space. It is the docking window object's responsibility to release this interface when it is no longer needed.

fReserved

Reserved for future use. This parameter should always be zero.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

The *prcBorder* parameter will contain the frame's entire available border space. The docking window object should negotiate its border space and then use this information to position itself.

For example, if the docking window object requires 25 pixels at the top of the border space, it should negotiate for this by allocating a **BORDERWIDTHS** structure, setting the **top** member to 25, calling **IDockingWindowSite::RequestBorderSpaceDW**, and then calling **IDockingWindowSite::SetBorderSpaceDW**. The docking window object can then position its window at **prcBorder->left** and **prcBorder->top**. The width of the docking window object's window is determined from **prcBorder->right - prcBorder->left**, and its height is contained in the **top** member of the **BORDERWIDTHS** structure.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IDockingWindow, **IDockingWindowFrame**, **IDockingWindowSite**

IDockingWindow::ShowDW

Instructs the docking window object to show or hide itself.

```
HRESULT ShowDW(
    BOOL bShow
);
```

Parameters

bShow

Boolean value indicating whether the docking window object should show or hide itself. If this parameter is nonzero, the docking window object should show its window. If it is zero, the docking window object should hide its window and return its border space by calling **IDockingWindowSite::SetBorderSpaceDW** with all zeros.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IDockingWindow, **IDockingWindowFrame**, **IDockingWindowSite**

IDockingWindowFrame

The **IDockingWindowFrame** interface is implemented by the browser to support adding **IDockingWindow** objects to a frame.

When to Implement

You normally do not implement the **IDockingWindowFrame** interface. The shell browser implements this interface to support docked windows inside the browser frame.

When to Use

You use **IDockingWindowFrame** to add, find, and remove docking window objects in a browser frame.

IDockingWindowFrame is derived from **IOleWindow**. The following are the methods specific to **IDockingWindowFrame**:

IDockingWindowFrame methods	Description
AddToolbar	Adds an **IDockingWindow** object to a frame.
FindToolbar	Finds an **IDockingWindow** object in a frame.
RemoveToolbar	Removes an **IDockingWindow** object from a frame.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IDockingWindowFrame::AddToolbar

Adds the specified **IDockingWindow** object to the frame.

```
HRESULT AddToolbar(
    IUnknown* punkSrc,
    LPCWSTR pwszItem,
    DWORD dwAddFlags
);
```

Parameters

punkSrc
Address of the **IDockingWindow** object to be added.

pwszItem
Address of a null-terminated UNICODE string. This is an application-defined string used for identifying the docking window object.

dwAddFlags
Flags for the docking window object being added. This can be one or more of the following values:

0	The docking window is a regular, visible docking window.
DWFAF_HIDDEN	The docking window is added but is not shown. To show it at a later time, call its **IDockingWindow::ShowDW** method.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IDockingWindowFrame, **IDockingWindowSite**

IDockingWindowFrame::FindToolbar

Finds the specified **IDockingWindow** object in the toolbar frame and returns an interface pointer to it.

```
HRESULT FindToolbar(
    LPCWSTR pwszItem,
    REFIID riid,
    LPVOID* ppvObj
);
```

Parameters

pwszItem
Address of a null-terminated UNICODE string. This is the same string that was passed to the **AddToolbar** method.

riid
Identifier of the desired COM interface.

ppvObj
Address to receive the interface pointer. If an error occurs, a NULL pointer is placed in this address.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IDockingWindowFrame, **IDockingWindowSite**

IDockingWindowFrame::RemoveToolbar

Removes the specified **IDockingWindow** from the toolbar frame.

```
HRESULT RemoveToolbar(
    IUnknown* punkSrc,
    DWORD dwRemoveFlags
);
```

Parameters

punkSrc

Address of the **IDockingWindow** object to be removed. The **IDockingWindowFrame** implementation will call the **IDockingWindow::CloseDW** and **Release** methods.

dwRemoveFlags

Option flags for removing the docking window object. This parameter can be one or more of the following values:

DWFRF_DELETECONFIGDATA	In addition to deleting the toolbar, any configuration data is removed as well.
DWFRF_NORMAL	The default delete processing is performed.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IDockingWindowSite

IDragSourceHelper

This interface is exposed by the shell to allow an application to specify the image that will be displayed during a shell drag-and-drop operation.

When to Implement

This interface is exposed by the shell's drag-image manager. It is not implemented by applications.

When to Use

Use this interface when you want to specify the image that will be displayed during a shell drag-and-drop operation. The **IDragSourceHelper** and **IDropTargetHelper** interfaces are exposed by the drag-image manager object to allow the **IDropTarget** interface to use custom drag images. To use either of these interfaces, you must create an in-process server drag-image manager object by calling **CoCreateInstance** with a CLSID of CLSID_DragDropHelper. Get interface pointers using standard COM procedures.

The **IDragSourceHelper** interface provides two alternative ways to specify the bitmap to be used as a drag image:

- Controls that have a window can register a **DI_GETDRAGIMAGE** window message for it and initialize the drag-image manager with **IDragSourceHelper::InitializeFromWindow**. When the **DI_GETDRAGIMAGE** message is received, the handler puts the drag image bitmap information in the **SHDRAGIMAGE** structure that is passed as the message's *lParam* value.
- Windowless controls can initialize the drag-image manager with **IDragSourceHelper::InitializeFromBitmap**. This method allows an application simply to specify the bitmap.

Note The drag-and-drop helper object calls **IDataObject::SetData** to load private formats into the data object that are used for cross-process support. It later retrieves these formats by calling **IDataObject::GetData**. To support the drag-drop helper object, the data object's **SetData** and **GetData** implementations must be able to accept and return arbitrary private formats.

For further discussion of shell drag-and-drop operations, see *Transferring Shell Data Using Drag-and-Drop or the Clipboard.*

Methods

IDragSourceHelper exposes the following methods in addition to **IUnknown**:

InitializeFromBitmap	Initializes the drag-image manager for a windowless control.
InitializeFromWindow	Initializes the drag-image manager for a control with a window.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IDragSourceHelper::InitializeFromBitmap

Initializes the drag-image manager for a windowless control.

```
HRESULT InitializeFromBitmap(
    LPSHDRAGIMAGE pshdi,
    IDataObject *pDataObject
);
```

Parameters

pshdi
[in] **SHDRAGIMAGE** structure that contains information about the bitmap.

pDataObject
[in] Pointer to the data object's **IDataObject** interface.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IDragSourceHelper

IDragSourceHelper::InitializeFromWindow

Initializes the drag-image manager for a control with a window.

```
HRESULT InitializeFromWindow(
    HWND hwnd,
    POINT *ppt,
    IDataObject *pDataObject
);
```

Parameters

hwnd

[in] Handle to the window that will receive the **DI_GETDRAGIMAGE** message.

ppt

[in] Pointer to a **POINT** structure that specifies the location of the cursor within the drag image. The structure should contain the offset from the upper-left corner of the drag image to the location of the cursor.

pDataObject

[in] Pointer to the data object's **IDataObject** interface.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

The DI_GETDRAGIMAGE message allows you to source a drag image from a custom control. It is defined in Shlobj.h and must be registered with **RegisterWindowMessage**. When the window specified by *hwnd* receives the DI_GETDRAGIMAGE message, the *lParam* value will hold a pointer to an **SHDRAGIMAGE** structure. The handler should fill the structure with the drag image bitmap information.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IDragSourceHelper

IDropTargetHelper

This interface allows drop targets to display a drag image while it is over the target window.

When to Implement

This interface is exposed by the shell's drag-image manager. It is not implemented by applications.

When to Use

This interface is used by drop targets to have the drag-image manager display the drag image while it is over the target window. The **IDragSourceHelper** and **IDropTargetHelper** interfaces are exposed by the drag-image manager object to allow the **IDropTarget** interface to use custom drag images. To use either of these interfaces, you must create an in-process server drag-image manager object by calling **CoCreateInstance** with a CLSID of CLSID_DragDropHelper. Get interface pointers using standard COM procedures.

Four of the **IDropTargetHelper** methods correspond to the four **IDropTarget** methods. When you implement **IDropTarget**, each of its methods should call the corresponding **IDropTargetHelper** method to pass the information to the drag-image manager. The fifth **IDropTargetHelper** method notifies the drag-image manager to show or hide the drag image. This method is used when dragging over a target window in a low color-depth video mode. It allows the target to hide the drag image while it is painting the window.

Note The drag-and-drop helper object calls **IDataObject::SetData** to load private formats into the data object that are used for cross-process support. It later retrieves these formats by calling **IDataObject::GetData**. To support the drag-and-drop helper object, the data object's **SetData** and **GetData** implementations must be able to accept and return arbitrary private formats.

For further discussion of shell drag-drop operations, see *Transferring Shell Data Using Drag-Drop or the Clipboard.*

Methods

IDropTargetHelper exposes the following methods in addition to **IUnknown**:

DragEnter	Notifies the drag-image manager that the drop target's **IDropTarget::DragEnter** method has been called.
DragLeave	Notifies the drag-image manager that the drop target's **IDropTarget::DragLeave** method has been called.
DragOver	Notifies the drag-image manager that the drop target's **IDropTarget::DragOver** method has been called.
Drop	Notifies the drag-image manager that the drop target's **IDropTarget::Drop** method has been called.
Show	Notifies the drag-image manager to show or hide the drag image.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IDragSourceHelper

IDropTargetHelper::DragEnter

Notifies the drag-image manager that the drop target's **IDropTarget::DragEnter** method has been called.

```
HRESULT DragEnter(
    HWND hwndTarget,
    IDataObject *pDataObject,
    POINT *ppt,
    DWORD dwEffect
);
```

Parameters

hwndTarget
[in] Target's window handle.

pDataObject
[in] Pointer to the data object's **IDataObject** interface.

ppt
[in] **POINT** structure pointer that was received in the **IDropTarget::DragEnter** method's *ppt* parameter.

dwEffect
[in] Value pointed to by the **IDropTarget::DragEnter** method's *pdwEffect* parameter.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

This method is called by a drop target when its **IDropTarget::DragEnter** method is called. It notifies the drag-image manager that the drop target has been entered, and provides it with the information needed to display the drag image.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IDropTargetHelper

IDropTargetHelper::DragLeave

Notifies the drag-image manager that the drop target's **IDropTarget::DragLeave** method has been called.

```
HRESULT DragLeave( VOID );
```

Parameters

None.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

This method is called by a drop target when its **IDropTarget::DragLeave** method is called. It notifies the drag-image manager that the cursor has left the drop target.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IDropTargetHelper

IDropTargetHelper::DragOver

Notifies the drag-image manager that the drop target's **IDropTarget::DragOver** method has been called.

```
HRESULT DragOver(
    POINT *ppt,
    DWORD dwEffect
);
```

Parameters

ppt

[in] **POINT** structure pointer that was received in the **IDropTarget::DragOver** method's *pt* parameter.

dwEffect

[in] Value pointed to by the **IDropTarget::DragOver** method's *pdwEffect* parameter.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

This method is called by a drop target when its **IDropTarget::DragOver** method is called. It notifies the drag-image manager that the cursor position has changed, and provides it with the information needed to display the drag image.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IDropTargetHelper

IDropTargetHelper::Drop

Notifies the drag-image manager that the drop target's **IDropTarget::Drop** method has been called.

```
HRESULT Drop(
    IDataObject *pDataObject,
    POINT *ppt,
    DWORD dwEffect
);
```

Parameters

pDataObject

[in] Pointer to the data object's **IDataObject** interface.

ppt

[in] **POINT** structure pointer that was received in the **IDropTarget::Drop** method's *pt* parameter.

dwEffect

[in] Value pointed to by the **IDropTarget::Drop** method's *pdwEffect* parameter.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

This method is called by a drop target when its **IDropTarget::Drop** method is called. It notifies the drag-image manager that the object has been dropped, and provides it with the information needed to display the drag image.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IDropTargetHelper

IDropTargetHelper::Show

Notifies the drag-image manager to show or hide the drag image.

```
HRESULT Show(
    BOOL fShow
);
```

Parameters

fShow

[in] Boolean value that is set to TRUE to show the drag image, and FALSE to hide it.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

This method is used when dragging over a target window in a low color-depth video mode. It allows the target to have the drag-image manager hide the drag image while it is painting the window. While you are painting a window that is currently being dragged over, hide the drag image by calling **Show** with *fShow* set to FALSE. Once the window has been painted, display the drag image again by calling **Show** with *fShow* set to TRUE.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IDropTargetHelper

IDockingWindowSite

The **IDockingWindowSite** interface is implemented by the browser to manage the border space for one or more **IDockingWindow** objects. This interface is similar to the **IOleInPlaceUIWindow** interface.

When to Implement

You normally do not implement the **IDockingWindowSite** interface. The shell browser implements this interface to support docked windows inside the browser frame.

When to Use

You use **IDockingWindowSite** to negotiate the space for a docking window object in a browser frame.

IDockingWindowSite is derived from **IOleWindow**. The following are the methods specific to **IDockingWindowSite**:

IDockingWindowSite methods	Description
GetBorderDW	Retrieves the allocated border space for a particular **IDockingWindow** object.
RequestBorderSpaceDW	Processes border space requests for an **IDockingWindow** object.
SetBorderSpaceDW	Allocates border space for an **IDockingWindow** object.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with
Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0
or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IDockingWindowSite::GetBorderDW

Retrieves the border space allocated for the specified **IDockingWindow** object.

```
HRESULT GetBorderDW(
    IUnknown* punkSrc,
    LPRECT prcBorder
);
```

Parameters

punkSrc

Address of the **IDockingWindow** object for which the border space is being requested.

prcBorder

Address of a **RECT** structure to receive the entire available border space for the browser. The docking window object should use this information to determine where to place itself. See the **IDockingWindow::ResizeBorderDW** method for more information.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with
Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0
or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IDockingWindowFrame, **IDockingWindowSite**

IDockingWindowSite::RequestBorderSpaceDW

Approves, modifies, or refuses a request for an **IDockingWindow** object's border space. The border space is not allocated until the **SetBorderSpaceDW** method is called.

```
HRESULT RequestBorderSpaceDW(
    IUnknown* punkSrc,
    LPCBORDERWIDTHS pbw
);
```

Parameters

punkSrc

Address of the **IDockingWindow** object for which the border space is being requested.

pbw

Address of a **BORDERWIDTHS** structure. Before calling this method, the structure must be filled with the desired border space. After the method's successful completion, the structure will contain the approved border space. The **IDockingWindowSite** object may change these values. If border space is critical, it is the caller's responsibility to determine if the returned border space is sufficient.

Return Values

Returns NOERROR if the border space request is approved or modified, or an OLE-defined error code otherwise.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IDockingWindowFrame

IDockingWindowSite::SetBorderSpaceDW

Allocates and reserves border space for an **IDockingWindow** object.

```
HRESULT SetBorderSpaceDW(
    IUnknown* punkSrc,
    LPCBORDERWIDTHS pbw
);
```

Parameters

punkSrc

Address of the **IDockingWindow** object for which the border space is being set.

pbw

Address of a **BORDERWIDTHS** structure that contains the **IDockingWindow** object's border space. The border space should have been approved by **IDockingWindowSite** through a successful call to the **RequestBorderSpaceDW** method.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IDockingWindowFrame

IEmptyVolumeCache

This interface is used by the disk-cleanup manager to communicate with a disk-cleanup handler. Its methods allow the manager to request information from a handler, and notify it of events such as the start of a scan or purge.

When to Implement

This interface must be implemented by disk-cleanup handlers running on Windows 98. Handlers running on Windows 2000 also should expose **IEmptyVolumeCache2**.

Methods

Method	Description
Deactivate	Notifies the handler that the disk-cleanup manager is shutting down.
GetSpaceUsed	Asks for the amount of disk space that the handler can free.
Initialize	Initializes the disk cleaner handler.
Purge	Notifies the handler to clean up its unessential files.
ShowProperties	Asks the handler to display its user interface (UI).

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in emptyvc.h.

+ See Also

IEmptyVolumeCacheCallBack

IEmptyVolumeCache::Deactivate

This method is used to notify the handler that the disk-cleanup manager is shutting down.

```
HRESULT Deactivate(
  DWORD dwFlags
);
```

Parameters

dwFlags

[out] Flag that can be set to return information to the disk-cleanup manager. It can have the following value:

EVCF_REMOVEFROMLIST

If this flag is set, the disk-cleanup manager will delete the handler's registry subkey.

Return Values

S_OK	This value should always be returned.

Remarks

If the EVCF_REMOVEFROMLIST flag is set, the handler will not be run again unless the registry entries are reestablished. This flag typically is used for a handler that will run only once.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in emptyvc.h.

IEmptyVolumeCache::GetSpaceUsed

This method requests the amount of disk space that the disk-cleanup handler can free.

```
HRESULT GetSpaceUsed(
  DWORDLONG *pdwSpaceUsed,
  IEmptyVolumeCacheCallback *picb
);
```

Parameters

pdwSpaceUsed

[out] Amount of disk space, in bytes, that the handler can free. This value will be displayed in the disk-cleanup manager's list, to the right of the handler's check box. To indicate that you do not know how much disk space can be freed, set this parameter to –1, and "???MB" will be displayed. If you set the EVCF_DONTSHOWIFZERO flag when **Initialize** was called, setting *pdwSpaceUsed* to zero will notify the disk-cleanup manager to omit the handler from its list.

picb

[in] Pointer to the disk-cleanup manager's **IEmptyVolumeCacheCallback** interface. This pointer can be used to call that interface's **ScanProgress** method to report on the progress of the operation.

Return Values

E_ABORT	The scan operation was ended prematurely. This value usually is returned when a call to **IEmptyVolumeCache::ScanProgress** returns E_ABORT. This return value indicates that the user cancelled the operation by clicking the disk-cleanup manager's **Cancel** button.
S_FALSE	An error occurred when the handler tried to calculate the amount of disk space that could be freed.
S_OK	Success.

Remarks

When this method is called by the disk-cleanup manager, the handler should start scanning its files to determine which of them can be deleted, and how much disk space will be freed. Handlers should call **IEmptyVolumeCache::ScanProgress** periodically to keep the user informed of the progress of the scan, especially if it will take a long time. Calling this method frequently also allows the handler to determine whether the user has cancelled the operation. If **ScanProgress** returns E_ABORT, the user has cancelled the scan. The handler should stop scanning immediately and return E_ABORT.

You should set the *pdwSpaceUsed* parameter to –1 only as a last resort. The handler is of limited value to users if they do not know how much space will be freed.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in emptyvc.h.

IEmptyVolumeCache::Initialize

This method is used to initialize the disk-cleanup handler, based on the information stored under the specified registry key.

```
HRESULT Initialize(
  HKEY hkRegKey,
  LPCWSTR pcwszVolume,
  LPWSTR *ppwszDisplayName,
  LPWSTR *ppwszDescription,
  DWORD *pdwFlags
);
```

Parameters

hkRegKey

[in] Handle to the registry key that holds the information about the handler object.

pcwszVolume

[in] Pointer to a null-terminated Unicode string with the volume root—for example, "C:\".

ppwszDisplayName

[out] Pointer to a null-terminated Unicode string with the name that will be displayed in the disk-cleanup manager's list of handlers. If no value is assigned, the registry value will be used.

ppwszDescription

[out] Pointer to a null-terminated Unicode string that will be displayed when this object is selected from the disk-cleanup manager's list of available disk-cleanup handlers. If no value is assigned, the registry value will be used.

pdwFlags

[in/out] Flags that are used to pass information to the handler, and back to the disk-cleanup manager.

These flags can be passed in to the object:

EVCF_OUTOFDISKSPACE

If this flag is set, the user is out of disk space on the drive. When this flag is received, the handler should be aggressive about freeing disk space, even if it results in a performance loss. The handler, however, should not delete files that would cause an application to fail, or the user to lose data.

EVCF_SETTINGSMODE

If the disk-cleanup manager is being run on a schedule, it will set this flag. You must assign values to the *ppwszDisplayName* and *ppwszDescription* parameters. If this flag is set, the disk-cleanup manager will not call **GetSpaceUsed**, **Purge**, or **ShowProperties**. Because **Purge** will not be called, cleanup must be handled by **Initialize**. The handler should ignore the *pcwszVolume* parameter and clean up any unneeded files regardless of what drive they are on. Because there is no opportunity for user feedback, only those files that are extremely safe to clean up should be touched.

These flags can be passed by the handler back to the disk-cleanup manager:

EVCF_DONTSHOWIFZERO

Set this flag when there are no files to delete. When **IEmptyVolumeCache::GetSpaceUsed** is called, set the *pdwSpaceUsed* parameter to zero, and the disk-cleanup manager will omit the handler from its list.

EVCF_ENABLEBYDEFAULT

Set this flag to have the handler checked by default in the cleanup manager's list. It will run every time the Disk Cleanup utility runs, unless the user clears the handler's check box. Once the check box has been cleared, the handler will not be run until the user selects it again.

EVCF_ENABLEBYDEFAULT_AUTO

Set this flag to have the handler run automatically during scheduled cleanup. This flag should be set only when deletion of the files is low-risk. As with EVCF_ENABLEBYDEFAULT, the user can choose not to run the handler by clearing its check box in the disk-cleanup manager's list.

EVCF_HASSETTINGS

Set this flag to indicate that the handler can display a user interface (UI). An example of a simple UI is a list box that displays the deletable files and allows the user to select which ones to delete. The disk-cleanup manager will then display a button below the cleanup handler's description. The user clicks this button to request the UI. The default button text is "Settings", but the handler can specify a different text by setting the AdvancedButtonText value in its registry key.

EVCF_REMOVEFROMLIST
Set this flag to remove the handler from the disk-cleanup manager's list. All registry information will be deleted, and the handler cannot be run again until the key and its values are restored. This flag is used primarily for one-time cleanup operations.

Return Values

E_ABORT	The cleanup operation was ended prematurely.
E_FAIL	The cleanup operation failed.
S_FALSE	There are no files to delete.
S_OK	Success.

Remarks

This method is used by the Windows 98 disk-cleanup manager. Windows 2000 uses the **InitializeEx** method exported by **IEmptyVolumeCache2**.

Use **CoTaskMemAlloc** to allocate memory for the strings returned through *ppwszDisplayName* and *ppwszDescription*. The disk-cleanup manager will free the memory when it is no longer needed.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in emptyvc.h.

IEmptyVolumeCache::Purge

This method notifies the handler to start deleting its unneeded files.

```
HRESULT Purge(
  DWORDLONG dwSpaceToFree,
  IEmptyVolumeCacheCallback *picb
);
```

Parameters

dwSpaceToFree
[in] Amount of disk space that the handler should free. If this parameter is set to –1, the handler should delete all its files.

picb

[in] Pointer to the disk-cleanup manager's **IEmptyVolumeCacheCallBack** interface. This pointer can be used to call the interface's **PurgeProgress** method to report on the progress of the operation.

Return Values

E_ABORT	The operation was ended prematurely. This value usually is returned when **IEmptyVolumeCache::PurgeProgress** returns E_ABORT. This typically happens when the user cancels the operation by clicking the disk-cleanup manager's **Cancel** button.
S_OK	Success.

Remarks

For Windows 98, the *dwSpaceToFree* parameter is always set to the value specified by the handler when **IEmptyVolumeCache::GetSpaceUsed** was called.

In general, handlers should be kept simple and delete all their files when this function is called. If there are significant performance advantages to deleting only a portion of the files, the handler should implement the **ShowProperties** method. When called, this method displays a UI that allows the user to select the files to be deleted.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in emptyvc.h.

IEmptyVolumeCache::ShowProperties

This method is used to notify the handler to display its user interface (UI).

```
HRESULT ShowProperties(
  HWND hwnd
);
```

Parameters

hwnd

[in] Parent window to be used when displaying the UI.

Return Values

S_OK	The user changed one or more settings.
S_FALSE	No settings were changed.

Remarks

A handler can display a UI, which typically is used to allow the user to select which files are to be cleaned up and how. To do so, the handler sets the EVCF_HASSETTINGS flag in the *pdwFlags* parameter when **Initialize** is called. The disk-cleanup manager then will display a **Settings** button. When that button is clicked, the disk-cleanup manager calls **ShowProperties** to notify the handler to display its UI.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in emptyvc.h.

IEmptyVolumeCache2

This interface extends **IEmptyVolumeCache**. It defines one additional method, **InitializeEx**, that provides better localization support than **IEmptyVolumeCache::Initialize**

When to Implement

This interface should be exported by disk-cleanup handlers running on Windows NT 5.0. Handlers running on Windows 98 must export **IEmptyVolumeCache**.

Methods

In addition to the methods exported by **IEmptyVolumeCache**, **IEmptyVolumeCache2** exports:

Method	Description
InitializeEx	Initializes the disk-cleanup handler.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in emptyvc.h.

+ See Also

IEmptyVolumeCacheCallBack

IEmptyVolumeCache2::InitializeEx

This method is used to initialize the disk-cleanup handler. It provides better support for localization than **IEmptyVolumeCache::Initialize**.

```
HRESULT InitializeEx(
  HKEY hkRegKey,
  LPCWSTR pcwszVolume,
  LPCWSTR pcwszKeyName,
  LPWSTR *ppwszDisplayName,
  LPWSTR *ppwszDescription,
  LPWSTR *ppwszBtnText,
  DWORD *pdwFlags
);
```

Parameters

hkRegKey
[in] Handle to the registry key that holds the information about the handler object.

pcwszVolume
[in] Pointer to a null-terminated Unicode string with the volume root—for example, "C:\".

pcwszKeyName
[in] Pointer to a null-terminated Unicode string with the name of the handler's registry key.

ppwszDisplayName
[out] Pointer to a null-terminated Unicode string with the name that will be displayed in the disk-cleanup manager's list of handlers. You must assign a value to this parameter.

ppwszDescription
[out] Pointer to a null-terminated Unicode string that will be displayed when this object is selected from the disk-cleanup manager's list of available disk cleaners. You must assign a value to this parameter.

ppwszBtnText
[out] Pointer to a null-terminated Unicode string with the text that will be displayed on the disk-cleanup manager's **Settings** button. If the EVCF_HASSETTINGS flag is set, you must assign a value to *ppwszBtnText*. Otherwise, you can set it to NULL.

pdwFlags
[in/out] Flags that are used to pass information to the handler, and back to the disk-cleanup manager.

These flags can be passed into the object:

EVCF_OUTOFDISKSPACE
If this flag is set, the user is out of disk space on the drive. When this flag is received, the handler should be aggressive about freeing disk space, even if it

results in a performance loss. The handler, however, should not delete files that would cause an application to fail or the user to lose data.

EVCF_SETTINGSMODE

If the disk-cleanup manager is being run on a schedule, it will set the EVCF_SETTINGSMODE flag. You must assign values to the *ppwszDisplayName* and *ppwszDescription* parameters. If this flag is set, the disk-cleanup manager will not call **GetSpaceUsed**, **Purge**, or **ShowProperties**. Because **Purge** will not be called, cleanup must be handled by **InitializeEx**. The handler should ignore the *pcwszVolume* parameter and clean up any unneeded files regardless of what drive they are on. Because there is no opportunity for user feedback, only those files that are extremely safe to clean up should be touched.

These flags can be passed by the handler back to the disk-cleanup manager:

EVCF_DONTSHOWIFZERO

Set this flag when there are no files to delete. When **IEmptyVolumeCache2::GetSpaceUsed** is called, set the *pdwSpaceUsed* parameter to zero, and the disk cleanup

EVCF_ENABLEBYDEFAULT

Set this flag to have the handler checked by default in the disk-cleanup manager's list. The handler will be run every time the disk cleanup utility runs, unless the user clears the handler's check box. Once the check box has been cleared, the handler will not be run until the user selects it again.

EVCF_ENABLEBYDEFAULT_AUTO

Set this flag to have the handler run automatically during scheduled cleanup. This flag should be set only when deletion of the files is low-risk. As with EVCF_ENABLEBYDEFAULT, the user can choose not to run the handler by clearing its check box in the disk-cleanup manager's list.

EVCF_HASSETTINGS

Set this flag to indicate that the handler can display a user interface (UI). An example of a simple UI is a list box that displays the deletable files and allows the user to select which ones to delete. The disk-cleanup manager then will display a button below the cleanup handler's description. The user clicks this button to request the UI. Use the *ppwszBtnText* parameter to specify the button's text.

EVCF_REMOVEFROMLIST

Set this flag to remove the handler from the disk-cleanup manager's list. All registry information will be deleted, and the handler cannot be run again until the key and its values are restored. This flag is used primarily for one-time cleanup operations.

Return Values

E_ABORT	The cleanup operation was ended prematurely.
E_FAIL	The cleanup operation failed.
S_FALSE	There are no files to delete.
S_OK	Success.

Remarks

The Windows NT 5.0 disk-cleanup manager first will call **IEmptyVolumeCache2::InitializeEx** to initialize a disk-cleanup handler. It will call **IEmptyVolumeCache::Initialize** only if the **IEmptyVolumeCache2** interface is not implemented. The Windows 98 disk-cleanup manager only supports **IEmptyVolumeCache::Initialize**.

InitializeEx is intended to provide better localization support than **Initialize**. When **InitializeEx** is called, the handler application must assign appropriately localized values to the *ppwszDisplayName* and *ppwszDescription* parameters. If the Settings button is enabled, you must also assign a value to the *ppwszBtnText* parameter. Unlike **Initialize**, if you set these strings to NULL to notify the disk-cleanup manager to get the default values from the registry, **InitializeEx** will fail.

Use **CoTaskMemAlloc** to allocate memory for the strings returned through *ppwszDisplayName*, *ppwszDescription*, and *ppwszBtnText*. The disk-cleanup manager will free the memory when it is no longer needed.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in emptyvc.h.

IEmptyVolumeCacheCallBack

This interface is used by a disk-cleanup handler to communicate with the disk-cleanup manager.

When to Use

A disk-cleanup handler uses this interface to report to the disk-cleanup manager on the progress either of deleting files or of scanning for deletable files. It also provides a way to query the manager, to find out if the user has cancelled the operation. The handler receives a pointer to this interface when the manager calls the **IEmptyVolumeCache::GetSpaceUsed** or **IEmptyVolumeCache::Purge** methods.

Methods

Method	Description
PurgeProgress	Reports the progress of a purge of deletable files.
ScanProgress	Reports the progress of a scan of the file system for deletable files.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in emptyvc.h.

IEmptyVolumeCacheCallback::PurgeProgress

This method is called periodically by a disk-cleanup handler to update the disk-cleanup manager on the progress of a purge of deletable files.

```
HRESULT PurgeProgress(
  DWORDLONG dwSpaceFreed,
  DWORDLONG dwSpaceToFree,
  DWORD dwFlags,
  LPCWSTR pwszReserved
);
```

Parameters

dwSpaceFreed
[in] Amount of disk space, in bytes, that has been freed at this point in the purge. The disk-cleanup manager uses this value to update its progress bar.

dwSpaceToFree
[in] Amount of disk space, in bytes, that remains to be freed at this point in the purge.

dwFlags
[in] Flag that can be sent to the disk-cleanup manager. It can have the following value:

EVCCBF_LASTNOTIFICATION
This flag should be set if the handler will not call this method again. Typically, it is set when the purge is near completion.

pwszReserved
[in] Reserved.

Return Values

E_ABORT	This value is returned when the user clicks the **Cancel** button on the disk-cleanup manager's dialog box while a scan is in progress. The handler should stop purging files and shut down.
S_OK	The handler should continue purging deletable files.

Remarks

This method is called typically by the handler's **IEmptyVolumeCache::Purge** method while the handler is purging deletable files. Handlers should call **PurgeProgress** periodically to keep the user informed of progress, especially if the purge will take a long time. Calling this method frequently also allows the handler to shut down promptly if a user cancels a purge.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in emptyvc.h.

IEmptyVolumeCacheCallback::ScanProgress

This method is called by a disk-cleanup handler to update the disk-cleanup manager on the progress of a scan for deletable files.

```
HRESULT ScanProgress(
  DWORDLONG dwSpaceUsed,
  DWORD dwFlags,
  LPCWSTR pwszReserved
);
```

Parameters

dwSpaceUsed
[in] Amount of disk space that, at this point in the scan, the handler can free.

dwFlags
[in] Flag that can be sent to the disk-cleanup manager. This flag can have the following value:

EVCCBF_LASTNOTIFICATION
This flag should be set if the handler will not call this method again. It is set typically when the scan is near completion.

pwszReserved
[in] Reserved.

Return Values

E_ABORT	This value is returned when the user clicks the **Cancel** button on the disk-cleanup manager's dialog box while a scan is in progress. The handler should stop scanning and shut down.
S_OK	The handler should continue scanning for deletable files.

Remarks

This method is called typically by the handler's **IEmptyVolumeCache::GetSpaceUsed** method while the handler is scanning for deletable files.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in emptyvc.h.

IEnumExtraSearch

The **IEnumExtraSearch** interface is a standard OLE enumerator that is used by a client to determine the available search objects for a folder.

When to Implement

Implement **IEnumExtraSearch** if your namespace extension supports one or more search objects.

When to Use

You do not call this interface directly. An **IEnumExtraSearch** interface is requested by a client only after it has determined that the **IShellFolder2** interface is exposed. Clients get a pointer to this interface by calling **IShellFolder2::EnumSearches**.

IEnumExtraSearch implements **IUnknown** and the standard OLE enumeration methods.

IEnumExtraSearch methods	Description
Clone	Used to save the enumeration state by creating a duplicate of the current enumerator.
Next	Used to request one or more items.
Reset	Resets the enumerator to the first item.
Skip	Skips one or more items.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IEnumExtraSearch::Clone

Used to request a duplicate of the enumerator object to preserve its current state.

```
HRESULT Clone(
    IEnumExtraSearch **ppEnum,
);
```

Parameters

ppEnum

Pointer to the **IEnumExtraSearch** interface of a new enumerator object.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

Remarks

The new enumerator should be created with the same state as the current one. Use the **Skip** method to advance the enumeration index to the appropriate value before returning.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IEnumExtraSearch::Next

Used to request information on one or more search objects.

```
HRESULT Next(
    ULONG celt,
    LPEXTRASEARCH rgelt,
    ULONG *pceltFetched
);
```

Parameters

celt

[in] Number of search objects to be enumerated, starting from the current object. If *celt* is too large, the method should stop and return the actual number of search objects in *pceltFetched*.

rgelt

[out] Pointer to an array of *pceltFetched* **EXTRASEARCH** structures containing information on the enumerated objects.

pceltFetched

[out] Number of objects actually enumerated. This may be less than *celt*.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IEnumExtraSearch::Reset

Used to reset the enumeration index to zero.

```
HRESULT Reset(void);
```

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IEnumExtraSearch::Skip

Skip past a specified number of objects.

```
HRESULT Clone(
    ULONG celt,
);
```

Parameters

celt
Number of objects to skip.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IEnumIDList

The **IEnumIDList** interface provides a standard set of methods that can be used to enumerate the item identifier lists (PIDLs) of the items in a shell folder. When a folder's **IShellFolder::EnumObjects** method is called, it creates an enumeration object and passes a pointer to the object's **IEnumIDList** interface back to the caller.

When to Implement

All shell folder objects must be able to respond to a call to their **IShellFolder::EnumObjects** method by creating an enumeration object that exports **IEnumIDList**. The shell, in particular, uses these objects to enumerate the items in a folder.

When to Use

Use this interface to enumerate the contents of a shell folder object. Call the folder's **IShellFolder::EnumObjects** method and use the returned **IEnumIDList** pointer to enumerate the PIDLs of the items in the folder.

IEnumIDList methods	Description
Clone	Creates a new item enumeration object identical to the current one.
Next	Retrieves the specified number of item identifiers.
Reset	Returns to the beginning of the enumeration.
Skip	Skips over the specified number of items.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IEnumIDList::Clone

Creates a new item enumeration object with the same contents and state as the current one.

```
HRESULT Clone(
    IEnumIDList **ppenum
);
```

Parameters

ppenum

Address of a pointer to the new enumeration object. The calling application eventually must free the new object by calling its **Release** member function.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

This method makes it possible to record a particular point in the enumeration sequence and then return to that point at a later time.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.

Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IEnumIDList

IEnumIDList::Next

Retrieves the specified number of item identifiers in the enumeration sequence and advances the current position by the number of items retrieved.

```
HRESULT Next(
    ULONG celt,
    LPITEMIDLIST *rgelt,
    ULONG *pceltFetched
);
```

Parameters

celt

Number of elements in the array pointed to by the *rgelt* parameter.

rgelt

Address of an array of **ITEMIDLIST** pointers that receives the item identifiers. The implementation must allocate these item identifiers using the shell's allocator (retrieved by the **SHGetMalloc** function). The calling application is responsible for freeing the item identifiers using the shell's allocator.

Note The **ITEMIDLIST** returned in the *rgelt* array are relative to the **IShellFolder** being enumerated.

pceltFetched

Address of a value that receives a count of the item identifiers actually returned in *rgelt*. The count can be smaller than the value specified in the *celt* parameter. This parameter can be NULL only if *celt* is one.

Return Values

Returns NOERROR if successful, S_FALSE if there are no more items in the enumeration sequence, or an OLE-defined error value otherwise.

Remarks

If this method returns a COM error code (as determined by the FAILED macro), then no entries in the *rgelt* array are valid on exit. If this method returns a success code (such as NOERROR or S_FALSE), then the ULONG pointed to by the *pceltFetched* parameter determines how many entries in the *rgelt* array are valid on exit.

The distinction is important in the case where *celt* > 1. For example, if you pass celt=10 and there are only 3 elements left, *pceltFetched will be 3 and the method will return S_FALSE meaning that you reached the end of the file. The three fetched elements will indeed be stored into *rgelt* and are valid.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IEnumIDList

IEnumIDList::Reset

Returns to the beginning of the enumeration sequence.

```
HRESULT Reset(void);
```

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IEnumIDList

IEnumIDList::Skip

Skips over the specified number of elements in the enumeration sequence.

```
HRESULT Skip(
    ULONG celt
);
```

Parameters

celt
Number of item identifiers to skip.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IEnumIDList

IExtractIcon

This interface allows a client to retrieve the icon that is associated with one of the objects in a folder.

There are two ways to get an object's icon. The simplest way is to call **SHGetFileInfo**. However, this approach is inflexible and can be rather slow. A more flexible and efficient way to get an item's icon is to use **IExtractIcon**. The shell uses **IExtractIcon** to retrieve icons when it displays the contents of a folder. To use **IExtractIcon** to get an object's icon:

1. Get a pointer to the **IShellFolder** interface of the folder that contains the object.
2. Call **IShellFolder::GetUIObjectOf** with the PIDL of the object and the interface ID of **IExtractIcon** (IID_IExtractIcon). The folder creates an object to handle the icon extraction, and returns the object's **IExtractIcon** interface pointer.
3. Call **IExtractIcon::GetIconLocation** to get the icon's location.
4. Call **IExtractIcon::Extract** to get the icon's handle.

It can be possible also to extract icons asynchronously on a background thread. This approach is useful when extraction is time-consuming operation. See **IExtractIcon::GetIconLocation** for details.

When to Implement

Namespace extensions implement **IExtractIcon** to provide icons for their objects. A client obtains an **IExtractIcon** interface pointer for an object in a folder by calling the folder's **IShellFolder::GetUIObjectOf** method. The **GetUIObjectOf** implementation must create an object to handle the icon extraction, and return a pointer to the object's **IExtractIcon** interface.

Icon handlers also implement **IExtractIcon**. An icon handler is a type of shell extension handler that allows you to assign icons dynamically to the members of a **file class**.

When to Use

Call this interface if your application needs a more flexible way to retrieve an object's icon than **SHGetFileInfo**.

Methods

Extract	Extracts an icon from the specified location.
GetIconLocation	Retrieves the icon location for an object.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IExtractIcon::Extract

Extracts an icon image from the specified location.

```
HRESULT Extract(
    LPCSTR pszFile,
    UINT nIconIndex,
    HICON *phiconLarge,
    HICON *phiconSmall,
    UINT nIconSize
);
```

Parameters

pszFile
[in] Pointer to a null-terminated string specifying the icon location.

nIconIndex
[in] The index of the icon in the file pointed to by *pszFile*.

phiconLarge
[out] Pointer to an HICON value that receives the handle to the large icon.

phiconSmall
[out] Pointer to an HICON value that receives the handle to the small icon.

nIconSize
[in] Desired size of the icon, in pixels. The low word contains the size of the large icon, and the high word contains the size of the small icon. The size specified can be the width or height. The width of an icon always equals its height.

Return Values

Returns NOERROR if the function extracted the icon, or S_FALSE if the calling application should extract the icon.

Remarks

The icon location and index are the same values returned by the **GetIconLocation** method. If this function returns S_FALSE, these values must specify an icon file name and index that form legal parameters for a call to **Extract**. If **Extract** does not return

S_FALSE, no assumptions should be made about the meanings of the *pszFile* and *nIconIndex* parameters.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IExtractIcon

IExtractIcon::GetIconLocation

Retrieves the location and index of an icon.

```
HRESULT GetIconLocation(
    UINT uFlags,
    LPSTR szIconFile,
    INT cchMax,
    LPINT piIndex,
    UINT *pwFlags
);
```

Parameters

uFlags

[in] Flags. This parameter can be zero or one or more of the following values:

GIL_ASYNC
A client sets this flag to discover whether or not the icon should be extracted asynchronously. If the icon can be extracted rapidly, this flag normally is ignored. If extraction will be time-consuming, **GetIconLocation** should return E_PENDING. See the Remarks for further discussion.

GIL_FORSHELL
The icon is to be displayed in a shell folder.

GIL_OPENICON
The icon should be in the open state if both open- and closed-state images are available. If this flag is not specified, the icon should be in the normal or closed state. This flag is used typically for folder objects.

szIconFile

[out] Pointer to a buffer that receives the icon location. The icon location is a null-terminated string that identifies the file that contains the icon.

cchMax

[in] Size of the buffer pointed to by *szIconFile*.

piIndex

[out] Pointer to an INT that receives the index of the icon in the file pointed to by *szIconFile*.

pwFlags

[out] Pointer to a UINT value that receives zero or a combination of the following values:

GIL_DONTCACHE

The physical image bits for this icon should not be cached by the caller. This distinction is important to consider because a GIL_DONTCACHELOCATION flag may be introduced in future versions of the shell.

GIL_NOTFILENAME

The location is not a file name/index pair. Callers that decide to extract the icon from the location must call this object's **IExtractIcon::Extract** method to obtain the desired icon images.

GIL_PERCLASS

All objects of this class have the same icon. This flag is used internally by the shell. Typical implementations of **IExtractIcon** do not require this flag because the flag implies that an icon handler is not required to resolve the icon on a per-object basis. The recommended method for implementing per-class icons is to register a DefaultIcon for the class.

GIL_PERINSTANCE

Each object of this class has its own icon. This flag is used internally by the shell to handle cases like Setup.exe, where objects with identical names can have different icons. Typical implementations of **IExtractIcon** do not require this flag.

GIL_SIMULATEDOC

The caller should create a document icon using the specified icon.

Return Values

Returns S_OK if the function returned a valid location, or S_FALSE if the shell should use a default icon. If the GIL_ASYNC flag is set in *uFlags*, the method can return E_PENDING to indicate that icon extraction will be time consuming.

Remarks

When a client sets the GIL_ASYNC flag in *uFlags* and receives E_PENDING as a return value, it typically creates a background thread to extract the icon. It calls **GetIconLocation** from that thread, without the GIL_ASYNC flag, to retrieve the icon location. It then calls **IExtractIcon::Extract** to extract the icon. Returning E_PENDING implies that the object is free threaded. In other words, it can be called concurrently by multiple threads safely.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IExtractIcon::Extract

IExtractImage

The **IExtractImage** interface is used to request a thumbnail image from a shell folder. There are two steps to the process. First, use **GetLocation** to request the path description of an image and specify how the image should be rendered. Then call **Extract** to extract the image.

If the object is free-threaded it also must expose an **IRunnableTask** interface, so it can be stopped and started as needed. This feature can be particularly useful when extraction may be slow.

When to Implement

Implement **IExtractImage** if your namespace extension needs to provide thumbnail images to be displayed in a shell view.

When to Use

Use **IExtractImage** if you are implementing a view of namespace objects, and want to display thumbnail images. You can use a shell folder's **IShellFolder::GetUIObjectOf** method to bind to its **IExtractImage** interface.

IExtractImage implements **IUnknown** and the following methods:

IExtractImage methods	Description
Extract	Used to request the image itself.
GetLocation	Used to request location of the file containing the image.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IExtractImage::Extract

Used to request an image from an object, such as an item in a shell folder.

```
HRESULT Extract(
    HBITMAP phBmpImage
);
```

Parameters

phBmpImage
[out] Buffer to hold the bitmapped image.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

Remarks

You must call **IExtractImage::GetLocation** prior to calling **Extract**.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IExtractImage::GetLocation

Used to request the path description of an image and specify how the image should be rendered.

```
HRESULT GetLocation(
    LPWSTR pszPathBuffer,
    DWORD cchMax,
    DWORD *pdwPriority
    const SIZE *prgSize
    DWORD dwRecClrDepth
    DWORD *pdwFlags
);
```

Parameters

pszPathBuffer
[out] Buffer used to return the path description. This value to identifies the image so you can avoid loading the same one more than once.

cchMax

[in] Size of *pszPathBuffer* in bytes.

pdwPriority

[out] Pointer used to return the priority of the item when the IEIFLAG_ASYNC flag is set in *pdwFlags*. This parameter is used normally to indicate the amount of time needed to extract the image. There are three standard priority levels:

Level	Description
IEI_PRIORITY_MAX	Maximum priority
IEI_PRIORITY_MIN	Minimum priority
IEI_PRIORITY_NORMAL	Normal priority

If you want more control over the order in which thumbnails are extracted, you can define as many priority levels as you wish (up to 32 bits). As long as the integer values assigned to the different levels increase from low to high priority, the actual numbers you use are not important. They are used only to determine the order in which the images will be extracted.

prgSize

[in] Pointer to a **SIZE** structure with the desired width and height of the image.

dwRecClrDepth

[in] Recommended color depth in units of bits per pixel.

pdwFlags

[in] Flags that specify how the image is to be handled. It can be a combination of the following:

Flag	Description
IEIFLAG_ASPECT	Used to ask the object to use the supplied aspect ratio. If this flag is set, a rectangle with the desired aspect ratio will be passed in *prgSize*. This flag cannot be used with IEIFLAG_SCREEN.
IEIFLAG_ASYNC	Used to ask the object if it supports asynchronous (free-threaded) extraction. If this flag is set, and **GetLocation** returns E_PENDING, the priority of the item should be returned in *pdwPriority*.
IEIFLAG_CACHE	Returned by the object to indicate that it will not cache the image. If this flag is returned, the shell will cache a copy of the image.
IEIFLAG_GLEAM	Used to ask the object if it has a gleam. If so, this flag should be set when the method returns.
IEIFLAG_OFFLINE	Used to tell the object to use only local content for rendering.

IEIFLAG_ORIGSIZE **Version 5.0**. Used to tell the object to render the image to the approximate size passed in *prgSize*, but crop it if necessary.

IEIFLAG_SCREEN Used to tell the object to render as if for the screen. This flag cannot be used with IEIFLAG_ASPECT.

Return Values

Value	Description
E_PENDING	If the IEIFLAG_ASYNC flag is set, this return value is used to indicate to the shell that the object is free-threaded.
NOERROR	Success.

It may also return an OLE-defined error code.

Remarks

If **GetLocation** is free-threaded, it can be placed in a background thread. The object must also expose an **IRunnableTask** interface, so the caller can start and stop the extraction process as needed.

You should return images that fit within the boundaries defined by *prgSize*. With Windows 2000 and later systems, you can set IEIFLAG_ORIGSIZE to use objects that do not have a standard aspect ratio, and they will be displayed properly. You do not need to fill in the unused part of the rectangle. If you try to use a nonstandard aspect ratio image with earlier versions of the shell, it will be stretched to fit the *prgSize* rectangle. Depending on how much the aspect ratio differs from what is specified, the image might be badly distorted.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IExtractImage2

The **IExtractImage2** interface extends the capabilities of **IExtractImage**.

When to Implement

Implement **IExtractImage2** to provide date stamps for your thumbnail images.

When to Use

You do not call this interface directly. **IExtractImage2** is used by the operating system only when it has confirmed that your application is aware of this interface.

IExtractImage2 implements all the **IExtractImage** methods as well as **IUnknown**. The following method is specific to **IExtractImage2**:

IExtractImage2 methods	Description
GetDateStamp	Used to request the date the thumbnail was last modified.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IExtractImage2::GetDateStamp

Used to request the date the image was last modified. This method allows the shell to determine whether or not cached images are out of date.

```
HRESULT GetDateStamp(
    FILETIME *pDateStamp
);
```

Parameters

pDateStamp
Pointer to a **FILETIME** structure used to return the last time the image was modified.

Return Values

Return NOERROR if successful, or an OLE-defined error code otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IFileViewer

The **IFileViewer** interface designates an interface that allows a registered file viewer to be notified when it must show or print a file.

Note File viewers are not supported by Windows 2000 and later systems.

When to Implement

You implement this interface to provide a means for your registered file types to be viewed and/or printed.

When to Use

You normally do not use this interface. The shell calls the interface when the user chooses the **Quick View** command from a file's context menu and the file is a type that the file viewer recognizes.

IFileViewer methods	Description
PrintTo	Prints a file.
Show	Displays a file.
ShowInitialize	Prepares to display a file.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IFileViewer::PrintTo

Prints a file. The shell specifies the name of the file to print by calling the file viewer's **IPersistFile::Load** method.

```
HRESULT PrintTo(
    LPSTR pszDriver,
    BOOL fSuppressUI
);
```

Parameters

pszDriver

Address of a buffer that contains the name of the printer device driver that should print the file. If this parameter is NULL, the file viewer determines which device driver to use.

fSuppressUI

User interface suppression flag. If this parameter is TRUE, the file viewer should not display any user interface, including error messages, during the print operation. If this parameter is FALSE, the file viewer can show dialog boxes, as needed.

Return Values

Returns the NOERROR value if successful, or an OLE-defined error value otherwise.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IFileViewer

IFileViewer::Show

Displays a file. The shell specifies the name of the file to display by calling the file viewer's **IPersistFile::Load** method.

```
HRESULT Show(
    LPFVSHOWINFO pvsi
);
```

Parameters

pvsi

Address of an **FVSHOWINFO** structure to contain information that the file viewer uses to display the file. A file viewer can return information to the shell by modifying the members of the structure.

Return Values

Returns NOERROR if successful, or E_UNEXPECTED if the **IFileViewer::ShowInitialize** method was not called before **IFileViewer::Show**.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IFileViewer

IFileViewer::ShowInitialize

Allows a file viewer to determine whether it can display a file and, if it can, to perform initialization operations before showing the file.

```
HRESULT ShowInitialize(
    LPFILEVIEWERSITE lpfsi
);
```

Parameters

lpfsi

Address of an **IFileViewerSite** interface. A file viewer uses this interface to retrieve the handle to the current pinned window or to specify a new pinned window.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

The shell calls this method before the **IFileViewer::Show** method. The shell specifies the name of the file to display by calling the file viewer's **IPersistFile::Load** method.

ShowInitialize must perform all operations that are prone to failure so that if it succeeds, **IFileViewer::Show** will not fail.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IFileViewer

IFileViewerSite

The **IFileViewerSite** interface designates an interface that allows a file viewer to retrieve the handle to the current pinned window or to set a new pinned window. The pinned window is the window in which the current file viewer displays a file. When the user selects a new file to view, the shell directs the file viewer to display the new file in the pinned window rather than create a new window.

Note File viewers are not supported by Windows 2000 and later systems.

When to Implement

You normally do not implement this interface. The shell implements this interface to provide a pinned window for the file viewer.

When to Use

You use this interface to obtain or set the window for a file viewer.

IFileViewerSite methods	Description
GetPinnedWindow	Retrieves the handle to the current pinned window.
SetPinnedWindow	Sets a new pinned window.

! Requirements

Version 4.0 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IFileViewerSite::GetPinnedWindow

Retrieves the handle to the current pinned window, if one exists.

```
HRESULT GetPinnedWindow(
    HWND *phwnd
);
```

Parameters

phwnd

Address of the handle to the current pinned window, or NULL if no pinned window exists.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IFileViewerSite

IFileViewerSite::SetPinnedWindow

Sets the pinned window. When the user selects a new file to view, the shell directs the file viewer to display the new file in the pinned window instead of creating a new window.

```
HRESULT SetPinnedWindow(
    HWND *hwnd
);
```

Parameters

hwnd

Handle to the new pinned window, or NULL if there is to be no pinned window.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IFileViewerSite

IInputObject

The **IInputObject** interface is used to change UI activation and process accelerators for a user input object contained in the shell.

When to Implement

Implement **IInputObject** if you are implementing a shell object that takes user input.

When to Use

You do not call this interface directly. **IInputObject** is used by the shell or the browser to notify the object of UI activation changes and to translate keyboard accelerators.

IInputObject is derived from **IUnknown**. The following methods are specific to **IInputObject**:

IInputObject methods	Description
HasFocusIO	Determines if one of the object's windows has the keyboard focus.
TranslateAcceleratorIO	Passes keyboard accelerators to the object.
UIActivateIO	Activates or deactivates the object.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IInputObject::HasFocusIO

Determines if one of the object's windows has the keyboard focus.

```
HRESULT HasFocusIO(void);
```

Return Values

Returns S_OK if one of the object's windows has the keyboard focus, or S_FALSE otherwise.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IInputObject::TranslateAcceleratorIO

Allows the object to process keyboard accelerators.

```
HRESULT TranslateAcceleratorIO(
    LPMSG lpMsg
);
```

Parameters

lpMsg

Address of an **MSG** structure that contains the keyboard message that is being translated.

Return Values

Returns S_OK if the accelerator was translated, or S_FALSE otherwise.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).

Windows CE: Unsupported.
Header: Declared in shlobj.h.

IInputObject::UIActivateIO

UI activates or deactivates the object.

```
HRESULT UIActivateIO(
    BOOL fActivate,
    LPMSG lpMsg
);
```

Parameters

fActivate

Indicates if the object is being activated or deactivated. If this value is nonzero, the object is being activated. If this value is zero, the object is being deactivated.

lpMsg

Address of an **MSG** structure that contains the message that caused the activation change. This value may be NULL.

Return Values

Returns S_OK if the activation change was successful, or an OLE-defined error code otherwise.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with
Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0
or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IInputObjectSite

The **IInputObjectSite** interface is used to communicate focus changes for a user input object contained in the shell.

When to Implement

You normally do not implement this interface. **IInputObjectSite** is implemented by the shell or the browser to maintain the input focus properly.

When to Use

You use **IInputObjectSite** if you are implementing a shell object that takes user input.

IInputObjectSite is derived from **IUnknown**. The following method is specific to **IInputObjectSite**:

IInputObjectSite method	Description
OnFocusChangeIS	Informs the browser that the focus has changed.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with
Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0
or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IInputObjectSite::OnFocusChangeIS

Informs the browser that the focus has changed.

```
HRESULT OnFocusChangeIS(
    IUnknown *punkObj,
    BOOL fSetFocus
);
```

Parameters

punkObj

Address of the **IUnknown** interface of the object gaining or losing the focus.

fSetFocus

Indicates if the object has gained or lost the focus. If this value is nonzero, the object has gained the focus. If this value is zero, the object has lost the focus.

Return Values

Returns S_OK if the method was successful, or an OLE-defined error code otherwise.

Remarks

The calling object should call this method whenever one of its windows gains or loses the input focus.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

INewShortcutHook

The **INewShortcutHook** interface is used when creating a new Internet shortcut.

When to Implement

You normally do not implement **INewShortcutHook**. It is implemented by the shell for Internet shortcuts.

When to Use

You use **INewShortcutHook** when creating a new Internet shortcut. The methods provided by this interface are supplied as a convenience.

INewShortcutHook is derived from **IUnknown**. The following methods are specific to **INewShortcutHook**:

INewShortcutHook methods	Description
GetExtension	Retrieves the file extension for the shortcut object.
GetFolder	Retrieves the folder name for the shortcut object.
GetName	Retrieves the file name of the shortcut object, without the extension.
GetReferent	Retrieves the referent of the shortcut object.
SetFolder	Sets the folder name for the shortcut object.
SetReferent	Sets the referent of the shortcut object.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

INewShortcutHook::GetExtension

Retrieves the file extension for the shortcut object.

```
HRESULT GetExtension(
    LPTSTR pszExtension,
    int cchExtension
);
```

Parameters

pszExtension
Address of a character buffer that receives the extension.

cchExtension
Size of the buffer at *pszExtension*, in characters.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

INewShortcutHook

INewShortcutHook::GetFolder

Retrieves the folder name for the shortcut object.

```
HRESULT GetFolder(
    LPTSTR pszFolder,
    int cchFolder
);
```

Parameters

pszFolder

Address of a character buffer that receives the folder name.

cchFolder

Size of the buffer at *pszFolder*, in characters.

Return Values

Returns S_OK if successful, S_FALSE if no folder has been assigned, or an OLE-defined error code otherwise.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

INewShortcutHook

INewShortcutHook::GetName

Retrieves the file name of the shortcut object, without the extension.

```
HRESULT GetName(
    LPTSTR pszName,
    int cchName
);
```

Parameters

pszName
Address of a character buffer that receives the name.

cchName
Size of the buffer at *pszName*, in characters.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

INewShortcutHook

INewShortcutHook::GetReferent

Retrieves the referent of the shortcut object.

```
HRESULT GetReferent(
    LPTSTR pszReferent,
    int cchReferent
);
```

Parameters

pszReferent
Address of a character buffer that receives the referent.

cchReferent
Size of the buffer at *pszReferent*, in characters.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

Remarks

For Internet shortcut objects, this method is the same as **IUniformResourceLocator::GetURL**.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

INewShortcutHook

INewShortcutHook::SetFolder

Sets the folder name for the shortcut object.

```
HRESULT SetFolder(
    LPTSTR pszFolder
);
```

Parameters

pszFolder
Address of a character buffer that contains the folder name.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

INewShortcutHook

INewShortcutHook::SetReferent

Sets the referent of the shortcut object.

```
HRESULT SetReferent(
    LPCTSTR pszReferent,
    HWND hWnd
);
```

Parameters

pszReferent
Address of a character buffer that contains the referent.

hWnd
Handle to the window that will be used as the parent if the object needs to display a message box or dialog box.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

Remarks

For Internet shortcut objects, this method is the same as **IUniformResourceLocator::SetURL**.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

INewShortcutHook

INotifyReplica

The **INotifyReplica** interface provides the object's creator with the means to notify an object that it might be subject to subsequent reconciliation. The briefcase reconciler is responsible for implementing this interface.

INotifyReplica method	Description
YouAreAReplica	Notifies an object that it might be subject to reconciliation.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in reconcil.h.

INotifyReplica::YouAreAReplica

Notifies an object that it might be subject to subsequent reconciliation through the **IReconcilableObject::Reconcile** method.

```
HRESULT INotifyReplica::YouAreAReplica(
    ULONG ulcOtherReplicas,
    IMoniker **rgpmkOtherReplicas
);
```

Parameters

ulcOtherReplicas
Number of other replicas of the object. This parameter must not be zero.

rgpmkOtherReplicas
Address of an array that contains the addresses of the monikers to use to access the other replicas.

Return Values

Returns S_OK if successful, or E_UNEXPECTED otherwise.

Remarks

An object can be notified that it is a replica more than once. Briefcase reconcilers are not required to implement this interface. Initiators are not required to call this interface if it is implemented. However, an object's implementation of **IReconcilableObject::Reconcile** can fail if that object has not been notified through **INotifyReplica::YouAreAReplica** previously that it may be subject to reconciliation.

The briefcase calls the **INotifyReplica** interface when objects are added to it.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in reconcil.h.

IObjMgr

The **IObjMgr** interface allows a client to append or remove an object from a collection of objects managed by a server object.

When to Implement

This interface is implemented by objects that manage a collection of other objects. It is exported to allow clients of the object to request that objects be added to or removed from the collection.

When to Use

Use this interface to add or delete an object from the server object's collection of managed objects.

Methods

IObjMgr exposes the following methods in addition to **IUnknown**:

Append Appends an object to the server object's collection of managed objects.

Remove Removes an object from the server object's collection of managed objects.

Requirements

Version 5.00 and later of Shell32.dll.
Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IObjMgr::Append

Appends an object to the collection of managed objects.

```
HRESULT Append(
    IUnknown *punk
);
```

Parameters

punk

[in] Address of the **IUnknown** interface of the object to be added to the list.

Return Values

Returns S_OK if successful, or an OLE error code otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.
Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IObjMgr, **IObjMgr::Remove**

IObjMgr::Remove

Removes an object from the collection of managed objects.

```
HRESULT Remove(
    IUnknown *punk
);
```

Parameters

punk

[in] Address of the **IUnknown** interface of the object to be removed from the list.

Return Values

Returns S_OK if successful, or an OLE error code otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IObjMgr, **IObjMgr::Append**

IPersistFolder

The **IPersistFolder** interface is used to initialize shell folder objects.

When to Implement

When implementing a shell namespace extension, specifically the **IShellFolder** interface, you need to implement this interface so the folder object can be initialized. Implementation of this interface is how the folder is told where it is in the shell namespace.

When to Use

You do not use this interface directly. It is used by the file system implementation of the **IShellFolder::BindToObject** interface when it is initializing a shell folder object.

IPersistFolder is derived from **IPersist**. The following method is specific to **IPersistFolder**:

IPersistFolder method	Description
Initialize	Instructs a shell folder object to initialize itself based on the information passed.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IPersistFolder::Initialize

Instructs a shell folder object to initialize itself based on the information passed.

```
HRESULT Initialize(
    LPCITEMIDLIST pidl,
);
```

Parameters

pidl

Address of the **ITEMIDLIST** (item identifier list) structure that specifies the absolute location of the folder.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

All objects that implement the **IShellFolder** interface for use in the shell's namespace must implement this method. When a folder's location in the namespace is not a relevant consideration, this method can return NOERROR. When the location is relevant to the folder, you should store the fully qualified IDLIST passed in for future reference.

For example, if the folder implementation needs to construct a fully qualified PIDL to elements that it contains, the PIDL passed to this method should be used to construct the fully qualified PIDLs.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IPersistFolder

IPersistFolder2

The **IPersistFolder2** interface is used to obtain information from shell folder objects.

When implementing a shell namespace extension, specifically the **IShellFolder** interface, you need to implement this interface so that the shell folder object's **ITEMIDLIST** can be retrieved.

IPersistFolder2 is derived from **IPersistFolder**. The following method is specific to **IPersistFolder2**:

Methods

GetCurFolder	Retrieves the **ITEMIDLIST** for the folder object.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IPersistFolder2::GetCurFolder

Retrieves the **ITEMIDLIST** for the folder object.

```
HRESULT GetCurFolder(
    LPITEMIDLIST *ppidl,
);
```

Parameters

ppidl

Address of an **ITEMIDLIST** pointer that receives a copy of the **PIDL** that represents the absolute location of the folder. This **PIDL** must be relative to the desktop. This normally will be a copy of the **PIDL** that was passed to **IPersistFolder::Initialize**.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).

Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IPersistFolder2

IPersistFolder3

The **IPersistFolder3** interface allows a folder object to implement nondefault handling of folder shortcuts. It is an extension of the **IPersistFolder2** interface.

When to Implement

Namespace extensions implement this interface if they need to perform nondefault handling of folder shortcuts.

When to Use

Applications normally do not use this interface directly.

Methods

This interface extends the **IPersistFolder** and **IPersistFolder2** interfaces. The following methods are specific to **IPersistFolder3**:

GetFolderTargetInfo	Provides the location and attributes of a folder shortcut's target folder.
InitializeEx	Initializes a folder, and specifies its location in the namespace. If the folder is a shortcut, the method also specifies the location of the target folder.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IPersistFileSystemFolder::GetFolderTargetInfo

Provides the location and attributes of a folder shortcut's target folder.

```
HRESULT GetFolderTargetInfo(
    PERSIST_FILESYSTEM_FOLDER_INFO *pfsfi
);
```

Parameters

pfsfi

out] Address of a **PERSIST_FOLDER_TARGET_INFO** structure that is used to return the target folder's location and attributes.

Return Values

Returns NOERROR if successful, or an OLE error code otherwise.

Remarks

The **PERSIST_FOLDER_TARGET_INFO** structure might not be initialized by the caller. **GetFolderTargetInfo** should assign values to all members of the structure before returning it to the caller.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IPersistFolder3

IPersistFileSystemFolder::InitializeEx

Initializes a folder, and specifies its location in the namespace. If the folder is a shortcut, the method also specifies the location of the target folder.

```
HRESULT InitializeEx(
  IBindCtx *pbc,
  LPCITEMIDLIST pidlRoot,
  const PERSIST_FOLDER_TARGET_INFO *ppfti
);
```

Parameters

pbc

[in] Address of an **IBindCtx** interface that provides the bind context. This parameter is set to NULL if not used.

pidlRoot

[in] Address of a fully qualified **PIDL** that specifies the absolute location of a folder or folder shortcut. The caller allocates and frees this PIDL.

ppfti

[in] Address of a **PERSIST_FOLDER_TARGET_INFO** structure that specifies the location of the target folder and its attributes. If *ppfti* points to a valid structure, *pidlRoot* represents a folder shortcut. If *ppfti* is set to NULL, *pidlRoot* represents a normal folder. In that case, **InitializeEx** should behave as if **IPersistFolder::Initialize** had been called.

Return Values

Returns S_OK if successful, or an OLE error code otherwise.

Remarks

This function is an extended version of **IPersistFolder::Initialize**. It allows the shell to initialize folder shortcuts as well as normal folders.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IPersistFolder3

IProgressDialog

The **IProgressDialog** interface is exported by the progress dialog box object (CLSID_ProgressDialog). This object is a generic way to show a user how an operation is progressing. It is used typically when deleting, uploading, copying, moving, or downloading large numbers of files.

The progress dialog box object creates a modeless dialog box and allows the client to set its title, animation, text lines, and progress bar. The object then handles updating on a background thread and allows the user to cancel the operation. Optionally, it estimates the time remaining until the operation is complete and displays the information as a line of text.

When to Implement

Applications normally do not implement this interface. It is exported by the progress dialog box object for use by applications.

When to Use

Use this interface when your application needs to display a progress dialog box. To initialize the object:

1. Create an in-process progress dialog box object (CLSID_ProgressDialog) with **CoCreateInstance**. Request a pointer to its **IProgressDialog** interface (IID_IProgressDialog).
2. Call **IProgressDialog::SetTitle** to specify the dialog box title.
3. Call **IProgressDialog::SetAnimation** to specify an AVI clip to be played while the operation progresses.
4. Call **IProgressDialog::SetCancelMsg** to specify the message that will be displayed if the user cancels the operation.

To display the progress of the operation:

1. Call **IProgressDialog::StartProgressDialog** to display the dialog box.
2. Assign a numerical value to the total amount of work the operation will perform. Use any number that allows you conveniently to define the progress of the operation. For example, set this value to 100 if you want to specify the progress of the operation in terms of the percent that has been completed.
3. Call **IProgressDialog::Timer** to reset the timer. This method sets the starting point that the progress dialog box object uses to estimate the time remaining in the operation. If you do not call this method, the starting point will be the call to **StartProgressDialog**.
4. As the operation progresses, call **IProgressDialog::SetProgress** periodically to update the dialog box as to how much of the operation has been completed. The progress dialog object will update its progress bar and recalculate its estimate of the remaining time. You can use any numerical measure of progress that is convenient.

However, if you want to use values larger than 4GB, you must call **IProgressDialog::SetProgress64** instead of **IProgressDialog::SetProgress**.

5. Your application does not receive a notification if the user clicks the Cancel button to cancel the operation. As the operation progresses, periodically call **IProgressDialog::HasUserCancelled** to see if the user has clicked the Cancel button. Applications typically call this method each time they call **IProgressDialog::SetProgress** or **IProgressDialog::SetProgress64**.
6. The dialog box displays three lines of text. An application periodically can call **IProgressDialog::SetLine** to display a message on one of these lines. This method is used normally to provide information on the current status of the operation. A typical message is something like: "Currently processing item XXX..". Messages are displayed normally on lines 1 and 2. You can display messages on line 3 only if you have not instructed the progress dialog box object to estimate the remaining time by setting the PROGDLG_AUTOTIME flag in the dwFlags parameter of **IProgressDialog::StartProgressDialog**. In that case, the third text line is used to display the estimated time.

When the operation is complete:

1. Call **IProgressDialog::StopProgressDialog** to close the dialog box.
2. Release the progress dialog box object.

IProgressDialog exposes the following methods in addition to **IUnknown**:

Methods	Description
HasUserCancelled	Checks whether the user has canceled the operation.
SetAnimation	Specifies an AVI clip that will run in the dialog box.
SetCancelMsg	Sets a message to be displayed if the user cancels the operation.
SetLine	Displays a message.
SetProgress	Updates the progress dialog box with the current state of the operation.
SetProgress64	Updates the progress dialog box with the current state of the operation.
SetTitle	Sets the title of the progress dialog box.
StartProgressDialog	Starts the progress dialog box.
StopProgressDialog	Stops the progress dialog box.
Timer	Resets the timer.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IProgressDialog::HasUserCancelled

Checks whether the user has canceled the operation.

```
BOOL HasUserCancelled(VOID);
```

Parameters

None.

Remarks

The system does not send a message to the application when the user clicks the **Cancel** button. You must use this function periodically to poll the progress dialog box object to determine whether the operation has been canceled.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IProgressDialog

IProgressDialog::SetAnimation

Specifies an AVI clip that will run in the dialog box.

```
HRESULT SetAnimation(
    HINSTANCE hInstAnimation,
    UINT idAnimation
);
```

Parameters

hInstAnimation
[in] Instance handle to the module from which the AVI resource should be loaded.

idAnimation

[in] AVI resource identifier. To create this value, use the MAKEINTRESOURCE macro. The control loads the AVI resource from the module specified by *hInstAnimation*.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

This method takes the instance handle specified by *hInstAnimation* and uses the common control's **animation control** to open and run a silent AVI clip. There are several restrictions as to what types of AVI clips can be used:

- Clips cannot include sound.
- The size of the AVI clip cannot exceed 272 by 60 pixels. Smaller rectangles can be used, but they might not be centered properly.
- AVI clips must be either uncompressed or compressed with run-length (BI_RLE8) encoding. If you attempt to use an unsupported compression type, no animation will be displayed.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IProgressDialog

IProgressDialog::SetCancelMsg

Sets a message to be displayed if the user cancels the operation.

```
HRESULT SetCancelMsg(
    LPCWSTR pwzCancelMsg,
    LPCVOID pvReserved
);
```

Parameters

pwzCancelMsg

[in] Pointer to a NULL-terminated Unicode string that contains the message to be displayed.

pvReserved
Reserved for future use.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

Even though the user clicks **Cancel**, the application cannot immediately call **IProgressDialog::StopProgressDialog** to close the dialog box. The application must wait until the next time it calls **IProgressDialog::HasUserCancelled** to discover that the user has canceled the operation. Since this delay might be significant, the progress dialog box provides the user with immediate feedback by clearing text lines 1 and 2, and displaying the cancel message on line 3. The message is intended to let the user know that the delay is normal and that the progress dialog box will be closed shortly. It typically is set to something like "Please wait while ...".

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IProgressDialog

IProgressDialog::SetLine

Displays a message.

```
HRESULT SetLine(
    DWORD dwLineNum,
    LPCWSTR pwzString,
    BOOL fCompactPath,
    LPCVOID pvReserved
);
```

Parameters

dwLineNum
[in] Line number on which the text is to be displayed. Currently, there are three lines—1, 2, and 3. If the PROGDLG_AUTOTIME flag was included in the *dwFlags* parameter when **IProgressDialog::StartProgressDialog** was called, only lines 1 and 2 can be used. The estimated time will be displayed on line 3.

pwzString

[in] NULL-terminated Unicode string that contains the text.

fCompactPath

[in] Value that is set to TRUE to have path strings compacted if they are too large to fit on a line. The paths are compacted with **PathCompactPath**.

pvReserved

[in] Reserved for future use.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

This function is used typically to display a message such as "Item XXX is now being processed." Normally, messages are displayed on lines 1 and 2, with line 3 reserved for the estimated time.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IProgressDialog

IProgressDialog::SetProgress

Updates the progress dialog box with the current state of the operation.

```
HRESULT SetProgress(
    DWORD dwCompleted,
    DWORD dwTotal
);
```

Parameters

dwCompleted

[in] Application-defined value that indicates what proportion of the operation has been completed at the time the method was called.

dwTotal

[in] Application-defined value that specifies what value *dwCompleted* will have when the operation is complete.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

Use any convenient numerical measure of the progress of the operation. To use values larger than 4 GB, you can call **IProgressDialog::SetProgress64** instead.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IProgressDialog

IProgressDialog::SetProgress64

Updates the progress dialog box with the current state of the operation.

```
HRESULT SetProgress64(
    ULONGLONG ullCompleted,
    ULONGLONG ullTotal
);
```

Parameters

ullCompleted
[in] Application-defined value that indicates what proportion of the operation has been completed at the time the method was called.

ullTotal
[in] Application-defined value that specifies what value *ullCompleted* will have when the operation is complete.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

This method has exactly the same function as **IProgressDialog::SetProgress** but allows you to use values larger than one DWORD (4 GB).

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IProgressDialog

IProgressDialog::SetTitle

Sets the title of the progress dialog box.

```
HRESULT SetTitle(
    LPCWSTR pwzTitle
);
```

Parameters

pwzTitle
[in] Pointer to a NULL-terminated Unicode string that contains the dialog box title.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IProgressDialog

IProgressDialog::StartProgressDialog

Starts the progress dialog box.

```
HRESULT StartProgressDialog(
    HWND hwndParent,
    IUnknown *punkEnableModless,
    DWORD dwFlags,
    LPCVOID pvReserved
);
```

Parameters

hwndParent

[in] Handle to the dialog box's parent window.

punkEnableModless

Reserved for future use.

dwFlags

Flags that control the operation of the progress dialog box. This can be a combination of the following values:

Flag	Description
PROGDLG_AUTOTIME	Automatically estimate the remaining time and display the estimate on line 3. If this flag is set, **IProgressDialog::SetLine** can be used only to display text on lines 1 and 2.
PROGDLG_MODAL	The progress dialog box will be modal to the window specified by *hwndParent*. By default, a progress dialog box is modeless.
PROGDLG_NORMAL	Normal progress dialog box behavior.
PROGDLG_NOMINIMIZE	Do not display a minimize button on the dialog box's caption bar.
PROGDLG_NOPROGRESSBAR	Do not display a progress bar. Normally, an application can determine quantitatively how much of the operation remains and pass periodically that value to **IProgressDialog::SetProgress**. The progress dialog box uses this information to update its progress bar. This flag is set typically when the calling application needs to wait for an operation to finish, but does not have any quantitative information it can use to update the dialog box.
PROGDLG_NOTIME	Do not show the "time remaining" text.

pvReserved

Reserved for future use.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IProgressDialog

IProgressDialog::StopProgressDialog

Stops the progress dialog box and removes it from the screen.

```
HRESULT StopProgressDialog(VOID);
```

Parameters

None.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IProgressDialog, **IProgressDialog::HasUserCancelled**

IProgressDialog::Timer

Resets the progress dialog box timer to zero.

```
HRESULT Timer(
    DWORD dwTimerAction,
    LPCVOID pvReserved
);
```

Parameters

dwTimerAction

[in] Flags that indicate the action that should be taken by the timer. Currently, there is only one value.

Flag	Description
PDTIMER_RESET	Resets the timer to zero. Progress will be calculated from the time this method is called.

pvReserved
Reserved for future use.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

The timer is used to estimate the remaining time. It is started when your application calls **IProgressDialog::StartProgressDialog**. Unless your application will start immediately, it should call **Timer** just before starting the operation. This practice ensures that the time estimates will be as accurate as possible. This method should not be called after the first call to **IProgressDialog::SetProgress**.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IProgressDialog

IQueryAssociations

When you create a file association by defining a file class and associating it with an application, a variety of related information can be stored in the registry. This interface is a utility exposed by the shell that simplifies the process of retrieving this information from HKEY_CLASSES_ROOT and HKEY_CURRENT_USER. A complete registry path or HKEY value is not required. Instead, you can retrieve information based on criteria such as the file name extension or executable name. For a discussion of file associations, see *Creating a File Association.*

When to Implement

This interface is exposed by the shell or by namespace extensions to simplify handling file associations.

When to Use

Use this interface if you need information from the registry related to file associations. For example, you can use this interface to retrieve information associated with a file name extension such as the command string of one of its verbs.

To use this interface, you must first get a pointer to it. Typically, you get an **IQueryInterface** pointer by calling a shell object's **IShellFolder::GetUIObjectOf** method. You can also get an interface pointer by calling **AssocCreate**. Then, initialize it with **IQueryAssociations::Init**. This method sets the root key that will be used when you call any of the remaining three methods to retrieve information from the registry. They will look only below the root key. You must release the interface when you no longer need it.

The **IQueryAssociations** interface is useful if you need to query the registry for information repeatedly. Once the interface is initialized, the overhead of calling the various methods is relatively small. There are also several related functions, listed in the See Also section, that allow you to get the same information from the registry with a single function call. While they are simpler to use, they incur the overhead of creating and initializing **IQueryAssociations** each time they are called. Because of this, they are best suited for single use.

IQueryAssociations exposes the following methods:

Method	Description
GetData	Retrieves binary data associated with a specified value.
GetEnum	Not implemented.
GetKey	Retrieves the HKEY value associated with a specified key.
GetString	Retrieves the string associated with a specified value.
Init	Initializes the interface and sets the root key.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

+ See Also

AssocQueryKey, **AssocQueryString**, **AssocQueryStringByKey**

IQueryAssociations::GetData

Searches for and retrieves file association-related binary data from the registry.

```
HRESULT GetData(
    ASSOCF flags,
    ASSOCDATA data,
    LPCWSTR pwszExtra,
    LPVOID pvOut,
    DWORD *pcbOut
);
```

Parameters

flags

[in] **ASSOCF** value that can be used to control the search.

data

[in] **ASSOCDATA** value that specifies the type of data that is to be returned.

pwszExtra

[in] Pointer to an optional NULL-terminated Unicode string with information about the location of the data. It is set normally to a shell verb such as **open**. Set this parameter to NULL if it is not used.

pvOut

[out] Pointer used to return the data value.

pcbOut

[out] Pointer to a value that will hold the size of *pvOut*, in bytes.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

+ See Also

IQueryAssociations

IQueryAssociations::GetEnum

This method is not implemented currently.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

+ See Also

IQueryAssociations

IQueryAssociations::GetKey

Searches for and retrieves a file association-related key from the registry.

```
HRESULT GetKey(
    ASSOCF flags,
    ASSOCKEY key,
    LPCWSTR pwszExtra,
    HKEY *phkeyOut
);
```

Parameters

flags
[in] ASSOCF value that can be used to control the search.

key
[in] ASSOCKEY value that specifies the type of key that is to be returned.

pwszExtra
[in] Pointer to an optional NULL-terminated Unicode string with information about the location of the key. It is set normally to a shell verb such as **open**. Set this parameter to NULL if it is not used.

phkeyOut
[out] Pointer to the key's HKEY value.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

See Also

IQueryAssociations

IQueryAssociations::GetString

Searches for and retrieves a file association-related string from the registry.

```
HRESULT GetString(
    ASSOCF flags,
    ASSOCSTR str,
    LPCWSTR pwszExtra,
    LPWSTR pwszOut,
    DWORD *pcchOut
);
```

Parameters

flags

[in] Flag that can be used to control the search. It can be any combination of the following **ASSOCF** values:

- ASSOCF_IGNOREBASECLASS
- ASSOCF_NOFIXUPS
- ASSOCF_NOTRUNCATE
- ASSOCF_NOUSERSETTINGS
- ASSOCF_REMAPRUNDLL
- ASSOCF_VERIFY

str

[in] **ASSOCSTR** value that specifies the type of string that is to be returned.

pwszExtra

[in] Pointer to an optional NULL-terminated Unicode string with information about the location of the string. It is set normally to a shell verb such as **open**. Set this parameter to NULL if it is not used.

pwszOut
: [out] Pointer to a NULL-terminated Unicode string used to return the requested string.

pcchOut
: [in/out] Pointer to a value that is set to the number of characters in the *pszOut* buffer. It returns the number of characters actually placed in the buffer. If the ASSOCF_NOTRUNCATE flag is set in *flags* and the buffer specified in *pcchOut* is too small, the function will return E_POINTER and *pcchOut* will point to the required size of the buffer.

Return Values

Returns S_OK if successful, E_POINTER if the buffer is too small, or an OLE error value otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

+ See Also

IQueryAssociations

IQueryAssociations::Init

Initializes the **IQueryAssociations** interface and sets the root key to the appropriate ProgID.

```
HRESULT Init(
    ASSOCF flags,
    LPCWSTR pwszAssoc,
    HKEY hkProgid,
    HWND hwnd
);
```

Parameters

flags
: [in] Flag that specifies how the search is to be initialized. It is set normally to NULL, but it can also take one of the following **ASSOCF** values:

- ASSOCF_INIT_BYEXENAME
- ASSOCF_INIT_DEFAULTTOFOLDER
- ASSOCF_INIT_DEFAULTTOSTAR

pwszAssoc

[in] Unicode string that is used to determine the root key. If a value is specified for *hkProgid*, set this parameter to NULL. Four types of string can be used.

String type	Description
CLSID	A CLSID GUID in the standard "{GUID}" format.
Executable name	The name of an application's .exe file. The ASSOCF_OPEN_BYEXENAME flag must be set in *flags*.
File name extension	A file name extension, such as .txt.
ProgID	An application's ProgID, such as Word.Document.8.

hkProgid

[in] HKEY value of the key that will be used as a root key. The search will look only below this key. If a value is specified for *pwszAssoc*, set this parameter to NULL.

hwnd

Reserved. Set to NULL.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

This method initializes the interface, and is called each time you need to specify a new root key. You can use *pwszAssoc* to specify a string, such as a file name extension or its associated ProgID, that identifies the root key. You also can specify the root key's HKEY value. **Init** will then use this information to locate the root key in the registry. Subsequent calls to the other **IQueryAssociations** methods will use it as their starting point and search for the information in the root key's subkeys.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

See Also

IQueryAssociations

IQueryInfo

The shell uses the **IQueryInfo** interface to retrieve flags and info tip information for an item that resides in an **IShellFolder** implementation. Info tips are displayed usually inside of a tooltip control.

When to Implement

Implement **IQueryInfo** to provide flags and text information that differs from the normal text that is displayed for an item in a folder. For example, if your folder contained file objects, you could use the info tip to provide the entire path and file name for the item rather than just the file name.

This interface is obtained by calling **IShellFolder::GetUIObjectOf** and passing IID_IQueryInfo for the interface identifier. The item for which info tip information is being requested is contained in the *apidl* argument of the **IShellFolder::GetUIObjectOf** call. If IQueryInfo is not supplied by the folder, the shell will use the standard display text in the info tip.

When to Use

In most cases, you do not use this interface directly. The shell will use this interface when it requires additional information to display inside of an info tip. However, you can use IQueryInfo directly if you want to obtain info tip information from another object.

IQueryInfo methods	Description
GetInfoFlags	Retrieves the information flags for an item.
GetInfoTip	Retrieves the info tip text for an item.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IQueryInfo::GetInfoFlags

Retrieves the information flags for an item. This method is not used currently.

```
HRESULT GetInfoFlags(
    DWORD *pdwFlags
);
```

Parameters

pdwFlags

[out] Pointer to a value that receives the flags for the item. If no flags are to be returned, this value should be set to zero.

Return Values

Returns S_OK if *pdwFlags* returns any flag values, or an OLE-defined error value otherwise.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IQueryInfo

IQueryInfo::GetInfoTip

Retrieves the info tip text for an item.

```
HRESULT GetInfoTip(
    DWORD dwFlags,
    LPWSTR *ppwszTip
);
```

Parameters

dwFlags

Not used currently.

ppwszTip

[out] Address of a Unicode string pointer that receives the tip string pointer. Applications that implement this method must allocate memory for *ppwszTip* by calling **SHGetMalloc**. Calling applications must call **SHGetMalloc** to free the memory when it is no longer needed.

Return Values

Returns S_OK if the function succeeds. If no info tip text is available, *ppwszTip* is set to NULL. Otherwise, returns an OLE-defined error value.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IQueryInfo

IReconcilableObject

The **IReconcilableObject** interface carries out the reconciliation of a document. The briefcase reconciler is responsible for implementing this interface.

IReconcilableObject methods	Description
GetProgressFeedbackMaxEstimate	Receives estimate of work required to complete a reconciliation.
Reconcile	Reconciles the state of an object with one or more other objects.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in reconcil.h.

IReconcilableObject::GetProgressFeedbackMaxEstimate

Retrieves an estimated measurement of the amount of work required to complete a reconciliation. Reconcilers typically use this method to estimate the work needed to reconcile an embedded document. This value corresponds to a similar value that is passed with the **IReconcileInitiator::SetProgressFeedback** method during reconciliation.

```
HRESULT IReconcilableObject::GetProgressFeedbackMaxEstimate(
    IMoniker ** pulProgressMax
);
```

Parameters

pulProgressMax

Address of the variable to receive the work estimate value.

Return Values

Returns S_OK if successful, or one of the following error values otherwise:

E_UNEXPECTED	Unspecified error.
OLE_E_NOTRUNNING	The object is an OLE embedded document that must be run before this operation can be carried out. The object state is unchanged as a result of the call.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in reconcil.h.

See Also

IReconcilableObject

IReconcilableObject::Reconcile

Reconciles the state of an object with one or more other objects. The reconciliation updates the internal state of the object by merging the states of all objects to form a combined state.

```
HRESULT IReconcilableObject::Reconcile(
    IReconcileInitiator *pInitiator,
    DWORD dwFlags,
    HWND hwndOwner,
    HWND hwndProgressFeedback,
    ULONG ulcInput,
    IMoniker **rgpmkOtherInput,
    LONG *plOutIndex,
    IStorage *pstgNewResidues,
    void *pvReserved
);
```

Parameters

pInitiator

Address of the **IReconcileInitiator** interface for the initiator of the reconciliation process. This parameter must not be NULL.

dwFlags

Control flags for the reconciliation. This parameter can be zero or a combination of these values:

RECONCILEF_FEEDBACKWINDOWVALID

The *hwndProgressFeedback* parameter is valid.

RECONCILEF_MAYBOTHERUSER

The briefcase reconciler can prompt for user interaction if it is needed. Without this value, user interaction is not permitted. The *hwndOwner* parameter is valid.

RECONCILEF_NORESIDUESOK

The briefcase reconciler can ignore requests for residues and carry out reconciliation. Reconcilers that do not support residues should check for this value whenever an initiator requests residues. Without this value, a reconciler that does not support residues must return immediately REC_E_NORESIDUES.

RECONCILEF_OMITSELFRESIDUE

The briefcase reconciler can discard any residue associated with this object. Initiators typically use this value for reconciliations that loop from generation to generation.

RECONCILEF_ONLYYOUWERECHANGED

The **Reconcile** method is being called to propagate changes in the changed object to other unchanged objects. This value will be set only if the following key exists in the registry.

```
HKEY_CLASSES_ROOT\
  CLSID\
    <clsid_of_reconciler>\
      SingleChangeHook
```

If this key is not present in the registry, the initiator carries out reconciliation by making the other unchanged objects binary identical copies of the changed object. The *rgpmkOtherInput* monikers identify the other objects. This value will be set only in *dwFlags* if RECONCILEF_YOUMAYDOTHEUPDATES is also set. If the briefcase reconciler completes the updates itself successfully, REC_S_IDIDTHEUPDATES should be returned and the variable pointed to by the *plOutIndex* parameter should be set to –1L. Note that S_OK should not be returned on success if this value is set in *dwFlags*. The initiator will not save the source object's storage if **Reconcile** returns REC_S_IDIDTHEUPDATES. If the reconciler wants to fall back to the initiator's bit copy implementation, it can return S_FALSE.

RECONCILEF_RESUMEDRECONCILIATION
The briefcase reconciler should resume reconciliation, using the partial residues provided. Without this value, the reconciler should ignore any "considered but rejected" information in any of the input versions.

RECONCILEF_YOUMAYDOTHEUPDATES
The briefcase reconciler can perform the updates. Without this value, the reconciler cannot perform the updates. If reconciliation is completed successfully, the reconciler should return REC_S_IDIDTHEUPDATES if it performed the updates or S_OK if it did not perform the updates.

hwndOwner
Handle to the window to be used as the parent for any child windows that the briefcase reconciler creates. This parameter is valid only if RECONCILEF_MAYBOTHERUSER is specified in *dwFlags*.

hwndProgressFeedback
Handle to the progress feedback window to be displayed by the initiator. This parameter is valid only if RECONCILEF_FEEDBACKWINDOWVALID is specified in *dwFlags*. The briefcase reconciler may call the **SetWindowText** function using this window handle to display additional reconciliation status information to the user.

ulcInput
Number of versions or partial residues specified in *dwFlags*. This parameter must not be zero.

rgpmkOtherInput
Address of an array that contains the addresses of the monikers to use to access the versions or partial residues to be reconciled.

plOutIndex
Address of the variable that receives an index value indicating whether the result of the reconciliation is identical to one of the initial versions. The variable is set to –1L if the reconciliation result is a combination of two or more versions. Otherwise, it is a zero-based index, with 0 indicating this object, 1 indicating the first version, 2 indicating the second version, and so on.

pstgNewResidues

Address of the **IStorage** interface used to store the new residues. This parameter can be NULL to indicate that residues should not be saved.

pvReserved

Reserved; must be NULL.

Return Values

Returns one of the following values if successful:

S_OK	Reconciliation completed successfully, and the changes must be propagated to the other objects.
S_FALSE	No reconciliation actions were performed. The briefcase reconciler wishes to fall back to the initiator's bit copy implementation. This value may only be returned if RECONCILEF_ONLYYOUWERECHANGED is set in *dwFlags*.
REC_S_IDIDTHEUPDATES	Reconciliation was completed successfully, and all the objects involved (the object implementing the **Reconcile** method and all the other objects described by *rgpmkOtherInput*) have been updated appropriately. The initiator does not need, therefore, to do anything further to propagate the changes. The variable pointed to by *plOutIndex* should be set to –1L if **Reconcile** returns this value. The initiator will not save the source object's storage if **Reconcile** returns this value. This value may only be returned if RECONCILEF_YOUMAYDOTHEUPDATES was set in *dwFlags*.
REC_S_NOTCOMPLETE	The briefcase reconciler completed some, but not all, of the reconciliation. It may need user interaction. The changes will not be propagated to other objects.
REC_S_NOTCOMPLETEBUTPROPAGATE	The briefcase reconciler completed some, but not all, of the reconciliation. It may need user interaction. The changes will be propagated to the other objects.

Otherwise, this method returns one of the following error values:

REC_E_NORESIDUES
The briefcase reconciler does not support the generation of residues, so the request for residues is denied. The state of the object is unchanged.

REC_E_ABORTED
The briefcase reconciler stopped reconciliation in response to a termination request from the initiator (see **IReconcileInitiator::SetAbortCallback** for more information). The state of the object is unspecified.

REC_E_TOODIFFERENT
Reconciliation cannot be carried out because the provided document versions are too dissimilar.

REC_E_INEEDTODOTHEUPDATES
The RECONCILEF_YOUMAYDOTHEUPDATES flag was not set when the object's **Reconcile** implementation was called; this implementation requires that this value be set in the *dwFlags* parameter.

OLE_E_NOTRUNNING
The object is an OLE embedded object that must be run before this operation can be carried out. The state of the object is unchanged.

E_UNEXPECTED
Unspecified error.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in reconcil.h.

+ See Also

IReconcilableObject

IReconcileInitiator

The **IReconcileInitiator** interface provides the briefcase reconciler with the means to notify the initiator of its progress, to set a termination object, and to request a given version of a document. The initiator is responsible for implementing this interface.

IReconcileInitiator methods	Description
SetAbortCallback	Sets the object through which the initiator can terminate a reconciliation.
SetProgressFeedback	Indicates the amount of progress in the reconciliation.

! Requirements

Version 4.00 and later of shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in reconcil.h.

IReconcileInitiator::SetAbortCallback

Sets the object through which the initiator can asynchronously terminate a reconciliation. A briefcase reconciler typically sets this object for reconciliations that are lengthy or involve user interaction.

```
HRESULT IReconcileInitiator::SetAbortCallback(
    IUnknown *pUnkForAbort
);
```

Parameters

pUnkForAbort

Address of the **IUnknown** interface for the object. The initiator signals a request to terminate the reconciliation by using the **IUnknown::Release** method to release the object. This parameter may be NULL to direct the initiator to remove the previously specified object.

Return Values

Returns the S_OK value if successful, or one of the following error values otherwise:

REC_E_NOCALLBACK	The initiator does not support termination of reconciliation operations and does not hold the specified object.
E_UNEXPECTED	Unspecified error.

Remarks

The initiator can accept or reject the object. If the initiator accepts the object, the briefcase reconciler must remove the object by calling this method with a NULL parameter when the reconciliation is complete. Because the reconciler removes the object after completing reconciliation, there may be times when the initiator releases the object after reconciliation is complete. In such cases, the reconciler ignores the request to terminate.

If the reconciliation is terminated, the **IReconcilableObject::Reconcile** method must return either the REC_E_ABORTED or REC_E_NOTCOMPLETE value.

! Requirements

Version 4.00 and later of shell32.dll

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in reconcil.h.

+ See Also

IReconcileInitiator

IReconcileInitiator::SetProgressFeedback

Indicates the amount of progress the briefcase reconciler has made toward completing the reconciliation. The amount is a fraction and is computed as the quotient of the *ulProgress* and *ulProgressMax* parameters. Reconcilers should call this method periodically during their reconciliation process.

```
HRESULT IReconcileInitiator::SetProgressFeedback(
    ULONG ulProgress,
    ULONG ulProgressMax
);
```

Parameters

ulProgress
Numerator of the progress fraction.

ulProgressMax
Denominator of the progress fraction.

Return Values

Returns the S_OK value if successful, or the E_UNEXPECTED value if an unspecified error occurred.

Remarks

The initiator typically uses this measure of progress to update a thermometer gauge or some other form of visual feedback for the user. The briefcase reconciler can change the value of *ulProgressMax* from call to call. This means successive calls to this method do not necessarily indicate steady forward progress. Backward progress is legal, although not desirable. It is the responsibility of the initiator to determine whether backward progress should be revealed to the user.

! Requirements

Version 4.00 and later of shell32.dll

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in reconcil.h.

See Also

IReconcileInitiator

IRemoteComputer

The **IRemoteComputer** interface is used to enumerate or initialize a namespace extension when it is invoked on a remote object. This interface is used, for example, to initialize the remote printers virtual folder.

When to Implement

Implement **IRemoteComputer** when your namespace extension may be invoked on a remote computer.

When to Use

You do not call this interface directly. **IRemoteComputer** is used by the operating system only when it has confirmed that your application is aware of this interface.

IRemoteComputer implements **IUnknown** and the following methods:

RemoteComputer methods	Description
Initialize	Used by the Explorer to initialize or enumerate a namespace extension invoked on a remote object.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IRemoteComputer::Initialize

Used by Windows Explorer or Internet Explorer when it is initializing or enumerating a namespace extension invoked on a remote computer.

```
HRESULT Initialize(
    WCHAR *pszMachine,
    BOOL bEnumerating
);
```

Parameters

pszMachine
Machine name of the remote computer.

bEnumerating
Value that is set to TRUE if Windows Explorer is enumerating the namespace extension, or FALSE if it is initializing it.

Return Values

Returns S_OK if successful, or standard OLE error values otherwise.

Remarks

If failure is returned, the extension won't appear for the specified computer. Otherwise, the extension will appear and target the remote computer.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IResolveShellLink

The **IResolveShellLink** interface allows an application to request that a shell folder object resolve a link for one of its items. With namespace extensions, shortcut objects (.lnk files) implement the essential functionality of **IShellLink::Resolve** by calling **IResolveShellLink::ResolveShellLink**. **IResolveShellLink** is exported by a link resolution object that is created on request by the shell folder. To get a pointer to a link resolution object's **IResolveShellLink** interface:

- For an object that is contained by a folder, call the folder's **IShellFolder::GetUIObjectOf** method and request an **IResolveShellLink** pointer (IID_IResolveShellLink).
- For the folder object itself, call the folder's **IShellFolder::CreateViewObject** method and request an **IResolveShellLink** pointer (IID_IResolveShellLink).

When to Implement

Namespace extensions implement this object to support link resolution.

When to Use

This interface is not normally used by applications.

Methods

IResolveShellLink exposes the following method in addition to **IUnknown**:

ResolveShellLink Requests that a folder object resolve a shell link.

❗ Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IResolveShellLink::ResolveShellLink

Requests that a folder object resolve a shell link.

```
HRESULT ResolveShellLink(
    IUnknown *punk,
    HWND hwnd,
    DWORD fFlags
);
```

Parameters

punk
[in] Address of the object's **IShellLink** interface. This interface can be queried to determine the contents of the link.

hwnd
[in] Handle to the window that the shell will use as the parent for a dialog box. The shell displays the dialog box if it needs to prompt the user for more information while resolving a shell link.

fFlags
[in] Action flags. This parameter can be a combination of the following values:

SLR_INVOKE_MSI Call the Microsoft Windows Installer.

SLR_NOLINKINFO Disable distributed link tracking. By default, distributed link tracking tracks removable media across multiple devices based on the volume name. It also uses the UNC path to track remote file systems whose drive letter has changed. Setting SLR_NOLINKINFO disables both types of tracking.

SLR_NO_UI Do not display a dialog box if the link cannot be resolved. When SLR_NO_UI is set, the high-order word of *fFlags* specifies a time-out duration, in milliseconds. The function returns if the link cannot be resolved within the time-out duration. If the high-order word is set to zero, the time-out duration defaults to 3000 milliseconds (3 seconds).

SLR_NOUPDATE	Do not update the link information.
SLR_NOSEARCH	Do not execute the search heuristics.
SLR_NOTRACK	Do not use distributed link tracking.
SLR_UPDATE	If the link object has changed, update its path and list of identifiers. If SLR_UPDATE is set, you do not need to call **IPersistFile::IsDirty** to determine whether the link object has changed.

Return Values

Returns S_OK if successful, or an OLE error code otherwise.

Remarks

This method should attempt to find the target of a shell link, even if the target has been moved or renamed.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IResolveShellLink

IRunnableTask

The **IRunnableTask** interface is a free-threaded interface that can be exposed by a namespace extension. It allows a scheduler to manage processes such as the extraction of thumbnail images in conjunction with the **IExtractImage** interface.

When to Implement

Implement **IRunnableTask** if your namespace extension is free-threaded, and you want to allow a task such as icon extraction to be managed by a scheduler. Only the **Run** and **IsRunning** methods must be implemented. If you don't wish to implement **Kill**, **Resume**, and **Suspend**, simply have them return E_NOTIMPL. If an object exposes **IExtractImage**, **Run** is not necessary, as the system will use **IExtractImage::Extract** to manage the task.

When to Use

You do not call this interface directly. **IRunnableTask** is used by the operating system only when it has confirmed that your application is aware of this interface.

IRunnableTask implements **IUnknown** and the following five methods:

IRunnableTask methods	Description
IsRunning	Used to determine the current state of the task.
Kill	Use to stop execution.
Resume	Use to resume execution.
Run	Use to start execution
Suspend	Use to suspend execution

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IRunnableTask::IsRunning

Used to request information on the state of a task, such as thumbnail extraction.

```
LONG IsRunning( void );
```

Return Values

Returns one of the following values to indicate the current execution state:

IRTIR_TASK_NOT_RUNNING	Extraction has not yet started.
IRTIR_TASK_RUNNING	The task is running.
IRTIR_TASK_SUSPENDED	The task is suspended.
IRTIR_TASK_PENDING	The thread has been killed but has not completely shut down yet.
IRTIR_TASK_FINISHED	The task is finished.

Remarks

This method must be implemented.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IRunnableTask::Kill

Used to request that a task be stopped.

```
HRESULT Kill(
    BOOL fUnused
);
```

Parameters

fUnused
Not used currently.

Return Values

Returns NOERROR if successful, or standard OLE-defined error codes otherwise.

Remarks

Implementation of this method is optional. If you do not wish to support this functionality, create a token implementation that simply returns E_NOTIMPL.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IRunnableTask::Resume

Used to request that a task resume.

```
HRESULT Resume( void );
```

Return Values

Return NOERROR if successful, or standard OLE-defined error codes otherwise.

Remarks

Implementation of this method is optional. If you do not wish to support this functionality, create a token implementation that simply returns E_NOTIMPL.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IRunnableTask::Run

Used to request that a task start.

```
HRESULT Run( void );
```

Return Values

Return NOERROR when execution is complete. If execution is suspended, **Run** should return E_PENDING. Return standard OLE-defined error codes otherwise.

Remarks

This method must be implemented, unless your object exposes **IExtractImage**. In that case, the system will use **IExtractImage::Extract** to manage the task.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IRunnableTask::Suspend

Used to request that a task be suspended.

```
HRESULT Suspend( void );
```

Return Values

Return NOERROR if successful, or standard OLE-defined error codes otherwise.

Remarks

Implementation of this method is optional. If you do not wish to support this functionality, create a token implementation that simply returns E_NOTIMPL.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellBrowser

The **IShellBrowser** interface provides services for namespace extensions and is the companion to the **IShellView** interface implemented by namespace extensions. It is similar to the site interfaces that are often found in OLE hosting situations, such as the **IOleControl** and **IOleControlSite** interfaces. This allows the extension to communicate with the host of the namespace by providing UI elements like menus, status text, and tool bars. This interface also provides the extension with a way to access storage to save its persistent view state.

IShellBrowser derives from the **IOleWindow** interface, and it represents the container's top-level window. **IShellBrowser** allows the contained views to insert their menus into the composite menu, install the composite menu into the appropriate window frame, and remove the container's menu elements from the composite menu. **IShellBrowser** sets and displays status text relevant to the in-place object. It also enables or disables the frame's modeless dialog boxes and translates accelerator keystrokes intended for the container's frame.

When to Implement

You do not implement this interface directly. **IShellBrowser** is implemented by Windows Explorer and by the Windows File Open common dialog box.

When to Use

When implementing a namespace extension, notably the **IShellView** interface, you will use the **IShellBrowser** implementation that is passed to **IShellView::CreateViewWindow** to communicate with Windows Explorer.

IShellBrowser is derived from **IOleWindow**. The following are the methods specific to **IShellBrowser**:

IShellBrowser methods	Description
BrowseObject	Tells Windows Explorer to browse in another folder.
EnableModelessSB	Enables or disables modeless windows of Windows Explorer, such as a floating toolbar.
GetControlWindow	Gets the window handle to a Windows Explorer control.
GetViewStateStream	Returns a view-specific stream that can be used to read and write the persistent data for a view.
InsertMenusSB	Inserts Windows Explorer's menu items to an empty menu created by the view.

(continued)

(continued)

IShellBrowser methods	Description
OnViewWindowActive	Informs Windows Explorer that the view was activated.
QueryActiveShellView	Returns the currently activated (displayed) shell view object.
RemoveMenusSB	Instructs the container to remove its items from a composite menu. The tasks it performs are the opposite of InsertMenusSB tasks.
SendControlMsg	Sends messages to Windows Explorer controls.
SetMenuSB	Installs the composite menu in Windows Explorer.
SetStatusTextSB	Sets and displays status text in the Windows Explorer window.
SetToolbarItems	Adds toolbar items to Windows Explorer's toolbar.
TranslateAcceleratorSB	Reserved for future use.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellBrowser::BrowseObject

Tells Windows Explorer to browse to another folder.

```
HRESULT BrowseObject(
    LPCITEMIDLIST pidl,
    UINT *wFlags
);
```

Parameters

pidl

Address of an **ITEMIDLIST** (item identifier list) structure that specifies an object's location. This value is dependent on the flag or flags set in the *wFlags* parameter.

wFlags

Flags specifying the folder to be browsed. It can be zero or one or more of the following values. These first three flags specify whether another window is to be created.

SBSP_SAMEBROWSER	Browse to another folder with the same Windows Explorer window.
SBSP_NEWBROWSER	Creates another window for the specified folder.
SBSP_DEFBROWSER	The default behavior is to respect the view option (the user setting to create new windows or to browse in place). In most cases, callers should use this flag.

The following flags specify either the open, explore, or default mode. These values are ignored if SBSP_SAMEBROWSER is specified or if SBSP_DEFBROWSER is specified and the user has selected Browse In Place:

SBSP_OPENMODE	Use a normal folder window.
SBSP_EXPLOREMODE	Use a Windows Explorer window.
SBSP_DEFMODE	Use the current window.

The following flags specify the *pidl* parameter's category:

SBSP_ABSOLUTE	An absolute pidl (relative from the desktop).
SBSP_RELATIVE	A relative pidl (relative from the current folder).
SBSP_PARENT	Browse the parent folder (ignores the pidl).
SBSP_NAVIGATEBACK	Navigate back (ignores the pidl).
SBSP_NAVIGATEFORWARD	Navigate forward(ignores the pidl).

The following two flags control how history is manipulated as a result of navigation:

SBSP_WRITENOHISTORY	Write no history (shell folder) entry.
SBSP_NOAUTOSELECT	Suppress selection in the history pane.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

Views can use this method to force Windows Explorer to browse to a specific place in the namespace. Typically, these are folders contained in the view.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellBrowser

IShellBrowser::EnableModelessSB

Tells Windows Explorer to enable or disable its modeless dialog boxes.

```
HRESULT EnableModelessSB(
    BOOL fEnable
);
```

Parameters

fEnable

Specifies whether the modeless dialog boxes are to be enabled or disabled. If this parameter is nonzero, modeless dialogs are enabled. If this parameter is zero, modeless dialogs are disabled.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

This method is similar to the **IOleInPlaceFrame::EnableModeless** method. Although the current version of Windows Explorer does not have any modeless dialog boxes, the view should call this method when it wants to disable or enable modeless dialog boxes associated with the Windows Explorer window.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellBrowser

IShellBrowser::GetControlWindow

Retrieves the window handle to a browser control.

```
HRESULT GetControlWindow(
    UINT id,
    HWND *lphwnd
);
```

Parameters

id

Control handle that is being requested. This parameter can be one of the following values:

FCW_TOOLBAR	Retrieves the window handle to the browser's toolbar.
FCW_STATUS	Retrieves the window handle to the browser's status bar.
FCW_TREE	Retrieves the window handle to the browser's tree view.
FCW_PROGRESS	Retrieves the window handle to the browser's progress bar.

lphwnd

Address of the window handle to the Windows Explorer control.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

GetControlWindow is used so views can directly manipulate the browser's controls. FCW_TREE should be used only to determine if the tree is present.

Note to Callers **GetControlWindow** is used to manipulate and test the state of the control windows. Do not send messages directly to these controls; instead, use **IShellBrowser::SendControlMsg**. Be prepared for this method to return NULL. Future versions of Windows Explorer may not include a toolbar, status bar, or tree window.

Note to Implementers **GetControlWindow** returns the window handle to these controls if they exist in your implementation.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellBrowser

IShellBrowser::GetViewStateStream

Retrieves an **IStream** interface that can be used for storage of view-specific state information.

```
HRESULT GetViewStateStream(
    DWORD grfMode,
    LPSTREAM *ppStrm
);
```

Parameters

grfMode

Read/write access of the **IStream** interface. This may be one of the following values:

STGM_READ	Requests an **IStream** suitable for reading.
STGM_WRITE	Requests an **IStream** suitable for writing.
STGM_READWRITE	Requests an **IStream** suitable for reading and writing.

ppStrm

Address that receives the **IStream** interface pointer.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

This method is used to save and restore the persistent state for a view (the icon positions, the column widths, and the current scroll position, for example).

Note to Callers Use **GetViewStateStream** when the view is being created to read in the saved view state and also when the view is being closed to save any changes to the view state. Typically, the view calls this method with STGM_READ when creating a view window and with STGM_WRITE when the **SaveViewState** method of its **IShellView** interface is called.

Note to Implementers Each shell view should have its own view stream. Windows Explorer implements a most recently used (MRU) list of view streams that are stored on a per-user basis in the registry.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellBrowser

IShellBrowser::InsertMenusSB

Allows the container to insert its menu groups into the composite menu that is displayed when an extended namespace is being viewed or used.

```
HRESULT InsertMenusSB(
    HMENU hmenuShared,
    LPOLEMENUGROUPWIDTHS lpMenuWidths
);
```

Parameters

hmenuShared
Handle to an empty menu.

lpMenuWidths
Address of an OLEMENUGROUPWIDTHS array of six LONG values. The container fills in elements 0, 2, and 4 to reflect the number of menu elements it provided in the File, View, and Window menu groups.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

This method is similar to the **IOleInPlaceFrame::InsertMenus** method. Windows Explorer puts **File** and **Edit** drop-down menus in the **File** menu group, **View** and **Tools** menus in the Container menu group, and a **Help** menu in the Window menu group. Each drop-down menu will have a unique identifier, FCIDM_MENU_FILE/EDIT/VIEW/TOOLS/HELP. The view is allowed to insert menu items into those submenus by their identifiers, which is different from OLE's in-place activation mechanism. The command identifiers for menus that the view inserts into either Windows Explorer's submenus or its own submenus must be between FCIDM_SHVIEWFIRST and FCIDM_SHVIEWLAST.

Note to Callers This method is called by namespace extensions when they are first being activated so they can insert their menus into the frame-level user interface.

The object application asks the container to add its menus to the menu specified in the *hmenuShared* parameter and to set the group counts in the OLEMENUGROUPWIDTHS array pointed to by the *lpMenuWidths* parameter. The object application then adds its own menus and counts. Objects can call the **IOleInPlaceFrame::InsertMenus** method as many times as necessary to build up the composite menus. The container should use the initial menu handle associated with the composite menu for all items in the drop-down menus.

Note to Implementers For **IShellBrowser** implementations, the menu identifiers must be in the range of FCIDM_BROWSERFIRST to FCIDM_BROWSERLAST.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellBrowser::OnViewWindowActive

Called by the shell view when the view window or one of its child windows gets the focus or becomes active.

```
HRESULT OnViewWindowActive(
    IShellView *ppshv
);
```

Parameters

ppshv
Address of the view object's **IShellView** pointer.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

The view must pass its **IShellView** implementation to this routine, although the current version of Windows Explorer does not use this parameter.

Note to Callers The shell view object must call this method before calling the **IShellBrowser::InsertMenusSB** method. This method will insert a different set of menu items depending on whether the view has the focus.

Note to Implementers This method informs the browser that the view is getting the focus (when the mouse is clicked on the view, for example).

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellBrowser

IShellBrowser::QueryActiveShellView

Returns the currently active (displayed) shell view object.

```
HRESULT QueryActiveShellView(
    IShellView **ppshv
);
```

Parameters

ppshv
Address of the pointer to the currently active shell view object.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

Note to Callers Because the **IShellBrowser** interface can host several shell views simultaneously, this method provides an easy way to determine the active shell view object.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellBrowser::RemoveMenusSB

Permits the container to remove any of its menu elements from the in-place composite menu and to free all associated resources.

```
HRESULT RemoveMenusSB(
    HMENU hmenuShared
);
```

Parameters

hmenuShared

Handle to the in-place composite menu that was constructed by calls to **IShellBrowser::InsertMenusSB** and the Win32 **InsertMenu** function.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

This method is similar to the **IOleInPlaceFrame::RemoveMenus** method.

The object should always permit the container to remove its menu elements from the composite menu before deactivating the shared user interface.

Note to Callers The method is called by the object application while it is being UI-deactivated so the browser can remove its menus.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellBrowser

IShellBrowser::SendControlMsg

Sends control messages to either the toolbar or the status bar in an Explorer window.

```
HRESULT SendControlMsg(
    UINT id,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam,
    LRESULT *pret
);
```

Parameters

id

Identifier for either a toolbar (FCW_TOOLBAR) or for a status bar window (FCW_STATUS).

uMsg

Message to be sent to the control.

wParam

Value depends on the message specified in the *uMsg* parameter.

lParam

Value depends on the message specified in the *uMsg* parameter.

pret

Address of the return value of the **SendMessage** function.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

Refer to the **Common Controls** documentation for more information on the messages that can be sent to the toolbar or status bar control.

Note to Callers Use of this call requires diligent attention, because leaving either the status bar or toolbar in an inappropriate state will affect the performance of Windows Explorer.

Note to Implementers If your Windows Explorer does not have these controls, you can return E_NOTIMPL.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellBrowser

IShellBrowser::SetMenuSB

Installs the composite menu in the view window.

```
RESULT SetMenuSB(
    HMENU hmenuShared,
    HOLEMENU holemenuReserved
);
```

Parameters

hmenuShared

Handle to the composite menu constructed by calls to **IShellBrowser::InsertMenusSB** and the Win32 **InsertMenu** function.

holemenuReserved

Reserved for future use.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

This method is similar to the **IOleInPlaceFrame::SetMenu** method. However, Windows Explorer performs menu dispatch based on the menu item identifier.

The availability of specific menu items depends on whether the view has the focus. Accordingly, it is necessary to call the **IShellBrowser::OnViewWindowActive** method whenever the view window (or one of its child windows) has the focus.

Note to Callers The object calls **IShellBrowser_SetMenuSB** to ask the container to install the composite menu structure set up by calls to **IShellBrowser::InsertMenusSB**.

Note to Implementers A container's implementation of this method should call the Windows **SetMenu** function.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellBrowser

IShellBrowser::SetStatusTextSB

Sets and displays status text about the in-place object in the container's frame-window status bar.

```
HRESULT SetStatusTextSB(
    LPCOLESTR lpszStatusText
);
```

Parameters

lpszStatusText
Address of a null-terminated character string to contain the message to display.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

It is also possible to send messages directly to the status window by using the **IShellBrowser::SendControlMsg** method.

Note to Callers Use this method to set the contents of the status bar.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellBrowser

IShellBrowser::SetToolbarItems

Adds toolbar items to Windows Explorer's toolbar.

```
HRESULT SetToolbarItems(
    LPTBBUTTON lpButtons,
    UINT nButtons,
    UINT uFlags
);
```

Parameters

lpButtons
Address of an array of **TBBUTTON** structures.

nButtons
Number of **TBBUTTON** structures in the *lpButtons* array.

uFlags
Flags specifying where the toolbar buttons should go. This parameter can be one or more of the following values:

FCT_ADDTOEND	Add at the right side of the toolbar.
FCT_CONFIGABLE	Not implemented.
FCT_MERGE	Merge the toolbar items instead of replacing all of the buttons with those provided by the view. This is the recommended choice.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellBrowser

IShellBrowser::TranslateAcceleratorSB

Translates accelerator keystrokes intended for the browser's frame while the view is active.

```
HRESULT TranslateAcceleratorSB(
    LPMSG lpmsg,
    WORD wID
);
```

Parameters

lpmsg
Address of an **MSG** structure containing the keystroke message.

wID
Command identifier value corresponding to the keystroke in the container-provided accelerator table. Containers should use this value instead of translating again.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

This method is similar to the **IOleInPlaceFrame::TranslateAccelerator** method.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellBrowser

IShellChangeNotify

The **IShellChangeNotify** is used to notify a shell namespace extension when the ID of an item has changed.

When to Implement

This interface is implemented by all namespace extensions.

When to Use

You do not call this interface directly. **IShellChangeNotify** is used by the operating system only when it has confirmed that your application is aware of this interface.

IShellChangeNotify implements **IUnknown** and one additional method:

IShellChangeNotify methods	Description
OnChange	Called when an item in the namespace extension changes.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellChangeNotify::OnChange

Called to inform a namespace extension that an event has taken place that affects its items.

```
HRESULT OnChange(
    LONG lEvent,
    LPCITEMIDLIST pidl1,
    LPCITEMIDLIST pidl2,
);
```

Parameters

lEvent

Value that describes the event that has occurred. Typically, only one event is specified at a time. If more than one event is specified, the values contained in the *pidl1* and *pidl2* parameters must be the same, respectively, for all specified events. The *lEvent* parameter may contain one or more be one or more of the following flags:

SHCNE_ALLEVENTS	All events have occurred.
SHCNE_ASSOCCHANGED	A file type association has changed. *pidl1* and *pidl2* are not used and must be NULL.
SHCNE_ATTRIBUTES	The attributes of an item or folder have changed. *pidl1* contains the item or folder that has changed. *pidl2* is not used and should be NULL.
SHCNE_CREATE	A nonfolder item has been created. *pidl1* contains the item that was created. *pidl2* is not used and should be NULL.
SHCNE_DELETE	A nonfolder item has been deleted. *pidl1* contains the item that was deleted. *pidl2* is not used and should be NULL.
SHCNE_DRIVEADD	A drive has been added. *pidl1* contains the root of the drive that was added. *pidl2* is not used and should be NULL.
SHCNE_DRIVEADDGUI	A drive has been added and the shell should create a new window for the drive. *pidl1* contains the root of the drive that was added. *pidl2* is not used and should be NULL.
SHCNE_DRIVEREMOVED	A drive has been removed. *pidl1* contains the root of the drive that was removed. *pidl2* is not used and should be NULL.
SHCNE_FREESPACE	The amount of free space on a drive has changed. *pidl1* contains the root of the drive on which the free space changed. *pidl2* is not used and should be NULL.
SHCNE_MEDIAINSERTED	Storage media has been inserted into a drive. *pidl1* contains the root of the drive that contains the new media. *pidl2* is not used and should be NULL.
SHCNE_MEDIAREMOVED	Storage media has been removed from a drive. *pidl1* contains the root of the drive from which the media was removed. *pidl2* is not used and should be NULL.
SHCNE_MKDIR	A folder has been created. *pidl1* contains the folder that was created. *pidl2* is not used and should be NULL.
SHCNE_NETSHARE	A folder on the local computer is being shared through the network. *pidl1* contains the folder that is being shared. *pidl2* is not used and should be NULL.
SHCNE_NETUNSHARE	A folder on the local computer is no longer being shared through the network. *pidl1* contains the folder that is no longer being shared. *pidl2* is not used and should be NULL.
SHCNE_RENAMEFOLDER	The name of a folder has changed. *pidl1* contains the previous PIDL or name of the folder. *pidl2* contains the new PIDL or name of the folder.
SHCNE_RENAMEITEM	The name of a nonfolder item has changed. *pidl1* contains the previous PIDL or name of the item. *pidl2* contains the new PIDL or name of the item.

(continued)

(continued)

SHCNE_RMDIR
A folder has been removed. *pidl1* contains the folder that was removed. *pidl2* is not used and should be NULL.

SHCNE_SERVERDISCONNECT
The computer has disconnected from a server. *pidl1* contains the server from which the computer was disconnected. *pidl2* is not used and should be NULL.

SHCNE_UPDATEDIR
The contents of an existing folder have changed, but the folder still exists and has not been renamed. *pidl1* contains the folder that has changed. *pidl2* is not used and should be NULL. If a folder has been created, deleted, or renamed, use SHCNE_MKDIR, SHCNE_RMDIR, or SHCNE_RENAMEFOLDER, respectively, instead.

SHCNE_UPDATEITEM
An existing nonfolder item has changed, but the item still exists and has not been renamed. *pidl1* contains the item that has changed. *pidl2* is not used and should be NULL. If a nonfolder item has been created, deleted, or renamed, use SHCNE_CREATE, SHCNE_DELETE, or SHCNE_RENAMEITEM, respectively, instead.

The following values specify combinations of other events:

SHCNE_DISKEVENTS
Specifies a combination of all of the disk event identifiers.

SHCNE_GLOBALEVENT
Specifies a combination of all of the global event identifiers.

The following value modifies other event values and cannot be used alone:

SHCNE_INTERRUPT
The specified event occurred as a result of a system interrupt.

pidl1
First event-dependent item identifier.

pidl2
Second event-dependent item identifier.

Return Values

Return S_OK if successful, or standard OLE error values otherwise.

Remarks

This method is similar in function and usage to **SHChangeNotify**.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellDetails

The **IShellDetails** interface is exposed by shell folders to provide detailed information about the items in a folder. This is the same information that is displayed by the Windows Explorer when the view of the folder is set to Details. For Windows 2000 and later systems, **IShellDetails** is superseded by **IShellFolder2**.

When to Implement

For Windows 2000 and later systems, folder objects should implement **IShellFolder2** instead of this interface. However, if your application needs to function on earlier systems, **IShellDetails** should also be exposed.

When to Use

This interface is exposed by some folder objects, including file system folders, on Windows 95, Windows NT 4.0, and later systems. However, you will need to include the version 5.0 shlobj.h header file. With Windows 2000 and later systems, use the **IShellFolder2** methods to get detailed information from shell folders.

Method	Description
IShellDetails::ColumnClick	Rearranges the column.
IShellDetails::GetDetailsOf	Retrieves specified information for an item.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellDetails::ColumnClick

Rearranges a column.

```
HRESULT ColumnClick(
  UINT iColumn,
);
```

Parameters

IColumn
The index of the column to be rearranged.

Return Values

Returns S_FALSE to tell the calling application to sort the selected column. Otherwise, returns S_OK if successful, a COM error code otherwise.

Remarks

This method is called when a client of a folder object wants to sort the object's items based on the contents of one of the Details columns. Folder objects typically return S_FALSE.

Note to Users This interface is exposed by some folder objects, including file system folders, on Windows 95, Windows NT 4.0, and later systems. However, you will need to include the version 5.0 shlobj.h header file.

Note to Implementers For Windows 2000 and later systems, folder objects should implement **IShellFolder2** instead of this interface. However, if your application needs to function on earlier systems, it should also expose **IShellDetails.**

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellDetails::GetDetailsOf

IShellDetails::GetDetailsOf

Retrieves detailed information on an item in a shell folder.

```
HRESULT GetDetailsOf(
  LPCITEMIDLIST pidl,
  UINT iColumn,
  LPSHELLDETAILS pDetails
);
```

Parameters

pidl

[in] The PIDL of the item that you are requesting information on. If this parameter is set to NULL, the title of the information field specified by *iColumn* will be returned in the **SHELLDETAILS** structure pointed to by *pDetails*.

iColumn

[in] The zero-based index of the desired information field. It is identical to column number of the information as it is displayed in a Windows Explorer Details view.

pDetails

[out] A pointer to a **SHELLDETAILS** structure with the information.

Return Values

Returns S_OK if successful. Returns E_FAIL if *iColumn* exceeds the number of columns supported by the folder. Otherwise, returns a standard COM error code.

Remarks

This method has been superseded by the **IShellFolder2** methods for shell version 5.0 and later.

The **GetDetailsOf** method provides access to the information that is displayed in the Windows Explorer Details view of a shell folder. The column numbers, column titles, and item information that you see in the Details view are identical to those returned by **GetDetailsOf**.

The available information fields and their column numbers vary depending on the particular folder. To enumerate the available fields call **GetDetailsOf** with *pidl* set to NULL for increasing values of *iColumn*. This approach provides you with the title associated with each column index. When *iColumn* exceeds the number of columns supported by the folder, **GetDetailsOf** will return E_FAIL. Bear in mind that these titles are localizable, and may not be the same for all locales.

File system folders have a large standard set of information fields. The first five fields are standard for all file system folders:

Column index	Column title
0	Name
1	Size
2	Type
3	Modified
4	Attributes

File system folders may support a number of additional fields. However, they are not required to do so and the column indexes assigned to these fields may vary.

Each virtual folder has its own unique set of information fields. Normally, the item's display name is in column zero, but the order and content of the available fields depend on the implementation of the particular folder object.

Note to Users This interface is exposed by some folder objects, including file system folders, on Windows 95, Windows NT 4.0, and later systems. However, you will need to include the version 5.0 shlobj.h header file. With Windows 2000 and later systems, use the **IShellFolder2** methods to get detailed information from shell folders.

Note to Implementers For Windows 2000 and later systems, folder objects should implement **IShellFolder2** instead of this interface. However, if your application needs to function on earlier systems, **IShellDetails** should also be exposed.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellExecuteHook

The **IShellExecuteHook** interface extends the behavior of the **ShellExecute** or **ShellExecuteEx** function. It is typically implemented by subsystems that expose the names of objects that the user can specify in the **Run** dialog box after clicking the Windows **Start** button.

When to Implement

You should implement **IShellExecuteHook** when you have named objects that the user would expect to be able to run from the **Run** dialog box after clicking the Windows **Start** button.

When to Use

You do not use this interface directly. It is generally used by the **ShellExecuteEx** function's code.

IShellExecuteHook method	Description
Execute	Called when **ShellExecute** or **ShellExecuteEx** is called for an object that is registered.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellExecuteHook::Execute

Called when **ShellExecute** or **ShellExecuteEx** is called for an object that is registered.

```
HRESULT Execute(
    LPSHELLEXECUTEINFO pei
);
```

Parameters

pei

Address of a **SHELLEXECUTEINFO** structure that contains information about the object being executed. On successful completion of the hook, the **hInstApp** member will be filled in by the hook.

Return Values

Returns one of the following values or an OLE-defined error value:

S_OK	The hook processed the execution. **ShellExecute** or **ShellExecuteEx** should not perform any other processing.
S_FALSE	The hook is installed but did not process the execution. **ShellExecute** or **ShellExecuteEx** should perform the default processing.

Remarks

This method provides an opportunity for the application to hook the execution of an object and change the default execution or perform some other action before the object is executed.

If the item should not be executed, **Execute** can simply return S_OK. To prevent the shell from generating an error message box, set the **hInstApp** member of **SHELLEXECUTE** to 32 or greater.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellExecuteHook

IShellExtInit

The **IShellExtInit** interface is used to initialize shell extensions for property sheets, context menus, and drag-and-drop handlers (extensions that add items to context menus during nondefault drag-and-drop operations).

When to Implement

Implement **IShellExtInit** when you are writing a handler based on the **IContextMenu** or **IShellPropSheetExt** interface.

Note that shell extensions based on other interfaces do not use this method of initialization.

When to Use

You do not use this interface directly. The shell calls it to initialize the handler.

IShellExtInit method	Description
Initialize	Initializes the shell extension.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellExtInit::Initialize

Initializes a property sheet extension, context menu extension, or drag-and-drop handler.

```
HRESULT Initialize(
    LPCITEMIDLIST pidlFolder,
    LPDATAOBJECT lpdobj,
    HKEY hkeyProgID
);
```

Parameters

pidlFolder

Address of an **ITEMIDLIST** structure that uniquely identifies a folder. For property sheet extensions, this parameter is NULL. For context menu extensions, it is the item identifier list for the folder that contains the item whose context menu is being displayed. For nondefault drag-and-drop menu extensions, this parameter specifies the target folder.

lpdobj

Address of an **IDataObject** interface object that can be used to retrieve the objects being acted upon.

hkeyProgID

Registry key for the file object or folder type.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

The meanings of some parameters depend on the extension type. For drag-and-drop handlers, the *pidlFolder* parameter specifies the destination folder (the drop target), the *lpdobj* parameter identifies the items being dropped, and the *hkeyProgID* parameter specifies the file class of the destination folder.

For context menu extensions, *pidlFolder* specifies the folder that contains the selected file objects, *lpdobj* identifies the selected file objects, and *hkeyProgID* specifies the file class of the file object that has the focus.

For property sheet extensions, *pidlFolder* is NULL, *lpdobj* identifies the selected file objects, and *hkeyProgID* specifies the file class of the file object that has the focus.

Notes to Implementers This is the first method that the shell calls after it creates an instance of a property sheet extension, context menu extension, or drag-and-drop handler.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellExtInit

IShellFolder

The **IShellFolder** interface is used to manage folders. It is exposed by all shell namespace folder objects.

When to Implement

Implement this interface for objects that extend the shell's namespace. For example, implement this interface to create a separate namespace that requires a rooted Windows Explorer or to install a new namespace directly within the hierarchy of the system namespace. You are most familiar with the contents of your namespace, so you are responsible for implementing everything needed to access your data.

When to Use

Use this interface when you need to display or perform an operation on the contents of the shell's namespace. Objects that support **IShellFolder** are usually created by other shell folder objects. To get a folder's **IShellFolder** interface, you normally start by calling **SHGetDesktopFolder**. This function returns a pointer to the desktop's **IShellFolder** interface. You can then use its methods to get an **IShellFolder** interface for a particular namespace folder.

Note Some **IShellFolder** methods, such as **IShellFolder::BindToObject**, only accept PIDLs that are relative to the parent folder. They must contain only a single **SHITEMID** structure plus the terminating NULL. When you enumerate the contents of a folder with **IEnumIDList**, you will receive PIDL of this form. Other methods, such as **IShellFolder::ParseDisplayName**, accept PIDLs with more than one **SHITEMID** structure that identify objects one or more levels below the parent folder. Check the reference to be sure what type of PIDL is expected by a particular method.

IShellFolder methods	Description
BindToObject	Retrieves the IShellFolder interface for the specified subfolder.
BindToStorage	Not currently implemented.
CompareIDs	Determines the relative order of two file objects or folders, given their item identifier lists.
CreateViewObject	Creates a view object of the folder itself.
EnumObjects	Enumerates the objects in a folder.
GetAttributesOf	Retrieves the attributes of the specified file object or subfolder.
GetDisplayNameOf	Retrieves the display name of a file object or subfolder.
GetUIObjectOf	Creates an OLE interface that can be used to carry out operations on a file object or subfolder.
ParseDisplayName	Translates a display name into an item identifier list.
SetNameOf	Sets the display name of the specified file object or subfolder and changes its identifier accordingly.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellFolder::BindToObject

Retrieves an **IShellFolder** object for a subfolder.

```
HRESULT BindToObject(
    LPCITEMIDLIST pidl,
    LPBC pbc,
    REFIID riid,
    LPVOID *ppvOut
);
```

Parameters

pidl

[in] Address of an **ITEMIDLIST** structure that identifies the subfolder relative to its parent folder. The structure must contain exactly one **SHITEMID** structure followed by a terminating zero.

pbc

[in] Optional address of an **IBindCtx** interface on a bind context object to be used during this operation. If this parameter is not used, set it to NULL. Because support for *pbc* is optional for folder object implementations, some folders may not support the use of bind contexts.

riid

[in] Identifier of the interface to return.

ppvOut

[out] Address that receives the interface pointer. If an error occurs, a NULL pointer is returned in this address.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

Use this method to get a pointer to the **IShellFolder** interface of a subfolder.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

SHGetDesktopFolder

IShellFolder::BindToStorage

Requests a pointer to an object's storage interface.

```
HRESULT BindToStorage(
    LPCITEMIDLIST pidl,
    LPBC pbc,
    REFIID riid,
    VOID **ppv
);
```

Parameters

pidl

[in] Address of an **ITEMIDLIST** structure that identifies the subfolder relative to its parent folder. The structure must contain exactly one **SHITEMID** structure followed by a terminating zero.

pbc

[in] Optional address of an **IBindCtx** interface on a bind context object to be used during this operation. If this parameter is not used, set it to NULL. Because support for *pbc* is optional for folder object implementations, some folders may not support the use of bind contexts.

riid

[in] IID of the requested storage interface. To retrieve an **IStream**, **IStorage**, or **IPropertySetStorage** interface pointer, set *riid* to IID_IStream, IID_IStorage, or IID_IPropertySetStorage, respectively.

ppvOut

[out] Address that receives the interface pointer specified by *riid*. If an error occurs, a NULL pointer is returned in this address.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

Remarks

Name space extensions have the option of allowing applications to bind to an object that represents an item's storage. If this option is supported, **BindToStorage** returns a specified interface pointer that can then be used to access the contents of object. See the **IMoniker::BindToStorage** reference for further discussion.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellFolder::CompareIDs

Determines the relative order of two file objects or folders, given their item identifier lists.

```
HRESULT CompareIDs(
    LPARAM lParam,
    LPCITEMIDLIST pidl1,
    LPCITEMIDLIST pidl2
);
```

Parameters

lParam

[in] Value that specifies how the comparison should be performed.

The lower sixteen bits of *lParam* define the sorting rule. Most applications set the sorting rule to the default value of zero, indicating that the two items should be compared by name. The system does not define any other sorting rules. Some folder objects might allow calling applications to use the lower sixteen bits of *lParam* to specify folder-specific sorting rules. The rules and their associated *lParam* values are defined by the folder.

The upper sixteen bits of *lParam* are used for flags that modify the sorting rule. The system currently defines one modifier flag:

SCHIDS_ALLFIELDS

Version 5.0. Compare all the information contained in the **ITEMIDLIST** structure, not just the display names. This flag is valid only for folder objects that support the **IShellFolder2** interface. For instance, if the two items are files, the folder should compare their names, sizes, file times, attributes, and any other information in the structures. If this flag is set, the lower sixteen bits of *lParam* must be zero.

pidl1

[in] Address of the first item's **ITEMIDLIST** structure. It will be relative to the folder. This **ITEMIDLIST** structure can contain more than one element; therefore, the entire structure must be compared, not just the first element.

pidl2

[in] Address of the second item's **ITEMIDLIST** structure. It will be relative to the folder. This **ITEMIDLIST** structure can contain more than one element; therefore, the entire structure must be compared, not just the first element.

Return Values

If this method is successful, the CODE field of the status code (SCODE) contains one of the following values:

Less than zero	The first item should precede the second (*pidl1* < *pidl2*).
Greater than zero	The first item should follow the second (*pidl1* > *pidl2*).
Zero	The two items are the same (*pidl1* = *pidl2*).

If this method is unsuccessful, it returns an OLE error code.

Remarks

Note to Callers Do not set the SHCIDS_ALLFIELDS flag in *lParam* if the folder object does not support **IShellFolder2**. Doing so might have unpredictable results. If you use the SHCIDS_ALLFIELDS flag, the lower sixteen bits of *lParam* must be set to zero.

Note to Implementers To extract the sorting rule, use a bitwise OR operator (&) to combine *lParam* with SHCIDS_COLUMNMASK (0X0000FFFF). This operation masks off the upper sixteen bits of *lParam*, including the SHCIDS_ALLFIELDS value.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellFolder

IShellFolder::CreateViewObject

Creates a view object of a folder.

```
HRESULT CreateViewObject(
    HWND hwndOwner,
    REFIID riid,
    LPVOID *ppvOut
);
```

Parameters

hwndOwner
[in] Handle to the owner window from which to create the view object.

riid
[in] Identifier of the interface to return.

ppvOut
[out] Address of a pointer to the view object.

Return Values

Returns NOERROR if successful, or an OLE defined error value otherwise.

Remarks

It is important to remember that the COM object created by **CreateViewObject** must be a different object than the shell folder object. Windows Explorer may call this method more than once to create multiple view objects, and each of these must behave as independent objects. A new view object must be created for each call.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellFolder

IShellFolder::EnumObjects

Allows a client to determine the contents of a folder by creating an item identifier enumeration object and returning its **IEnumIDList** interface. The methods supported by that interface can then be used enumerate the folder's contents.

```
HRESULT EnumObjects(
  HWND hwndOwner,
  DWORD grfFlags,
  LPENUMIDLIST *ppenumIDList
);
```

Parameters

hwndOwner

[in] If user input is required to perform the enumeration, this window handle should be used by the enumeration object as the parent window to take user input. An example would be a dialog box to ask for a password or prompt the user to insert a CD or floppy disk. If *hwndOwner* is set to NULL, the enumerator should not post any messages, and if user input is required, it should silently fail.

grfFlags

[in] Flags indicating which items to include in the enumeration. For a list of possible values, see the **SHCONTF** enumerated type.

ppenumIDList

[out] Address that receives a pointer to the **IEnumIDList** interface of the enumeration object created by this method. If an error occurs, *ppenumIDList* is set to NULL.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

The calling application must free the returned **IEnumIDList** object by calling its **Release** method.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellFolder, **SHGetDesktopFolder**

IShellFolder::GetAttributesOf

Retrieves the attributes of one or more file objects or subfolders.

```
HRESULT GetAttributesOf(
    UINT cidl,
    LPCITEMIDLIST *apidl,
    ULONG *rgfInOut
);
```

Parameters

cidl

[in] Number of file objects from which to retrieve attributes.

apidl

[in] Address of an array of pointers to **ITEMIDLIST** structures, each of which uniquely identifies a file object relative to the parent folder. Each **ITEMIDLIST** structure must contain exactly one **SHITEMID** structure followed by a terminating zero.

rgflnOut

[in/out] Address of a single ULONG value that, on entry, contains the attributes that the caller is requesting. On exit, this value contains the requested attributes that are common to all of the specified objects. Note that this is the address of a single ULONG value, not an array of ULONG values. The lists below describe the possible flags for this parameter.

A file object's capability flags may be zero or a combination of the following values:

Value	Description
SFGAO_CANCOPY	The specified file objects or folders can be copied (same value as the DROPEFFECT_COPY flag).
SFGAO_CANDELETE	The specified file objects or folders can be deleted.
SFGAO_CANLINK	Shortcuts can be created for the specified file objects or folders. This flag has the same value as **DROPEFFECT_LINK**. The normal use of this flag is to add a "Create Shortcut" item to the context menu that is displayed during drag-drop operations. However, SFGAO_CANLINK also adds a "Create Shortcut" item to the Windows Explorer's File menu, and to normal context menus. If this item is selected, your application's **IContextMenu::InvokeCommand** will be invoked with the lpVerb member of the **CMINVOKECOMMANDINFO** structure set to "link". Your application is responsible for creating the link.
SFGAO_CANMONIKER	It is possible to create monikers for the specified file objects or folders.
SFGAO_CANMOVE	The specified file objects or folders can be moved (same value as the DROPEFFECT_MOVE flag).

(continued)

(continued)

Value	Description
SFGAO_CANRENAME	The specified file objects or folders can be renamed. Note that this flag is essentially a suggestion. It does not guarantee that a namespace client will rename the file or folder object.
SFGAO_CAPABILITYMASK	This flag is a mask for the capability flags.
SFGAO_DROPTARGET	The specified file objects or folders are drop targets.
SFGAO_HASPROPSHEET	The specified file objects or folders have property sheets.

A file object's display attributes may be zero or a combination of the following values:

Value	Description
SFGAO_DISPLAYATTRMASK	This flag is a mask for the display attributes.
SFGAO_GHOSTED	The specified file objects or folders should be displayed using a ghosted icon.
SFGAO_LINK	The specified file objects are shortcuts.
SFGAO_READONLY	The specified file objects or folders are read-only.
SFGAO_SHARE	The specified folders are shared.

A file object's contents flags may be zero or a combination of the following values:

Value	Description
SFGAO_CONTENTSMASK	This flag is a mask for the contents attributes.
SFGAO_HASSUBFOLDER	The specified folders may have subfolders, and are, therefore expandable in the left pane of Windows Explorer.

Note The SFGAO_HASSUBFOLDER attribute is only advisory, and may be returned by shell folder implementations even if they do not contain subfolders. Returning SFGAO_HASSUBFOLDER is recommended whenever a significant amount of time is required to determine whether or not any subfolders exist. For example, the shell always returns SFGAO_HASSUBFOLDER when a folder is located on a network drive.

A file object's miscellaneous attributes may be zero or a combination of the following values:

Value	Description
SFGAO_BROWSABLE	The specified items can be browsed in place.
SFGAO_COMPRESSED	The specified items are compressed.

SFGAO_FILESYSTEM	The specified folders or file objects are part of the file system (that is, they are files, directories, or root directories).
SFGAO_FILESYSANCESTOR	The specified folders contain one or more file system folders.
SFGAO_FOLDER	The specified items are folders.
SFGAO_NEWCONTENT	The objects contain new content.
SFGAO_NONENUMERATED	The items are nonenumerated items.
SFGAO_REMOVABLE	The specified file objects or folders are on removable media.
SFGAO_VALIDATE	Validate cached information. The shell will validate that the objects specified in *apidl* still exist and will not use cached information when retrieving the attributes. If one or more of the items specified in *apidl* no longer exist, this method will return an error code. If *cidl* is zero, the shell will discard all cached information for the shell folder. This is similar to doing a refresh of the folder.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

You can optimize this operation by not returning unspecified flags.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellFolder

IShellFolder::GetDisplayNameOf

Retrieves the display name for the specified file object or subfolder.

```
HRESULT GetDisplayNameOf(
    LPCITEMIDLIST pidl,
    DWORD uFlags,
    LPSTRRET lpName
);
```

Parameters

pidl

[in] Address of an **ITEMIDLIST** structure that uniquely identifies the file object or subfolder relative to the parent folder. The structure must contain exactly one **SHITEMID** structure followed by a terminating zero.

uFlags

[in] Flags used to request the type of display name to return. For a list of possible values, see the **SHGNO** enumerated type.

lpName

[out] Pointer to a **STRRET** structure in which to return the display name. The type of name returned in this structure may be the requested type, but the shell folder may return a different type.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

The simplest way to retrieve the display name from the structure pointed to by *lpName* is to pass it to either **StrRetToBuf** or **StrRetToStr**. These functions take a **STRRET** structure and return the name. You can also by examine the structure's **uType** member and get the name from the appropriate member.

The flags specified in *uFlags* are effectively hints about the intended use of the name. They do not guarantee that **IShellFolder** will return the requested form of the name. If that form is not available, a different one may be returned. In particular, there is no guarantee that the name returned by the SHGDN_FORPARSING flag will be successfully parsed by **ParseDisplayName**. There are also some combinations of flags that may cause the **GetDisplayName/ParseDisplayName** roundtrip to not return the original identifier list. This occurrence is exceptional, but you should check to be sure.

Note The parsing name that is returned when *uFlags* has the SHGDN_FORPARSING flag set is not necessarily a normal text string. Virtual folders such as My Computer may return a string containing the folder object's GUID in the form "::{GUID}".

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellFolder

IShellFolder::GetUIObjectOf

Retrieves an OLE interface that can be used to carry out actions on the specified file objects or folders.

```
HRESULT GetUIObjectOf(
    HWND hwndOwner,
    UINT cidl,
    LPCITEMIDLIST *apidl,
    REFIID riid,
    UINT *prgfInOut,
    LPVOID *ppvOut
);
```

Parameters

hwndOwner

[in] Handle to the owner window that the client should specify if it displays a dialog box or message box.

cidl

[in] Number of file objects or subfolders specified in the *apidl* parameter.

apidl

[in] Address of an array of pointers to **ITEMIDLIST** structures, each of which uniquely identifies a file object or subfolder relative to the parent folder. Each item identifier list must contain exactly one **SHITEMID** structure followed by a terminating zero.

riid

[in] Identifier of the COM interface object to return. This can be any valid interface identifier that can be created for an item. The most common identifiers used by the shell are listed in the comments at the end of this reference.

prgflnOut

Reserved.

ppvOut

[out] Pointer to the requested interface. If an error occurs, a NULL pointer is returned in this address.

Return Values

Returns NOERROR if successful, E_NOINTERFACE if the interface is not supported, or an OLE-defined error value otherwise.

Remarks

If *cidl* is greater than one, the **GetUIObjectOf** implementation should only succeed if it can create one object for all items specified in *apidl*. If the implementation cannot create one object for all items, this method should fail.

The following are the most common interface identifiers the shell uses when requesting an interface from this method. The list also indicates if *cidl* can be greater than one for the requested interface.

Interface identifier	Allowed *cidl* value
IContextMenu	The *cidl* parameter can be greater than or equal to one.
IContextMenu2	The *cidl* parameter can be greater than or equal to one.
IDataObject	The *cidl* parameter can be greater than or equal to one.
IDropTarget	The *cidl* parameter can only be one.
IExtractIcon	The *cidl* parameter can only be one.
IQueryInfo	The *cidl* parameter can only be one.

Requirements

Version 4.00 and later of Shell32.dll.
Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellFolder

IShellFolder::ParseDisplayName

Translates a file object's or folder's display name into an item identifier list.

```
HRESULT ParseDisplayName(
    HWND hwndOwner,
    LPBC pbc,
    LPOLESTR lpwszDisplayName,
    ULONG *pchEaten,
    LPITEMIDLIST *ppidl,
    ULONG *pdwAttributes
);
```

Parameters

hwndOwner

[in] Optional window handle. The client should provide a window handle if it displays a dialog or message box. Otherwise set *hwndOwner* to NULL.

pbc

[in] Optional bind context that controls the parsing operation. This parameter is set normally to NULL.

lpwszDisplayName

[in] Null-terminated UNICODE string with the display name. Because each shell folder defines its own parsing syntax, the form this string can take may vary. The desktop folder, for instance, accepts paths such as "c:\My Docs\My File.txt". It also will accept references to items in the namespace that have a GUID associated with them using the "::{GUID}" syntax. For example, to retrieve a fully qualified identifier list for the control panel from the desktop folder, you can use:

```
"{CLSID for My Computer}\::{CLSID for the Control Panel}"
```

pchEaten

[out] Pointer to a ULONG value that receives the number of characters of the display name that was parsed. If your application does not need this information, set *pchEaten* to NULL, and no value will be returned.

ppidl

[out] Pointer to an **ITEMIDLIST** pointer that receives the item identifier list for the object. The returned item identifier list specifies the item relative to the parsing folder. If the object associated with *lpwszDisplayName* is within the parsing folder, the returned item identifier list will contain only one **SHITEMID** structure. If the object is in a subfolder of the parsing folder, the returned item identifier list will contain multiple **SHITEMID** structures. If an error occurs, NULL is returned in this address.

pdwAttributes

[in/out] Optional parameter that can be used to query for file attributes. If not used, it should be set to NULL. To query for one or more attributes, initialize the *pdwAttributes* with the flags that represent the attributes of interest. On return, those attributes that are true *and* were requested will be set. A file object's attribute flags may be zero or a combination of the following values:

Value	Description
SFGAO_CANCOPY	The specified file objects or folders can be copied (same value as the DROPEFFECT_COPY flag).
SFGAO_CANDELETE	The specified file objects or folders can be deleted.
SFGAO_CANLINK	Shortcuts can be created for the specified file objects or folders. This flag has the same value as **DROPEFFECT_LINK**. The normal use of this flag is to add a "Create Shortcut" item to the context menu that is displayed during drag-drop operations. However, SFGAO_CANLINK also adds a "Create

(continued)

(continued)

Value	Description
	Shortcut" item to the Windows Explorer's File menu, and to normal context menus. If this item is selected, your application's **IContextMenu::InvokeCommand** will be invoked with the lpVerb member of the **CMINVOKECOMMANDINFO** structure set to "link". Your application is responsible for creating the link.
SFGAO_CANMONIKER	It is possible to create monikers for the specified file objects or folders.
SFGAO_CANMOVE	The specified file objects or folders can be moved (same value as the DROPEFFECT_MOVE flag).
SFGAO_CANRENAME	The specified file objects or folders can be renamed. Note that this flag is essentially a suggestion. It does not guarantee that a namespace client will rename the file or folder object.
SFGAO_CAPABILITYMASK	This flag is a mask for the capability flags.
SFGAO_DROPTARGET	The specified file objects or folders are drop targets.
SFGAO_HASPROPSHEET	The specified file objects or folders have property sheets.

A file object's display attributes may be zero or a combination of the following values:

Value	Description
SFGAO_DISPLAYATTRMASK	This flag is a mask for the display attributes.
SFGAO_GHOSTED	The specified file objects or folders should be displayed using a ghosted icon.
SFGAO_LINK	The specified file objects are shortcuts.
SFGAO_READONLY	The specified file objects or folders are read-only.
SFGAO_SHARE	The specified folders are shared.

A file object's contents flags may be zero or a combination of the following values:

Value	Description
SFGAO_CONTENTSMASK	This flag is a mask for the contents attributes.
SFGAO_HASSUBFOLDER	The specified folders have subfolders (and are, therefore, expandable in the left pane of Windows Explorer).

A file object's miscellaneous attributes may be zero or a combination of the following values:

Value	Description
SFGAO_BROWSABLE	The specified items can be browsed in place.
SFGAO_COMPRESSED	The specified items are compressed.
SFGAO_FILESYSTEM	The specified folders or file objects are part of the file system (that is, they are files, directories, or root directories).
SFGAO_FILESYSANCESTOR	The specified folders contain one or more file system folders.
SFGAO_FOLDER	The specified items are folders.
SFGAO_NEWCONTENT	The objects contain new content.
SFGAO_NONENUMERATED	The items are nonenumerated items.
SFGAO_REMOVABLE	The specified file objects or folders are on removable media.
SFGAO_VALIDATE	This flag is used to confirm the existence of the folder or object that corresponds to *lpwszDisplayName*. Not every folder implements this flag, but if it is passed in to ParseDisplayName, and the folder or object does not exist, a failure code should be returned.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

Some shell folders may not implement **ParseDisplayName**. Each folder that does will define its own parsing syntax.

Querying for some attributes may be relatively slow and use significant amounts of memory. For example, to determine if a file is shared, the shell will load network components. This procedure may require the loading of several DLLs. The purpose of *dwAttributes* is to allow you to restrict the query to only that information that is needed. The following code fragment illustrates how to find out if a file is compressed:

```
IShellFolder *psf;
DWORD dwAttribs = SFGAO_COMPRESSED;
..
hres = psf->ParseDisplayName(NULL,
            NULL,
            lpwszDisplayName,
            &cbEaten,
            &pidl,
            &dwAttribs);
if(dwAttribs & SFGAO_COMPRESSED)
```

```
{
//do something with the compressed file
}
```

Since *pdwAttributes* is an in/out parameter, it should always be initialized. If you pass in an uninitialized value, some of the bits may be set. **ParseDisplayName** will then query for the corresponding attributes, which may lead to undesirable delays or memory demands. If you do not wish to query for attributes, set *pdwAttributes* to NULL to avoid unpredictable behavior.

This method is similar to the OLE **IParseDisplayName::ParseDisplayName** method.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellFolder, **IShellLink**, **IShellFolder::GetAttributesOf**

IShellFolder::SetNameOf

Sets the display name of a file object or subfolder, changing the item identifier in the process.

```
HRESULT SetNameOf(
    HWND hwndOwner,
    LPCITEMIDLIST pidl,
    LPCOLESTR lpszName,
    DWORD uFlags,
    LPITEMIDLIST *ppidlOut
);
```

Parameters

hwndOwner

[in] Handle to the owner window of any dialog or message boxes that the client displays.

pidl

[in] Address of an **ITEMIDLIST** structure that uniquely identifies the file object or subfolder relative to the parent folder. The structure must contain exactly one **SHITEMID** structure followed by a terminating zero.

lpszName
[in] Address of a null-terminated string that specifies the new display name.

uFlags
[in] Flags indicating the type of name specified by the *lpszName* parameter. For a list of possible values, see the description of the **SHGNO** enumerated type.

ppidlOut
[in/out] Address of a pointer to the new **ITEMIDLIST** structure. Note that there is no guarantee that this parameter will point to a valid **ITEMIDLIST**. Some implementations of **SetNameOf** may ignore the request. If you call **SetNameOf** with *ppidlOut* set to NULL, it will not return a new **ITEMIDLIST** for the object. If an error occurs, *ppidlOut* will point to NULL.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

Changing the display name of a file system object, or a folder within it, renames the file or directory.

Before calling this method, applications should call **IShellFolder::GetAttributesOf** and check that the SFGAO_CANRENAME flag is set. Note that this flag is essentially a hint to name space clients. It does not necessarily imply that **IShellFolder::SetNameOf** will succeed or fail.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellFolder

IShellFolder2

The **IShellFolder2** interface extends the capabilities of **IShellFolder**. Its methods provide a variety of information about the contents of a shell folder.

When to Implement

Implement **IShellFolder2** if your namespace extension provides services to clients beyond those in **IShellFolder**.

When to Use

Call **IShellFolder2** when you need detailed information on items contained by a shell folder. This interface supersedes **IShellDetails**.

IShellFolder2 implements all the **IShellFolder** methods as well as **IUnknown**. The following methods are specific to **IShellFolder2**.

Methods	Description
EnumSearches	Requests a pointer to an interface that allows a client to enumerate the available search objects.
GetDefaultColumn	Gets the default sorting and display columns.
GetDefaultColumnState	Gets the default state for a specified column.
GetDefaultSearchGUID	Requests the GUID of the default search object for the folder.
GetDetailsEx	Retrieves detailed information, identified by a property set ID (FMTID) and property ID (PID), on an item in a shell folder.
GetDetailsOf	Retrieves detailed information, identified by a column index, on an item in a shell folder.
MapNameToSCID	Converts a column name to the appropriate property set ID (FMTID) and property ID (PID).

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellFolder2::EnumSearches

Requests a pointer to an interface that allows a client to enumerate the available search objects.

```
HRESULT EnumSearches(
    IEnumExtraSearch **ppEnum
);
```

Parameters

ppEnum

Address of a pointer to an enumerator object's **IEnumExtraSearch** interface.

Return Values

Returns NOERROR if successful, or an OLE error code otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellFolder2::GetDefaultColumn

Gets the default sorting and display columns.

```
HRESULT GetDefaultColumn(
    DWORD dwReserved,
    ULONG *pSort,
    ULONG *pDisplay
);
```

Parameters

dwReserved
[in] Reserved. Set to zero.

pSort
[out] Pointer to the index of the default sorted column. This column is the one that should be used for sorting the items in the folder. To determine the sorting order of any pair of items, pass their PIDLs to **IShellFolder::CompareIDs**.

pDisplay
[out] Pointer to the index of the default display column. If a view will display only one string to represent an item, it should be taken from this column. Pass this index and the item's PIDL to **IShellFolder2::GetDetailsOf** to retrieve the string.

Return Values

Returns NOERROR if successful, or a COM error code otherwise.

Remarks

Both column indexes returned by this method are intended for use by an application that is presenting a view of this folder.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellFolder2::GetDefaultColumnState

Gets the default state for a specified column.

```
HRESULT GetDefaultColumnState(
    UINT iColumn
    DWORD *pcsFlags
);
```

Parameters

iColumn

[in] Column number.

pcsFlags

[out] Pointer to a value containing flags that indicate the default column state. It can be a combination of the following flags:

Flag	Description
SHCOLSTATE_TYPE_STR	A string.
SHCOLSTATE_TYPE_INT	An integer.
SHCOLSTATE_TYPE_DATE	A date.
SHCOLSTATE_ONBYDEFAULT	Should be shown by default in the Windows Explorer Details view.
SHCOLSTATE_SLOW	Will be slow to compute. You should do the computation on a background thread.
SHCOLSTATE_EXTENDED	Provided by a handler, not the folder object.
SHCOLSTATE_SECONDARYUI	Not displayed in the context menu, but listed in the More. dialog box.
SHCOLSTATE_HIDDEN	Not displayed in the user interface.

Return Values

Returns NOERROR if successful, or an OLE error code otherwise.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellFolder2::GetDefaultSearchGUID

Returns the GUID of the default search object for the folder.

```
HRESULT GetDefaultSearchGUID(
    LPGUID lpGUID
);
```

Parameters

lpGUID

[out] GUID of the default search object.

Return Values

Returns NOERROR if successful, or an OLE error code otherwise.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellFolder2::GetDetailsEx

Retrieves detailed information, identified by a property set ID (FMTID) and property ID (PID), on an item in a shell folder.

```
HRESULT GetDetailsEx(
    LPCITEMIDLIST pidl,
    SHCOLUMNID *pscid,
    VARIANT *pv
);
```

Parameters

pidl

[in] Item's PIDL.

pscid

[in] Pointer to an **SHCOLUMNID** structure that identifies the column.

pv

[out] Pointer to a VARIANT with the requested information. The value will be fully typed.

Return Values

Returns NOERROR if successful, or a COM error code otherwise.

Remarks

This function is a more robust version of **IShellFolder2::GetDetailsOf**. It provides access to the information that is displayed in the Windows Explorer Details view of a shell folder. The primary difference is that **GetDetailsEx** allows you to identify the column with an **FMTID** and **PID** instead of having to first determine the column index.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellFolder2::GetDetailsOf

Retrieves detailed information, identified by a column index, on an item in a shell folder.

```
HRESULT GetDetailsOf(
  LPCITEMIDLIST pidl,
  UINT iColumn,
  LPSHELLDETAILS pDetails
);
```

Parameters

pidl

[in] PIDL of the item for which you are requesting information. If this parameter is set to NULL, the title of the information field specified by *iColumn* is returned.

iColumn

[in] Zero-based index of the desired information field. It is identical to the column number of the information as it is displayed in a Windows Explorer Details view.

pDetails

[out] Pointer to a **SHELLDETAILS** structure with the information.

Return Values

Returns S_OK if successful, or a standard COM error code otherwise.

Remarks

The **GetDetailsOf** method is identical to **IShellDetails::GetDetailsOf**. For a more robust way to get item information that doesn't require you to know the column index, use **GetDetailsEx**.

The **GetDetailsOf** method provides access to the information that is displayed in the Windows Explorer Details view of a shell folder. The column numbers, headings, and information that you see in the Windows Explorer Details view are identical to those of **GetDetailsOf**. Note that the available information fields and their column numbers vary depending on the particular folder. You can enumerate the available fields by calling this method with *pidl* set to NULL, and examining the title associated with each column index. Bear in mind that these titles are localizable and might not be the same for all locales.

File system folders have a large, standard set of information fields. The first five fields are standard for all file system folders.

Column index	Column title
0	Name
1	Size
2	Type
3	Modified
4	Attributes

File system folders can support a number of additional fields. However, they are not required to do so, and the column indexes assigned to these fields might vary.

Each virtual folder has its own unique set of information fields. Normally, the item's display name is in column zero, but the order and content of the remaining fields depend on the implementation of the particular folder object.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellFolder2::MapNameToSCID

Converts a column name to the appropriate property set ID (FMTID) and property ID (PID).

```
HRESULT MapNameToSCID(
    LPCWSTR pwszName,
    SHCOLUMNID *pscid
);
```

Parameters

pwszName

[in] Pointer to a NULL-terminated Unicode string with the column's name.

pscid

[out] Pointer to an SHCOLUMNID structure containing the FMTID and PID.

Return Values

Returns NOERROR if successful, or a COM error code otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellIcon

The **IShellIcon** interface is used to obtain an icon index for an **IShellFolder** object. **IShellIcon** allows an application to obtain the icon for any object within a folder by using only one instance of the interface. **IExtractIcon**, on the other hand, requires that a separate instance of the interface be created for each object.

When to Implement

Implement **IShellIcon** when creating an **IShellFolder** implementation to provide a quick way to obtain the icon for an object in the folder.

If **IShellIcon** is not implemented by an **IShellFolder** object, **IShellFolder::GetUIObjectOf** is used to get an icon for all objects.

When to Use

Use **IShellIcon** when retrieving the icon index for an item in a shell folder.

IShellIcon method	Description
GetIconOf	Retrieves an icon for an object in a folder.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellIcon::GetIconOf

Retrieves an icon for an object inside a specific folder.

```
HRESULT GetIconOf(
    LPCITEMIDLIST pidl,
    UINT flags,
    LPINT lpIconIndex
);
```

Parameters

pidl

Address of the **ITEMIDLIST** structure that specifies the relative location of the folder.

flags

Flags specifying how the icon is to display. This parameter can be zero or one of the following values:

GIL_FORSHELL	The icon is to be displayed in a shell folder.
GIL_OPENICON	The icon should be in the open state if both open- and closed-state images are available. If this flag is not specified, the icon should be in the normal or closed state. This flag is typically used for folder objects.

lpIconIndex

Address of the index of the icon in the system image list. The following standard image list indexes can be returned:

0	Document (blank page, not associated)
1	Document (with data on the page)
2	Application (file extension must be .exe, .com, or .bat)
3	Folder (plain)
4	Folder (open)

Return Values

Returns NOERROR if *lpIconIndex* contains the correct system image list index, or S_FALSE if an icon can't be obtained for this object.

Remarks

If you are unable to get an icon for this object using **GetIconOf**, use the **IShellFolder::GetUIObjectOf** method to get an object that supports **the IExtractIcon::Extract** method.

Note to Callers The index returned is from the system image list.

Note to Implementers If the icon index used is not one of the standard images listed, it is the implementer's responsibility to add the image to the system image list and then place the index into the *lpIconIndex* parameter. To prevent the system image list from growing too large, each image should only be added once.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellIcon

IShellIconOverlay

Icon overlays are small images placed at the lower-left corner of the icon that represents a shell object. They are normally used to add some extra information to the icon. A commonly used icon overlay is the small arrow that indicates that a file or folder icon represents a link. A namespace extension can specify icon overlays for the objects it contains by implementing an **IShellIconOverlay** interface.

Icon overlays are part of the system image list. They have two identifiers. One is a one-based overlay index that identifies the overlay relative to other overlays in the image list. The other is an image index that identifies the actual image. These two indexes are equivalent to the values that you assign to the *iOverlay* and *iImage* parameters, respectively, when you add an icon overlay to a private image list with **ImageList_SetOverlayImage**.

Before displaying the icon for an object, the shell calls the associated folder's **IShellIconOverlay** interface to query whether the object's icon should have an overlay. Normally it calls **IShellIconOverlay::GetOverlayIndex** to request the overlay's overlay index. In some cases, the shell might call **IShellIconOverlay::GetOverlayIconIndex** to request the overlay's image index. To specify an icon overlay, the methods must return the requested index. Otherwise, they return S_FALSE.

To specify an icon overlay, both methods must first get the overlay's overlay index in the system image list by calling **SHGetIconOverlayIndex**. When **SHGetIconOverlayIndex** is called for the first time, the shell uses the overlay's file name and index within the file to add the image to the system image list. Once an overlay is in the system image list, the shell simply uses the file name and index as an identifier. You can also use **SHGetIconOverlayIndex** to get the overlay index of several standard system overlays.

IShellIconOverlay::GetOverlayIndex simply returns the overlay index to the shell. **IShellIconOverlay::GetOverlayIconIndex** must call **INDEXTOOVERLAYMASK** to convert the overlay index to the equivalent image index.

Note The number of different icon overlay handlers that the system can support is limited by the amount of space available for icon overlays in the system image list. There are currently fifteen slots allotted for icon overlays, some of which are reserved by the system. For this reason, icon overlays should be specified only if there are no satisfactory alternatives.

When to Implement

This interface is implemented by namespace extensions that need to specify icon overlays for their objects.

When to Use

This interface is not normally used by applications.

Methods

IShellIconOverlay implements the following methods in addition to **IUnknown**:

GetOverlayIndex	Returns the overlay index in the system image list.
GetOverlayIconIndex	Returns the index of the icon overlay in the system image list.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellIconOverlay::GetOverlayIconIndex

Returns the index of the icon overlay in the system image list.

```
HRESULT GetOverlayIconIndex(
    LPCITEMIDLIST pidlItem,
    int *pIconIndex
);
```

Parameters

pidlItem

[in] Address of an **ITEMIDLIST** structure that identifies the object whose icon is being displayed.

pIconIndex

[out] Address of the zero-based index of the icon overlay's image in the system image list. This index is equivalent to the *iImage* value that is specified when you add an overlay image to a private image list with the **ImageList_SetOverlayImage** function.

Return Values

Returns NOERROR if successful, or an OLE error code otherwise.

Remarks

To get the overlay's image index in the system image list, you must first call **SHGetIconOverlayIndex to get the overlay** index. Then call **INDEXTOOVERLAYMASK** to convert the overlay index into the equivalent image index.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellIconOverlay

IShellIconOverlay::GetOverlayIndex

Returns the overlay index in the system image list.

```
HRESULT GetOverlayIndex(
    LPCITEMIDLIST pidlItem,
    int *pIndex
);
```

Parameters

pidlItem

[in] Address of an **ITEMIDLIST** structure that identifies the object whose icon is being displayed.

pIndex

[out] Address of the overlay's one-based overlay index in the system image list. This index is equivalent to the *iOverlay* value that is specified when you add an overlay image to a private image list with the **ImageList_SetOverlayImage** function.

Return Values

Returns NOERROR if successful, or an OLE error code otherwise.

Remarks

To get the overlay index in the system image list, call **SHGetIconOverlayIndex**.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellIconOverlay

IShellIconOverlayIdentifier

Icon overlays are small images placed at the lower-left corner of the icon that represents a shell object in Windows Explorer or on the desktop. They are used to add some extra information to the object's normal icon. A commonly used icon overlay is the small arrow that indicates that a file or folder is actually a link. You can specify custom icon overlays for shell objects by implementing and registering an icon overlay handler.

Icon overlay handlers are COM objects that are associated with a particular icon overlay. All communication between the shell and the handler takes place through the handler's **IShellIconOverlayIdentifier** interface. For a general discussion of icon overlay handlers, see *Creating Icon Overlay Handlers*.

When to Implement

This interface must be implemented by all icon overlay handlers.

When to Use

This interface is not normally called by applications.

Methods

IShellIconOverlayIdentifier implements the following methods in addition to **IUnknown**.

GetOverlayInfo	Provides the location of the icon overlay's bitmap.
GetPriority	Specifies the priority of an icon overlay.
IsMemberOf	Specifies whether an icon overlay should be added to a shell object's icon.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellIconOverlayIdentifier::GetOverlayInfo

Provides the location of the icon overlay's bitmap.

```
HRESULT GetOverlayInfo(
    LPWSTR pwszIconFile,
    int cchMax,
    int *pIndex,
    DWORD *pdwFlags
);
```

Parameters

pwszIconFile

[out] NULL-terminated Unicode string that contains the fully qualified path of the file containing the icon. The .dll, .exe, and .ico file types are all acceptable. You must set the ISIOI_ICONFILE flag in *pdwFlags* if you return a file name.

cchMax

[in] Size of the *pwszIconFile* buffer.

pIndex

[out] Address of the index of the icon in a file containing multiple icons. You must set the ISIOI_ICONINDEX flag in *pdwFlags* if you return an index.

pdwFlags

Address of a flag that specifies what information is being returned. This parameter can be one or both of the following values.

ISIOI_ICONFILE The path of the icon file is returned through *pwszIconFile*.

ISIOI_ICONINDEX There is more than one icon in *pwszIconFile*. The icon's index is returned through *pIndex*.

Remarks

This method is called by the shell at startup so that the handler's icon overlay can be added to the system image list. After initialization is complete, the shell calls **GetOverlayInfo** when it needs to display the handler's icon overlay.

Note Once the image has been loaded into the system image list during initialization, it cannot be changed. After initialization, the file name and index are used only to identify the icon overlay. The system will not load a new icon overlay. When **GetOverlayInfo** is called, your handler must return the same file name and index that were specified when the function was first called.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellIconOverlayIdentifier

IShellIconOverlayIdentifier::GetPriority

Specifies the priority of an icon overlay.

```
HRESULT GetPriority(
    int *pPriority
);
```

Parameters

pPriority

[out] Address of a value that indicates the priority of the overlay identifier. Possible values range from zero to 100, with zero the highest priority.

Return Values

Returns S_OK if successful, or an OLE error code otherwise.

Remarks

If more than one icon overlay is available for an object, the one with highest priority is chosen. The shell has a set of internal rules that determine priority for many cases. The value returned by **GetPriority** is used for those cases where the shell's internal rules do not apply. Normally, you should set the value to zero. However, the priority value is useful when you have implemented two or more icon overlay handlers that can request icon overlay icons for the same object. By setting the priority values appropriately, you can specify which of the requested icon overlays will be displayed.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellIconOverlayIdentifier

IShellIconOverlayIdentifier::IsMemberOf

Specifies whether an icon overlay should be added to a shell object's icon.

```
HRESULT IsMemberOf(
    LPCWSTR pwszPath,
    DWORD dwAttrib
);
```

Parameters

pwszPath

Unicode string that contains the fully qualified path of the shell object.

dwAttrib

Object's attributes. For a complete list of file attributes and their associated flags, see **IShellFolder::GetAttributesOf**.

Return Values

This method returns one of the following:

S_OK	The icon overlay should be displayed.
S_FALSE	The icon overlay should not be displayed.
E_FAIL	The operation failed.

Remarks

The shell calls this method to determine whether it should display a handler's icon overlay for a particular object. Icon overlay handlers are normally intended to work with a particular group of files. A typical example is a **file class**, identified by a specific file name extension. An icon overlay handler might request an icon overlay for all members of the file class. Some handlers request an icon overlay only if a member of the file class is in a particular state. However, icon overlay handlers are free to request their icon overlay for any object that they want.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellIconOverlayIdentifier

IShellLink

The **IShellLink** interface allows shell links to be created, modified, and resolved.

Methods

IShellLink supports the following methods:

GetArguments	Retrieves the command-line arguments associated with a shell link object.
GetDescription	Retrieves the description string for a shell link object.
GetHotkey	Retrieves the hot key for a shell link object.
GetIconLocation	Retrieves the location (path and index) of the icon for a shell link object.
GetIDList	Retrieves the list of item identifiers for a shell link object.
GetPath	Retrieves the path and file name of a shell link object.
GetShowCmd	Retrieves the show (SW_) command for a shell link object.
GetWorkingDirectory	Retrieves the name of the working directory for a shell link object.
Resolve	Resolves a shell link by searching for the shell link object and updating the shell link path and its list of identifiers (if necessary).
SetArguments	Sets the command-line arguments associated with a shell link object.
SetDescription	Sets the description string for a shell link object.
SetHotkey	Sets the hot key for a shell link object.
SetIconLocation	Sets the location (path and index) of the icon for a shell link object.
SetIDList	Sets the list of item identifiers for a shell link object.
SetPath	Sets the path and file name of a shell link object.
SetRelativePath	Sets the relative path for a shell link object.
SetShowCmd	Sets the show (SW_) command for a shell link object.
SetWorkingDirectory	Sets the name of the working directory for a shell link object.

Remarks

Note The **IShellLink** interface has an ANSI version (IShellLinkA) and a Unicode version (IShellLinkW). The version that will be used depends on whether you compile for ANSI or Unicode. However, Windows 95 and Windows 98 only support **IShellLinkA**.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellLink::GetArguments

Retrieves the command-line arguments associated with a shell link object.

```
HRESULT STDMETHODCALLTYPE GetArguments(
    LPSTR pszArgs,
    int cchMaxPath
);
```

Parameters

pszArgs
Address of a buffer that receives the command-line arguments.

cchMaxPath
Maximum number of characters to copy to the buffer pointed to by the *pszArgs* parameter.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.

Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellLink

IShellLink::GetDescription

Retrieves the description string for a shell link object.

```
RESULT STDMETHODCALLTYPE GetDescription(
    LPSTR pszName,
    int cchMaxName
);
```

Parameters

pszName
Address of a buffer that receives the description string.

cchMaxName
Maximum number of characters to copy to the buffer pointed to by the *pszName* parameter.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellLink

IShellLink::GetHotkey

Retrieves the hot key for a shell link object.

```
HRESULT STDMETHODCALLTYPE GetHotkey(
    WORD *pwHotkey
);
```

Parameters

pwHotkey

Address of the hot key. The virtual key code is in the low-order byte, and the modifier flags are in the high-order byte. The modifier flags can be a combination of the following values:

HOTKEYF_ALT	ALT key
HOTKEYF_CONTROL	CTRL key
HOTKEYF_EXT	Extended key
HOTKEYF_SHIFT	SHIFT key

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellLink

IShellLink::GetIconLocation

Retrieves the location (path and index) of the icon for a shell link object.

```
HRESULT STDMETHODCALLTYPE GetIconLocation(
    LPSTR pszIconPath,
    int cchIconPath,
    int *piIcon
);
```

Parameters

pszIconPath

Address of a buffer that receives the path of the file containing the icon.

cchIconPath

Maximum number of characters to copy to the buffer pointed to by the *pszIconPath* parameter.

piIcon

Address of a value that receives the index of the icon.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellLink

IShellLink::GetIDList

Retrieves the list of item identifiers for a shell link object.

```
HRESULT STDMETHODCALLTYPE GetIDList(
    LPITEMIDLIST *ppidl
);
```

Parameters

ppidl

Address of a pointer to a list of item identifiers.

Return Values

Returns NOERROR if the operation is successful, and one or more valid PIDLs is retrieved. If the operation is successful, but no PIDLs are retrieved, it returns S_FALSE with *ppidl* set to NULL. Otherwise, it returns an OLE-defined error value.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellLink

IShellLink::GetPath

Retrieves the path and file name of a shell link object.

```
HRESULT STDMETHODCALLTYPE GetPath(
    LPSTR pszFile,
    int cchMaxPath,
    WIN32_FIND_DATA *pfd,
    DWORD fFlags
);
```

Parameters

pszFile

Address of a buffer that receives the path and file name of the shell link object.

cchMaxPath

Maximum number of bytes to copy to the buffer pointed to by the *pszFile* parameter.

pfd

Address of a **WIN32_FIND_DATA** structure that contains information about the shell link object.

fFlags

Flags that specify the type of path information to retrieve. This parameter can be a combination of the following values:

Flag	Description
SLGP_SHORTPATH	Retrieves the standard short (8.3 format) file name.
SLGP_UNCPRIORITY	Retrieves the Universal Naming Convention (UNC) path name of the file.
SLGP_RAWPATH	Retrieves the raw path name. A raw path is something that might not exist and may include environment variables that need to be expanded.

Return Values

Returns NOERROR if the operation is successful, and a valid path is retrieved. If the operation is successful, but no path is retrieved, it returns S_FALSE and *pszPath* will be empty. Otherwise, it returns an OLE-defined error value.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellLink

IShellLink::GetShowCmd

Retrieves the show command for a shell link object.

```
HRESULT STDMETHODCALLTYPE GetShowCmd(
    int *piShowCmd
);
```

Parameters

piShowCmd

Pointer to the command. The following commands are supported:

Value	Description
SW_SHOWNORMAL	Activates and displays a window. If the window is minimized or maximized, the system restores it to its original size and position. An application should specify this flag when displaying the window for the first time.
SW_SHOWMAXIMIZED	Activates the window and displays it as a maximized window.
SW_SHOWMINIMIZED	Activates the window and displays it as a minimized window.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

The show command is used to set the initial show state of the corresponding object. This is one of the SW_xxx values described in **ShowWindow**.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellLink, **IShellLink::SetShowCmd**

IShellLink::GetWorkingDirectory

Retrieves the name of the working directory for a shell link object.

```
HRESULT STDMETHODCALLTYPE GetWorkingDirectory(
    LPSTR pszDir,
    int cchMaxPath
);
```

Parameters

pszDir
Address of a buffer that receives the name of the working directory.

cchMaxPath
Maximum number of characters to copy to the buffer pointed to by the *pszDir* parameter. The name of the working directory is truncated if it is longer than the maximum specified by this parameter.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellLink

IShellLink::Resolve

Attempts to find the target of a shell link, even if it has been moved or renamed.

```
HRESULT Resolve(
    HWND hwnd,
    DWORD fFlags
);
```

Parameters

hwnd

Handle to the window that the shell will use as the parent for a dialog box. The shell displays the dialog box if it needs to prompt the user for more information while resolving a shell link.

fFlags

Action flags. This parameter can be a combination of the following values:

SLR_INVOKE_MSI

Call the Microsoft Windows Installer.

SLR_NOLINKINFO

Disable distributed link tracking. By default, distributed link tracking tracks removable media across multiple devices based on the volume name. It also uses the UNC path to track remote file systems whose drive letter has changed. Setting SLR_NOLINKINFO disables both types of tracking.

SLR_NO_UI

Do not display a dialog box if the link cannot be resolved. When SLR_NO_UI is set, the high-order word of *fFlags* can be set to a time-out value that specifies the maximum amount of time to be spent resolving the link. The function returns if the link cannot be resolved within the time-out duration. If the high-order word is set to zero, the time-out duration will be set to the default value of 3,000 milliseconds (3 seconds). To specify a value, set the high word of *fFlags* to the desired time-out duration, in milliseconds.

SLR_NOUPDATE

Do not update the link information.

SLR_NOSEARCH

Do not execute the search heuristics.

SLR_NOTRACK

Do not use distributed link tracking.

SLR_UPDATE

If the link object has changed, update its path and list of identifiers. If SLR_UPDATE is set, you do not need to call **IPersistFile::IsDirty** to determine whether or not the link object has changed.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

Following link creation, the name or location of the target may change. The **IShellLink::Resolve** method first retrieves the path associated with the link. If the object is no longer there or has been renamed, **Resolve** will attempt to find it. If successful, and the following conditions are met:

- The SLR_UPDATE flag is set.
- The target has been moved or renamed, updating the internal state of the shell link object to refer to the new target.
- The shell link object was loaded from a file through **IPersistFile**.

The file that the link object was loaded from will be updated to reflect the new state of the link object. The client can also call **IPersistFile::IsDirty** method to determine whether the link object has changed and the file needs to be updated.

Resolve has two approaches to finding target objects. The first is the distributed link tracking service. If the service is available, it can find an object that was on an NTFS version 5.0 volume and was moved to another location on that volume. It can also find an object that was moved to another NTFS version 5.0 volume, including volumes on other computers. To suppress the use of this service, set the SLR_NOTRACK flag.

If distributed link tracking is not available or fails to find the link object, **Resolve** attempts to find it with search heuristics. It first looks in the object's last known directory for an object with a different name but the same attributes and file creation time. Next, it recursively searches subdirectories in the vicinity of the object's last known directory. It looks for an object with the same name or creation time. Finally, **Resolve** looks for a matching object on the desktop and other local volumes. To suppress the use of the search heuristics, set the SLR_NOSEARCH flag.

If both approaches fail, the system will display a dialog box prompting the user for a location. To suppress the dialog box, set the SLR_NO_UI flag.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellLink

IShellLink::SetArguments

Sets the command-line arguments for a shell link object.

```
HRESULT STDMETHODCALLTYPE SetArguments(
    LPCSTR pszArgs
);
```

Parameters

pszArgs
Address of a buffer that contains the new command-line arguments.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

This method is useful when creating a link to an application that takes special flags as arguments, such as a compiler.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellLink

IShellLink::SetDescription

Sets the description for a shell link object. The description can be any application-defined string.

```
HRESULT STDMETHODCALLTYPE SetDescription(
    LPCSTR pszName
);
```

Parameters

pszName
Address of a buffer containing the new description string.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellLink

IShellLink::SetHotkey

Sets a hot key for a shell link object.

```
HRESULT STDMETHODCALLTYPE SetHotkey(
    WORD wHotkey
);
```

Parameters

wHotkey

New hot key. The virtual key code is in the low-order byte, and the modifier flags are in the high-order byte. The modifier flags can be a combination of the values specified in the description of the **IShellLink::GetHotkey** method.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

Setting a hot key allows the user to activate the object by pressing a particular combination of keys.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellLink

IShellLink::SetIconLocation

Sets the location (path and index) of the icon for a shell link object.

```
HRESULT STDMETHODCALLTYPE SetIconLocation(
    LPCSTR pszIconPath,
    int iIcon
);
```

Parameters

pszIconPath

Address of a buffer to contain the path of the file containing the icon.

iIcon

Index of the icon.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.

+ See Also

IShellLink

IShellLink::SetIDList

Sets the list of item identifiers for a shell link object.

```
HRESULT STDMETHODCALLTYPE SetIDList(
    LPCITEMIDLIST pidl
);
```

Parameters

pidl

Address of a list of item identifiers.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

This method is useful when an application needs to set a shell link to an object that is not a file, such as a Control Panel application, a printer, or another computer.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellLink

IShellLink::SetPath

Sets the path and file name of a shell link object.

```
HRESULT STDMETHODCALLTYPE SetPath(
    LPCSTR pszFile
);
```

Parameters

pszFile
Address of a buffer that contains the new path.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellLink

IShellLink::SetRelativePath

Sets the relative path to the shell link object.

```
HRESULT SetRelativePath(
    LPCSTR pszPathRel,
    DWORD dwReserved
);
```

Parameters

pszPathRel

Address of a buffer that contains the new relative path. It should be a file name, not a folder name.

dwReserved

Reserved. Set this parameter to zero.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Example

Consider the following scenario:

- You have a link: c:\MyLink.lnk
- The link target is c:\MyDocs\MyFile.txt
- You want to move the link and MyDocs\MyFile.txt to d:\.

You can assist the resolution process by creating the original link with a relative path before the shortcut is saved:

```
::SetRelativePath("c:\MyLink.lnk", NULL);
```

Before the shortcut is resolved, set a new relative path, and the Resolve code will find the file in its new location.

```
::SetRelativePath("d:\MyLink.lnk", NULL);
```

Remarks

Clients commonly define a relative link when it may be moved along with its target, causing the absolute path to become invalid. The **SetRelativePath** method can be used to help the link resolution process find its target based on a common path prefix between the target and the relative path. To assist in the resolution process, clients should set the relative path as part of the link creation process.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellLink, **IShellLink::Resolve**, **IShellLink::SetPath**, **IShellLink::SetIDList**

IShellLink::SetShowCmd

Sets the show command for a shell link object. The show command sets the initial show state of the window.

```
HRESULT STDMETHODCALLTYPE SetShowCmd(
    int iShowCmd
);
```

Parameters

iShowCmd

Command. **SetShowCmd** accepts one of the following **ShowWindow** commands:

SW_SHOWNORMAL
Activates and displays a window. If the window is minimized or maximized, the system restores it to its original size and position. An application should specify this flag when displaying the window for the first time.

SW_SHOWMAXIMIZED
Activates the window and displays it as a maximized window.

SW_SHOWMINIMIZED
Activates the window and displays it as a minimized window.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellLink, **IShellLink::GetShowCmd**

IShellLink::SetWorkingDirectory

Sets the name of the working directory for a shell link object.

```
HRESULT STDMETHODCALLTYPE SetWorkingDirectory(
    LPCSTR pszDir
);
```

Parameters

pszDir

Address of a buffer that contains the name of the new working directory.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

The working directory is optional unless the target requires a working directory. For example, if an application creates a shell link to a Word document that uses a template residing in a different directory, the application would use this method to set the working directory.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellLink

IShellLinkDataList

The **IShellLinkDataList** interface allows an application to attach extra data blocks to a shell link. It provides methods to add, copy, or remove data blocks. The data blocks are in the form of a structure. The first two members are the same for all data blocks. The first member gives the structure's size. The second member is a signature that identifies the type of data block. The remaining members hold the block's data. There are five types of data block currently supported.

Data block structure	Description
EXP_DARWIN_LINK	The link's Windows Installer ID.
EXP_SPECIAL_FOLDER	Special folder information.
EXP_SZ_LINK	The target name.
NT_CONSOLE_PROPS	Console properties.
NT_FE_CONSOLE_PROPS	The console's code page.

When to Implement

This interface is not implemented by applications.

When to Use

Use this interface if your application needs to add extra data blocks to a shell link.

Methods

IShellLinkDataList exposes the following methods in addition to **IUnknown**:

AddDataBlock	Adds a data block to a link.
CopyDataBlock	Gets a copy of a link's data block.
GetFlags	Gets the current option settings.
RemoveDataBlock	Removes a data block from a link.
SetFlags	Specifies the current option settings.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellLinkDataList::AddDataBlock

Adds a data block to a link.

```
HRESULT AddDataBlock(
    VOID *pDataBlock
);
```

Parameters

pDataBlock

[in] Data block structure. For a list of supported structures, see **IShellLinkDataList**.

Return Values

Returns S_OK if successful, or an OLE error code otherwise.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellLinkDataList

IShellLinkDataList::CopyDataBlock

Gets a copy of a link's data block.

```
HRESULT CopyDataBlock(
    DWORD dwSig
    VOID **ppDataBlock)
);
```

Parameters

dwSig

[in] Data block's signature. The signature value for a particular type of data block can be found in its structure reference. For a list of supported data block types and their associated structures, see **IShellLinkDataList**.

ppDataBlock

[out] Address of a pointer to a copy of the data block structure.

Return Values

Returns S_OK if successful, or an OLE error code otherwise.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellLinkDataList

IShellLinkDataList::GetFlags

Gets the current option settings.

```
HRESULT GetFlags(
    DWORD *pdwFlags
);
```

Parameters

pdwFlags

[out] Address of a flag value that indicates the current option settings. This parameter can be a combination of the following values.

SLDF_FORCE_NO_LINKINFO	Do not create link information. Distributed tracking will be disabled.
SLDF_HAS_ARGS	The link has arguments.
SLDF_HAS_DARWINID	The link is a special Windows Installer link.
SLDF_HAS_EXP_ICON_SZ	The link contains an expandable environment string for the icon path.
SLDF_HAS_EXP_SZ	The link contains expandable environment strings such as "%windir%".
SLDF_HAS_ICONLOCATION	The link has an icon location.
SLDF_HAS_ID_LIST	The shell link was saved with an ID list.
SLDF_HAS_LINK_INFO	The shell link was saved with link information to enable distributed tracking.
SLDF_HAS_LOGO3ID	Not currently supported.
SLDF_HAS_NAME	The link has a name.
SLDF_HAS_RELPATH	The link has a relative path.
SLDF_HAS_WORKINGDIR	The link has a working directory.
SLDF_RUNAS_USER	Run this link as a different user.
SLDF_RUN_IN_SEPARATE	Run the 16-bit target application in a separate VDM/WOW.
SLDF_UNICODE	Strings are in Unicode.

Return V5alues

Returns S_OK if successful, or an OLE error code otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellLinkDataList

IShellLinkDataList::RemoveDataBlock

Removes a data block from a link.

```
HRESULT RemoveDataBlock(
    DWORD dwSig
);
```

Parameters

dwSig

[in] Data block's signature. The signature value for a particular type of data block can be found in its structure reference. For a list of supported data block types and their associated structures, see **IShellLinkDataList**.

Return Values

Returns S_OK if successful, or an OLE error code otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellLinkDataList

IShellLinkDataList::SetFlags

Specifies the current option settings.

```
HRESULT SetFlags(
    DWORD dwFlags
);
```

Parameters

dwFlags

[in] Flags that specify the option settings. This parameter can be a combination of the following values:

SLDF_FORCE_NO_LINKINFO	Do not create link information. Distributed tracking will be disabled.
SLDF_HAS_ARGS	The link has arguments.
SLDF_HAS_DARWINID	The link is a special Windows® Installer link.
SLDF_HAS_EXP_ICON_SZ	The link contains an expandable environment string for the icon path.
SLDF_HAS_EXP_SZ	The link contains expandable environment strings such as "%windir%".
SLDF_HAS_ICONLOCATION	The link has an icon location.
SLDF_HAS_ID_LIST	The shell link was saved with an ID list.
SLDF_HAS_LINK_INFO	The shell link was saved with link information to enable distributed tracking.
SLDF_HAS_LOGO3ID	Not currently supported.
SLDF_HAS_NAME	The link has a name.
SLDF_HAS_RELPATH	The link has a relative path.
SLDF_HAS_WORKINGDIR	The link has a working directory.
SLDF_RUNAS_USER	Run this link as a different user.
SLDF_RUN_IN_SEPARATE	Run the 16-bit target application in a separate VDM/WOW.
SLDF_UNICODE	Strings are in Unicode.

Return Values

Returns S_OK if successful, or an OLE error code otherwise.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellLinkDataList

IShellPropSheetExt

The **IShellPropSheetExt** interface allows a property sheet handler to add or replace pages in the property sheet displayed for a file object.

IShellPropSheetExt methods	Description
AddPages	Adds one or more pages to a property sheet for a file object.
ReplacePage	Replaces a page in a property sheet for a Control Panel object.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellPropSheetExt::AddPages

Adds one or more pages to a property sheet that the shell displays for a file object. When it is about to display the property sheet, the shell calls this method for each property sheet handler registered to the file type.

```
HRESULT STDMETHODCALLTYPE AddPages(
    LPFNADDPROPSHEETPAGE lpfnAddPage,
    LPARAM lParam
);
```

Parameters

lpfnAddPage

Address of a function that the property sheet handler calls to add a page to the property sheet. The function takes a property sheet handle returned by the **CreatePropertySheetPage** function and the *lParam* parameter passed to the **AddPages** method.

lParam

Parameter to pass to the function specified by the *lpfnAddPage* method.

Return Values

Returns S_OK if successful. If the method fails, an OLE-defined error code is returned. **Version 4.71.** If successful, returns a one-based index to specify the page that should be initially displayed. See the remarks for more information.

If the method fails, an OLE-defined error code is returned.

Remarks

For each page the property sheet handler needs to add to a property sheet, the handler fills a **PROPSHEETPAGE** structure, calls the **CreatePropertySheetPage** function, and then calls the function specified by the *lpfnAddPage* parameter.

Prior to **Version 4.71**, the property sheet determines which page will be initially displayed. With **Version 4.71** and later, you can request that a particular property sheet page be displayed first, instead of the default page. To do so, return the one-based index of the desired page. For example, if you want the second of three pages displayed, the return value should be 2. Note that this return value is a request. The property sheet may still display the default page.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellPropSheetExt

IShellPropSheetExt::ReplacePage

Replaces a page in a property sheet for a Control Panel object.

```
HRESULT STDMETHODCALLTYPE ReplacePage(
    UINT uPageID,
    LPFNADDPROPSHEETPAGE lpfnReplacePage,
    LPARAM lParam
);
```

Parameters

uPageID

Identifier of the page to replace. The values for this parameter for Control Panels can be found in the Cplext.h header file.

lpfnReplacePage

Address of a function that the property sheet handler calls to replace a page to the property sheet. The function takes a property sheet handle returned by the **CreatePropertySheetPage** function and the *lParam* parameter passed to the **ReplacePage** method.

lParam
Parameter to pass to the function specified by the *lpfnReplacePage* parameter.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

To replace a page, a property sheet handler fills a **PROPSHEETPAGE** structure, calls **CreatePropertySheetPage**, and then calls the function specified by *lpfnReplacePage*.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellPropSheetExt

IShellView

The **IShellView** interface is implemented to present a view in the Windows Explorer or folder windows. The object that exposes **IShellView** is created by a call to the **IShellFolder::CreateViewObject** method. This provides the channel of communication between a view object and Windows Explorer's outermost frame window. The communication involves the translation of messages, the state of the frame window (activated or deactivated), the state of the document window (activated or deactivated), and the merging of menus and toolbar items.

When to Implement

This interface is implemented by namespace extensions that display themselves in Windows Explorer's namespace. This object is created by the **IShellFolder** object that hosts the view.

When to Use

These methods are used by the shell view's Windows Explorer window to manipulate objects while they are active.

IShellView is derived from **IOleWindow**. The following are the methods specific to **IShellView**:

IShellView methods	Description
AddPropertySheetPages	Allows the view to add pages to the Options property sheet.
CreateViewWindow	Creates the view window.
DestroyViewWindow	Destroys the view window.
EnableModeless	Enables or disables modeless dialog boxes. Not in use by Windows Explorer at this time.
EnableModelessSV	Not currently in use.
GetCurrentInfo	Returns the current folder settings.
GetItemObject	Allows callers to get an object that represents something in the view.
Refresh	Refreshes the display in response to user input.
SaveViewState	Saves the shell's view settings so the current state can be restored during a subsequent browsing session.
SelectItem	Changes the state of items within the shell view window.
TranslateAccelerator	Translates accelerator key strokes when a namespace extension's view has the focus.
UIActivate	Called whenever the activation state of the view window is changed by an event not caused by the shell view itself.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellView::AddPropertySheetPages

Allows the view to add pages to the Options property sheet from the **View** menu.

```
HRESULT AddPropertySheetPages(
    DWORD dwReserved,
    LPFNADDPROPSHEETPAGE lpfn,
    LPARAM lparam
);
```

Parameters

dwReserved
Reserved for future use.

lpfn
Address of the callback function used to add the pages.

lparam
Value that must be passed to the callback function in the *lpfn* parameter.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

Note to Implementers Windows Explorer calls this method when it is opening the Options property sheet from the **View** menu. Views can add pages by creating them and calling the callback function with the page handles.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellView

IShellView::CreateViewWindow

Creates a view window. This can be either the right pane of Windows Explorer or the client window of a folder window.

```
RESULT CreateViewWindow(
    ISHELLLINK *lpPrevView,
    LPFOLDERSETTINGS lpfs,
    IShellBrowser *psb,
    RECT *prcView,
    HWND *phWnd
);
```

Parameters

lpPrevView
Address of the view window being exited. Views can use this parameter to communicate with a previous view of the same implementation. This can be used to optimize browsing between like views. This pointer may be NULL.

lpfs
Address of a **FOLDERSETTINGS** structure. The view should use this when creating its view.

psb
Address of the current instance of the **IShellBrowser** interface. The view should call this interface's **AddRef** method and keep the interface pointer to allow communication with the Windows Explorer window.

prcView
Dimensions of the new view, in client coordinates.

phWnd
Address of the window handle being created.

Return Values

Returns an OLE success code if successful, or an OLE error code otherwise. Use the **SUCCEEDED** and **FAILED** macros to determine whether the operation succeeded or failed.

Remarks

Note to Callers Call this method when the view needs to be created.

Note to Implementers Create your view window and restore any persistent state by calling the **IShellBrowser::GetViewStateStream** method.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellView

IShellView::DestroyViewWindow

Destroys the view window.

```
HRESULT DestroyViewWindow(void)
```

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

Windows Explorer calls this method when a folder window or Explorer is being closed.

Note to Implementers Clean up all states that represent the view, including the window and any other associated resources.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellView

IShellView::EnableModeless

Enables or disables modeless dialog boxes. This method is not currently implemented.

```
HRESULT EnableModeless(
    BOOL fEnable
);
```

Parameters

fEnable
Nonzero to enable modeless dialog box windows or zero to disable them.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellView

IShellView::EnableModelessSV

Not currently implemented.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellView::GetCurrentInfo

Retrieves the current folder settings.

```
HRESULT GetCurrentInfo(
    LPFOLDERSETTINGS lpfs
);
```

Parameters

lpfs

Address of a **FOLDERSETTINGS** structure to receive the settings.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

Windows Explorer uses this method to query the view for standard settings.

Note to Callers This method is used to get the current view settings of the view.

Note to Implementers Return as many of the settings as apply. This is intended to maintain the same basic settings when the user browses from view to view. For example, if the user sets the Details view, that view should be maintained as the user goes from one folder to the other in Windows Explorer mode.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellView

IShellView::GetItemObject

Retrieves an interface that refers to data presented in the view.

```
HRESULT GetItemObject(
    UINT uItem,
    REFIID riid,
    LPVOID *ppv
);
```

Parameters

uItem

Constants that refer to an aspect of the view. This parameter can be any of the following values:

SVGIO_BACKGROUND
Refers to the background of the view. It is used with IID_IContextMenu to get a context menu for the view background.

SVGIO_SELECTION
Refers to the currently selected items. IID_IDataObject uses this constant to get a data object that represents the selected items.

SVGIO_ALLVIEW
Same as SVGIO_SELECTION but refers to all items in the view.

riid

Identifier of the COM interface being requested.

ppv

Address that receives the interface pointer. If an error occurs, the pointer returned must be NULL.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

Used by the common dialogs to get the selected items from the view.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellView

IShellView::Refresh

Refreshes the view's contents in response to user input.

```
HRESULT Refresh(void)
```

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

Tells the view to refresh its contents, revalidating any view information it has.

Note to Callers Windows Explorer calls this method when the F5 key is pressed on an already open view.

Note to Implementers Refill the view by going to any underlying storage for the contents.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellView

IShellView::SaveViewState

Saves the shell's view settings so the current state can be restored during a subsequent browsing session.

```
HRESULT SaveViewState(void);
```

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

The shell view obtains a view stream by calling the **IShellBrowser::GetViewStateStream** method and stores the current view state in that stream.

Note to Callers Windows Explorer calls this method when it wants to save the view state for a view.

Note to Implementers Be sure to make the format of the data stored in the stream robust enough that different versions of the implementation can read it without error.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellView

IShellView::SelectItem

Changes the selection state of one or more items within the shell view window.

```
HRESULT SelectItem(
    LPCITEMIDLIST pidlItem,
    UINT uFlags
);
```

Parameters

pidlItem

Address of the **ITEMIDLIST** structure.

uFlags

Flag specifying what type of selection to apply. This parameter can be one of the following values:

SVSI_DESELECT	Deselect the specified item.
SVSI_DESELECTOTHERS	Deselect all but the specified item. If *pidlItem* is NULL, deselect all items.
SVSI_EDIT	Put *pidlItem* in edit mode.
SVSI_ENSUREVISIBLE	Ensure the item is displayed on the screen.
SVSI_FOCUSED	The item should be given the focus.
SVSI_SELECT	The item should be selected.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

Note to Implementers This method is used to implement the Target command from the **File** menu of the shell shortcut property sheet.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellView

IShellView::TranslateAccelerator

Translates accelerator key strokes when a namespace extension's view has the focus.

```
HRESULT TranslateAccelerator(
    LPMSG lpmsg
);
```

Parameters

lpmsg
Address of the message to be translated.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

If the view returns S_OK, it indicates that the message was translated and should not be translated or dispatched by Windows Explorer.

Remarks

This method is called by Windows Explorer to let the view translate its accelerators.

Note to Callers Windows Explorer calls this method before any other translation if the view has the focus. If the view does not have the focus, it is called after Explorer translates its own accelerators.

Note to Implementers By default, the view should return S_FALSE so that Windows Explorer can either do its own accelerator translation or normal menu dispatching. The view should return S_OK only if it has processed the message as the accelerator and does not want Explorer to process it further.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellView

IShellView::UIActivate

Called when the activation state of the view window is changed by an event that is not caused by the shell view itself. For example, if the TAB key is pressed when the tree has the focus, the view should be given the focus.

```
HRESULT UIActivate(
    UINT uState
);
```

Parameters

uState

Flag specifying the activation state of the window. This parameter can be one of the following values:

SVUIA_ACTIVATE_FOCUS
Windows Explorer has just created the view window with the input focus. This means the shell view should be able to set menu items appropriate for the focused state.

SVUIA_ACTIVATE_NOFOCUS
The shell view is losing the input focus, or it has just been created without the input focus. The shell view should be able to set menu items appropriate for the nonfocused state. This means no selection-specific items should be added.

SVUIA_DEACTIVATE
Windows Explorer is about to destroy the shell view window. The shell view should remove all extended user interfaces. These are typically merged menus and merged modeless pop-up windows.

SVUIA_INPLACEACTIVATE
The shell view is active without focus. This flag is only used when **UIActivate** is exposed through the **IShellView2** interface.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

Before remerging menu items, the shell view typically hooks the WM_SETFOCUS message and calls the **IShellBrowser::OnViewWindowActive** method. The shell view should not hook the WM_KILLFOCUS message to remerge menu items.

Note to Callers Call this method to inform the view of an activation state change.

Note to Implementers Use this method to track the activation state and change any behavior, as appropriate.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellView

IShellView2

The **IShellView2** interface extends the capabilities of **IShellView**.

When to Implement

Implement **IShellView2** if your namespace extension provides services to clients beyond those in **IShellView**.

When to Use

You do not call this interface directly. **IShellView2** is used by the operating system only when it has confirmed that your application is aware of this interface. Objects that expose **IShellView** and **IShellView2** are usually created by other shell folder objects.

IShellView2 implements all the **IShellView** methods as well as **IUnknown**. The following methods are specific to **IShellFolder2**.

IShellFolder2 methods	Description
CreateViewWindow2	Used to request the creation of a Shell View window.
GetView	Used to request the GUID of the default or current Shell View.
HandleRename	Used to change an item's ID.
SelectAndPositionItem	Used to reposition an item.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellView2::CreateViewWindow2

Used to request the creation of a new shell view window. It can be either the right pane of Windows Explorer or the client window of a folder window.

```
HRESULT CreateViewWindow2(
    LPSV2CVW2_PARAMS lpParams
);
```

Parameters

lpParams

Pointer to an **SV2CVW2_PARAMS** structure that defines the new view window.

Return Values

Returns an OLE success code if successful, or an OLE error code otherwise. Use the **SUCCEEDED** and **FAILED** macros to determine whether the operation succeeded or failed.

Remarks

This method supersedes **IShellView::CreateViewWindow**. With **CreateViewWindow2**, developers are not restricted to the standard view modes provided by **IShellView::CreateWindow**, but may also create their own. All view modes are now identified by their globally unique identifier (GUID).

The size of the structure, previous view window, folder settings, parent shell browser, and view rectangle are passed into **CreateViewWindow2** in the first five members of *lpParams*. The method is responsible for creating the new window and passing back its window handle and the GUID of the view mode in the last two parameters. **CreateViewWindow2** should call the parent browser's **IShellBrowser::AddRef** method and store the interface pointer. It can be used for communication with the Windows Explorer window.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellView2::GetView

Used to request the GUID for either the current or default shell view.

```
HRESULT GetView(
    SHELLVIEWID *pvid,
    ULONG uView
);
```

Parameters

pvid

[out] Identifier (GUID) of the requested view.

uView

[in] Type of view requested:

SV2GV_CURRENTVIEW	Current shell view.
SV2GV_DEFAULTVIEW	Default shell view.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellView2::HandleRename

Used to change an item's identifier.

```
HRESULT HandleRename(
    LPCITEMIDLIST pidlNew,
);
```

Parameters

pidlNew

Pointer to an **ITEMIDLIST** structure. The current identifier is passed in and is replaced by the new one.

Return Values

Return NOERROR if successful, or an OLE-defined error code otherwise.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IShellView2::SelectAndPositionItem

Used to select and position an item in a Shell view.

```
HRESULT SelectAndPositionItem(
    LPCITEMIDLIST pidlItem,
    UINT uFlags,
    POINT point
);
```

Parameters

pidlItem

Pointer to an **ITEMIDLIST** structure that uniquely identifies the item of interest.

uFlags

Flags specifying the selection type. You can combine one or more of the following:

SVSI_DESELECT	Deselect the specified item. This flag is incompatible with SVSI_DESELECTOTHERS.
SVSI_DESELECTOTHERS	Deselect all but the specified item. If *pidlItem* is NULL, deselect all items. This flag is incompatible with SVSI_DESELECTOTHERS.
SVSI_EDIT	Put *pidlItem* in edit mode.
SVSI_ENSUREVISIBLE	Ensure the item is displayed on the screen.
SVSI_FOCUSED	The item should be given the focus.
SVSI_SELECT	The item should be selected.
SVSI_TRANSLATEPT	Convert the point from screen coordinates to client coordinates.

point
  POINT structure containing the new position.

Return Values

Return NOERROR if successful, or an OLE-defined error code otherwise.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

ITaskbarList

The **ITaskbarList** interface is used to control the taskbar. It allows you to dynamically add items to the taskbar, remove items from the taskbar, and activate items on the taskbar. See *Modifying the Contents of the Taskbar* for more information about using this interface.

When to Implement

You do not implement **ITaskbarList**; it is implemented by the shell.

When to Use

You use **ITaskbarList** to add items to the taskbar, remove items from the taskbar, and activate items on the taskbar.

ITaskbarList is derived from **IUnknown**. The following methods are specific to **ITaskbarList**:

ITaskbarList methods	Description
ActivateTab	Activates an item on the taskbar.
AddTab	Adds an item to the taskbar.
DeleteTab	Deletes an item from the taskbar.
HrInit	Initializes the taskbar list object.
SetActiveAlt	Marks a taskbar item as active but does not visually activate it.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

ITaskbarList::ActivateTab

Activates an item on the taskbar. The window is not actually activated; the window's item on the taskbar is merely displayed as active.

```
HRESULT ActivateTab(
    HWND hwnd
);
```

Parameters

hwnd

Handle to the window on the taskbar to be displayed as active.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

ITaskbarList::AddTab

Adds an item to the taskbar.

```
HRESULT AddTab(
    HWND hwnd
);
br
```

Parameters

hwnd

Handle to the window to be added to the taskbar.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

Remarks

Any type of window can be added to the taskbar, but it is recommended that the window at least have the WS_CAPTION style.

Any window added with this method must be removed with the **DeleteTab** method when the added window is destroyed.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

ITaskbarList::DeleteTab

Deletes an item from the taskbar.

```
HRESULT DeleteTab(
    HWND hwnd
);
```

Parameters

hwnd

Handle to the window to be deleted from the taskbar.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

ITaskbarList::HrInit

Initializes the taskbar list object. This method must be called before any other **ITaskbarList** methods can be called.

```
HRESULT HrInit(void);
```

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise. If the method fails, no other methods can be called. The calling application should release the interface pointer.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

ITaskbarList::SetActiveAlt

Marks a taskbar item as active but does not visually activate it.

```
HRESULT SetActiveAlt(
    HWND hwnd
);
```

Parameters

hwnd
Handle to the window to be marked as active.

Return Values

Returns NOERROR if successful, or an OLE-defined error code otherwise.

Remarks

SetActiveAlt marks the item associated with *hwnd* as the currently active item for the window's process without changing the pressed state of any item. Any user action that would activate a different tab in that process will activate the tab associated with *hwnd* instead. The active state of the window's item is not guaranteed to be preserved when the process associated with *hwnd* is not active. To ensure that a given tab is always active, call **SetActiveAlt** whenever any of your windows are activated. Calling **SetActiveAlt** with a NULL *hwnd* clears this state.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IUniformResourceLocator

The **IUniformResourceLocator** interface contains methods that can be used to handle an object's Uniform Resource Locator. There are three methods:

IUniformResourceLocator::GetURL	Retrieves an object's uniform resource locator.
IUniformResourceLocator::InvokeCommand	Runs a command on an object's URL.
IUniformResourceLocator::SetURL	Sets an object's URL.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in intshcut.h.

IUniformResourceLocator::GetURL

Retrieves an object's uniform resource locator (URL).

```
HRESULT GetURL(
  LPSTR *ppszURL,
);
```

Parameters

ppszURL

Address of an LPSTR that will be filled in with a pointer to the object's URL.

Return Value

Returns S_OK if the object's URL was retrieved successfully. If the object does not have a URL associated with it, the function returns S_FALSE and sets *ppszURL* to NULL. Otherwise, the return value is one of the following error codes:

E_OUTOFMEMORY	There is not enough memory to complete the operation.
IS_E_EXEC_FAILED	The URL's protocol handler failed to run.
URL_E_INVALID_SYNTAX	The URL's syntax is invalid
URL_E_UNREGISTERED_PROTOCOL	The URL's protocol does not have a registered protocol handler

Remarks

Because this method allocates memory for *ppszURL*, you must instantiate an **IMalloc** interface and free the memory using **IMalloc::Free** when it is no longer needed. The following code fragment provides an example of how this can be done.

```
// START CODE FRAGMENT
{
    // In this example, pURL is a global
    // IUniformResourceLocator pointer.
    LPSTR lpTemp;

    hres = pURL->GetURL(&lpTemp);
    if (SUCCEEDED(hres)){
        IMalloc* pMalloc;
        hres = SHGetMalloc(&pMalloc);
        if (SUCCEEDED(hres)){
            pMalloc->Free(lpTemp);
            pMalloc->Release();
        }
    }
}
// END CODE FRAGMENT
```

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in intshcut.h.

IUniformResourceLocator::InvokeCommand

Runs a command on an object's URL.

```
HRESULT InvokeCommand(
  PURLINVOKECOMMANDINFO pURLCommandInfo,
);
```

Parameters

pURLCommandInfo

Address of a **URLINVOKECOMMANDINFO** structure that contains command information for the function.

Return Value

Returns S_OK if the object's URL was opened successfully. If the object does not have a URL associated with it, the function returns S_FALSE. Otherwise, the return value is one of the following error codes:

E_OUTOFMEMORY	There is not enough memory to complete the operation.
IS_E_EXEC_FAILED	The URL's protocol handler failed to run
URL_E_INVALID_SYNTAX	The URL's syntax is invalid
URL_E_UNREGISTERED_PROTOCOL	The URL's protocol does not have a registered protocol handler.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in intshcut.h.

IUniformResourceLocator::SetURL

Sets an object's URL.

```
HRESULT SetURL(
  LPCSTR SetURL,
  DWORD dwInFlags,
);
```

Parameters

pcszURL

Address of a zero-terminated string that contains the URL to set. The protocol scheme may be included as part of the URL.

dwInFlags

Flag value that specifies the behavior for setting the protocol scheme. This field can contain one of the following values:

IURL_SETURL_FL_GUESS_PROTOCOL

If the protocol scheme is not specified in *pcszURL*, the system automatically chooses a scheme and adds it to the URL.

IURL_SETURL_FL_USE_DEFAULT_PROTOCOL

If the protocol scheme is not specified in *pcszURL*, the system adds the default protocol scheme to the URL.

Return Value

Returns S_OK if the object's URL was set successfully. Otherwise, the return value is one of the following error codes:

E_OUTOFMEMORY	There is not enough memory to complete the operation.
IS_E_EXEC_FAILED	The URL's protocol handler failed to run.
URL_E_INVALID_SYNTAX	The URL's syntax is invalid.
URL_E_UNREGISTERED_PROTOCOL	The URL's protocol does not have a registered protocol handler.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in intshcut.h.

IURLSearchHook

The **IURLSearchHook** interface is used by the browser to translate the address of an unknown URL protocol. When attempting to browse to a URL address that does not contain a protocol, the browser will first attempt to determine the correct protocol from the address. If this is not successful, the browser will create URL Search Hook objects and call each object's **Translate** method until the address is translated or all of the hooks have been queried.

URL Search Hooks are registered by adding a key that contains the object's CLSID string under the following key in the registry:

```
HKEY_LOCAL_MACHINE\
    Software\
    Microsoft\
    Internet Explorer\
    UrlSearchHooks
```

When to Implement

Implement this interface if your application defines a custom URL protocol and if address translation for this protocol is required.

When to Use

You do not normally use this interface; it is called by the browser.

IURLSearchHook method	Description
Translate	Called by the browser when the browser cannot determine the protocol of a URL address.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IURLSearchHook::Translate

Called by the browser when the browser cannot determine the protocol of a URL address.

```
HRESULT Translate(
    LPWSTR lpwszSearchURL,
    DWORD cchBufferSize
);
```

Parameters

lpwszSearchURL

Address of a wide character buffer that, on entry, contains the URL address for which the browser is trying to determine the protocol. On exit, this buffer contains the modified URL address if the method was successful. See the return value for more information.

cchBufferSize

Size, in characters, of the buffer at *lpwszSearchURL*.

Return Values

This method must return one of the following values:

S_OK	The URL address was completely translated. The *lpwszSearchURL* parameter contains the full URL address. The browser will not call any other URL Search Hooks and will attempt to browse to the modified address.
S_FALSE	The URL address has been partially processed, but further translation is still required. The *lpwszSearchURL* parameter contains the result of the processing. The browser will continue executing the rest of the URL Search Hooks.
E_FAIL	The URL address was not translated. The *lpwszSearchURL* parameter has not been modified. The browser will continue executing the rest of the URL Search Hooks.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

CHAPTER 8

Shell Functions

Shell Functions

CPlApplet

Library-defined callback function that serves as the entry point for a Control Panel application.

```
LONG CPlApplet(
    HWND hwndCPl,
    UINT uMsg,
    LONG lParam1,
    LONG lParam2
);
```

Parameters

hwndCPl
Identifier of the main window of the controlling application. Use the *hwndCPl* parameter for dialog boxes or other windows that require a handle to a parent window.

uMsg
Message being sent to the Control Panel application.

lParam1
Additional message-specific information.

lParam2
Additional message-specific information.

Return Values

The return value depends on the message.

For more information, see the descriptions of the individual Control Panel messages.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.

Header: Declared in cpl.h.
Import Library: user-defined.

DefScreenSaverProc

Provides default processing for any messages that a screen-saver application does not process.

```
LONG DefScreenSaverProc(
    HWND hWnd,
    UINT msg,
    WPARAM wParam,
    LPARAM lParam
);
```

Parameters

hWnd
Identifier of the screen-saver window.

msg
Message to be processed. The **DefScreenSaverProc** function responds to messages that affect the screen saver's operation, as detailed in the Remarks section.

If a screen-saver application must perform a different action in response to any of these messages, the application's **ScreenSaverProc** window procedure should process the message.

wParam
Additional message-specific information.

lParam
Additional message-specific information.

Return Values

The return value specifies the result of the message processing and depends on the message sent.

Remarks

A screen-saver application's **ScreenSaverProc** window procedure should use **DefScreenSaverProc** instead of the **DefWindowProc** function to provide default message processing. The **DefScreenSaverProc** function passes any messages that do not affect screen-saver operation to **DefWindowProc**.

The following table describes how the **DefScreenSaverProc** processes a variety of window messages:

WM_ACTIVATE, WM_ACTIVATEAPP, WM_NCACTIVATE
Closes the screen saver if the *wParam* parameter is FALSE. A *wParam* value of FALSE indicates that the screen saver is losing the input focus. The screen saver is closed by sending a **WM_CLOSE** message.

WM_DESTROY
Posts a WM_CLOSE message to close the screen-saver window.

WM_LBUTTONDOWN, WM_RBUTTONDOWN, WM_MBUTTONDOWN, WM_KEYDOWN, WM_KEYUP, WM_MOUSEMOVE
Calls the **PostQuitMessage** function to close the screen saver.

WM_SETCURSOR
Removes the cursor from the screen by setting the cursor to NULL.

WM_SYSCOMMAND
Returns FALSE if the *wParam* parameter of WM_SYSCOMMAND is either SC_CLOSE or SC_SCREENSAVE.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in scrnsave.h.
Import Library: scrnsave.lib.

DllGetVersion

Implemented by many of the Windows shell DLLs to allow applications to obtain DLL-specific version information.

```
HRESULT CALLBACK DllGetVersion(
    DLLVERSIONINFO *pdvi
);
```

Parameters

pdvi
Pointer to a **DLLVERSIONINFO** structure that receives the version information. The **cbSize** member must be filled in before calling the function.

Version 5.0. DLLs that are shipped with Windows 2000 or later systems might return a **DLLVERSIONINFO2** structure. To maintain backward compatibility, the first member of a **DLLVERSIONINFO2** structure is a **DLLVERSIONINFO** structure.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

This function is not an API. It is exported by name from each DLL that implements it. Currently, most of the Windows shell and common controls DLLs implement **DllGetVersion**. These include, but are not limited to, Shell32.dll, Comctl32.dll, Shdocvw.dll, and Shlwapi.dll.

To call this function, use the **LoadLibrary** and **GetProcAddress** functions to obtain the function pointer. The **DLLGETVERSIONPROC** type is used as the data type for the function pointer. See *Shell and Common Controls Versions* for a detailed discussion of shell and common controls versions, and how to use **DllGetVersion**.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in shlwapi.h.

DLLGETVERSIONPROC

This type definition is used to define a pointer to a **DllGetVersion** function. This pointer is used when calling the function dynamically by loading the library and getting the function's address. For more information, see the example in **DllGetVersion**.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: user_defined.

DoEnvironmentSubst

Parses an input string containing references to one or more environment variables, and replaces them with their current values.

```
DWORD DoEnvironmentSubst(
    LPCTSTR pszString,
    UINT cbSize
);
```

Parameters

pszString

Null-terminated string that contains references to one or more environment variables in the form:

```
%VariableName%
```

Case is ignored. Each %*VariableName*% string is replaced with the variable's current value. The replacement rules are the same as those used by the command interpreter. If the name is not found, the %*variableName*% string is left intact.

As environment variables can be added by the user or applications, the complete list is system-dependent. The following environment variables are standard with Microsoft Windows NT, and are available to both interactive applications and services:

COMPUTERNAME
NUMBER_OF_PROCESSORS
OS
PROCESSOR_ARCHITECTURE
PROCESSOR_IDENTIFIER
PROCESSOR_LEVEL
PROCESSOR_REVISION
PROGRAMFILES
SYSTEMDRIVE
SYSTEMROOT
USERPROFILE
WINDIR

The remainder are available only to interactive applications.

HOMEDRIVE
HOMEPATH
LOGONSERVER
USERDOMAIN
USERNAME

Only the **WINDIR** variable is available on Windows 95/98 systems.

cbSize

Size of the *pszString* buffer. It must be large enough to hold the values that will be returned.

Return Values

If the expanded values do not overfill the buffer, they replace the original *pszString* array. TRUE is returned in the LOWORD, and the length of *pszString* is returned in the HIWORD. If the buffer is too small, *pszString* is not modified. FALSE is returned in the LOWORD and *cbSize* in the HIWORD.

Remarks

This function is retained for backward compatibility with Windows 3.1. Use **ExpandEnvironmentStrings**, instead.

Because the string that is returned in *pszString* normally will be longer than the input string, make sure that the buffer is large enough.

The environment variables that correspond to file-system folders can be mapped to an equivalent CSIDL value and obtained with **SHGetFolderLocation**. CSIDLs are more reliable than variable names and should be used if at all possible.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

DragAcceptFiles

Registers whether a window accepts dropped files.

```
VOID DragAcceptFiles(
    HWND hWnd,
    BOOL fAccept
);
```

Parameters

hWnd
Identifier of the window that is registering whether it will accept dropped files.

fAccept
Value that indicates if the window identified by the *hWnd* parameter accepts dropped files. This value is TRUE to accept dropped files, or FALSE to discontinue accepting dropped files.

Return Values

No return value.

Remarks

An application that calls **DragAcceptFiles** with the *fAccept* parameter set to TRUE has identified itself as able to process the **WM_DROPFILES** message from File Manager.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.

Windows CE: Unsupported.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

DragFinish

Releases memory that the system allocated for use in transferring file names to the application.

```
VOID DragFinish(
  HDROP hDrop
);
```

Parameters

hDrop

Identifier of the structure that describes dropped files. This handle is retrieved from the *wParam* parameter of the **WM_DROPFILES** message.

Return Values

No return value.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

DragQueryFile

Retrieves the file names of dropped files that have resulted from a successful drag-and-drop operation.

```
UINT DragQueryFile(
    HDROP hDrop,
    UINT iFile,
    LPTSTR lpszFile,
    UINT cch
);
```

Parameters

hDrop
Identifier of the structure containing the file names of the dropped files.

iFile
Index of the file to query. If the value of the *iFile* parameter is 0xFFFFFFFF, **DragQueryFile** returns a count of the files dropped. If the value of the *iFile* parameter is between zero and the total number of files dropped, **DragQueryFile** copies the file name with the corresponding value to the buffer pointed to by the *lpszFile* parameter.

lpszFile
Address of a buffer to receive the file name of a dropped file when the function returns. This file name is a null-terminated string. If this parameter is NULL, **DragQueryFile** returns the required size, in characters, of the buffer.

cch
Size, in characters, of the *lpszFile* buffer.

Return Values

When the function copies a file name to the buffer, the return value is a count of the characters copied, not including the terminating null character.

If the index value is 0xFFFFFFFF, the return value is a count of the dropped files.

If the index value is between zero and the total number of dropped files and the *lpszFile* buffer address is NULL, the return value is the required size, in characters, of the buffer, not including the terminating null character.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

See Also

DragQueryPoint

DragQueryPoint

Retrieves the position of the mouse pointer at the time a file was dropped during a drag-and-drop operation.

```
BOOL DragQueryPoint(
    HDROP hDrop,
    LPPOINT lppt
);
```

Parameters

hDrop

Identifier of the structure that describes the dropped file.

lppt

Address of a **POINT** structure that the function fills with the coordinates of the mouse pointer at the time the file was dropped.

Return Values

Returns nonzero if the drop occurred in the client area of the window, or zero if the drop did not occur in the client area of the window.

Remarks

The window for which coordinates are returned is the window that received the **WM_DROPFILES** message.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

See Also

DragQueryFile

FindEnvironmentString

Looks up the specified environment variable and returns a pointer to its value.

```
LPCTSTR FindEnvironmentString(
    LPCTSTR pszEnvVar,
);
```

Parameters

pszEnvVar

Null-terminated string with the environment variable of interest. Case is ignored. Because environment variables can be added by the user or applications, the complete list is system-dependent. The following environment variables are standard with Microsoft Windows NT, and are available to both interactive applications and services:

COMPUTERNAME
NUMBER_OF_PROCESSORS
OS
PROCESSOR_ARCHITECTURE
PROCESSOR_IDENTIFIER
PROCESSOR_LEVEL
PROCESSOR_REVISION
PROGRAMFILES
SYSTEMDRIVE
SYSTEMROOT
USERPROFILE
WINDIR

The remainder are available only to interactive applications.

HOMEDRIVE
HOMEPATH
LOGONSERVER
USERDOMAIN
USERNAME

Only the **WINDIR** variable is available on Windows 95/98 systems.

Return Values

Returns the value of the environment variable, or returns NULL if the variable is not in the environment.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shellapi.h.

FindExecutable

Retrieves the name of and handle to the executable (.exe) file associated with the specified file name.

```
HINSTANCE FindExecutable(
    LPCTSTR lpFile,
    LPCTSTR lpDirectory,
    LPTSTR lpResult
);
```

Parameters

lpFile

Address of a null-terminated string specifying a file name. This can be a document or an executable file.

lpDirectory

Address of a null-terminated string specifying the default directory.

lpResult

Address of a buffer to receive the file name when the function returns. This file name is a null-terminated string specifying the executable file started when an “open” by association is run on the file specified in the *lpFile* parameter.

Return Values

Returns a value greater than 32 if successful, or a value less than or equal to 32 otherwise.

The following table lists the possible error values:

0	The system is out of memory or resources.
31	There is no association for the specified file type.
ERROR_BAD_FORMAT	The .exe file is invalid (non-Win32 .exe or error in .exe image).
ERROR_FILE_NOT_FOUND	The specified file was not found.
ERROR_PATH_NOT_FOUND	The specified path was not found.

Remarks

When **FindExecutable** returns, the *lpResult* parameter might contain the path to the DDE (Dynamic Data Exchange) server started if a server does not respond to a request to initiate a DDE conversation with the DDE client application.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.

Header: Declared in shellapi.h.
Import Library: shell32.lib.

See Also

ShellExecute

GetMenuContextHelpId

Retrieves the Help context identifier associated with the specified menu.

```
DWORD GetMenuContextHelpId(
    HMENU hmenu
);
```

Parameters

hmenu

Handle to the menu for which the Help context identifier is to be retrieved.

Return Values

Returns the Help context identifier if the menu has one, or zero otherwise.

See Also

SetMenuContextHelpId

Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in winuser.h.
Import Library: user32.lib.

GetWindowContextHelpId

Retrieves the Help context identifier, if any, associated with the specified window.

```
DWORD GetWindowContextHelpId(
    HWND hwnd
);
```

Parameters

hwnd

Handle to the window for which the Help context identifier is to be retrieved.

Return Values

Returns the Help context identifier if the window has one, or zero otherwise.

! Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in winuser.h.
Import Library: user32.lib.

+ See Also

SetWindowContextHelpId

InetIsOffline

Determines whether or not the system is connected to the Internet.

```
BOOL InetIsOffline(
  DWORD dwFlags,
);
```

Parameters

dwFlags

Input flags for the function. This must be set to zero.

Return Value

Returns TRUE if the local system in not connected currently to the Internet. Returns FALSE if the local system is connected to the Internet or if no attempt has been made yet to connect to the Internet.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in intshcut.h.

MIMEAssociationDialog

Runs the unregistered MIME content type dialog box.

```
HRESULT MIMEAssociationDialog(
  HWND     hwndParent,
  DWORD    dwInFlags,
  LPCSTR   pcszFile,
  LPCSTR   pcszMIMEContentType,
  LPSTR    pszAppBuf,
  UINT     ucAppBufLen,
);
```

Parameters

hwndParent

Handle to the parent window of any posted child windows.

dwInFlags

Bit flag value that specifies if an association is to be registered. The bit flag is the value MIMEASSOCDLG_FL_REGISTER_ASSOC (0x0001). If this bit is set, the selected application is registered as the handler for the given MIME type. If this bit is clear, no association is registered.

An application is registered only if this flag is set and the user indicates that a persistent association is to be made.

Registration is impossible if the string at *pcszFile* does not contain an extension.

pcszFile

Address of a null-terminated string that contains the name of the target file. This file must conform to the content type described by the *pcszMIMEContentType* parameter.

pcszMIMEContentType

Address of a null-terminated string that contains the unregistered content type.

pszAppBuf

Address of a buffer that receives the path of the application specified by the user.

ucAppBufLen

Size of *pszAppBuf*, in characters.

Return Value

Returns one of the following values:

Returns S_OK if the content type was successfully associated with the extension. In this case, the extension is associated as the default for the content type, and *pszAppBuf* points to the string that contains the path of the specified application. The function returns S_FALSE if nothing was registered. Otherwise, the return value will be one of the following:

E_ABORT	The user canceled the operation.
E_FLAGS	The flag combination passed in *dwInFlags* is invalid.
E_OUTOFMEMORY	There was insufficient memory available to complete the operation.
E_POINTER	One of the input pointers is invalid.

Remarks

This function does not validate the syntax of the input content type string at *pcszMIMEContentType*. A successful return value does not indicate that the specified MIME content type is valid.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in intshcut.h.

RegisterDialogClasses

Registers any nonstandard window classes required by a screen saver's configuration dialog box.

```
BOOL RegisterDialogClasses(
  HANDLE hInst
);
```

Parameters

hInst

Identifier of an instance of the module registering the window classes.

Return Values

Returns nonzero if successful, or zero otherwise.

To get extended error information, call **GetLastError**.

Remarks

The **RegisterDialogClasses** function should not be exported. It is called by routines defined in the Scrnsave.lib file.

If a screen saver does not register any special window classes for the configuration dialog box, the **RegisterDialogClasses** function must return TRUE.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in scrnsave.h.
Import Library: scrnsave.lib.

See Also

ScreenSaverConfigureDialog

ScreenSaverConfigureDialog

Receives messages sent to a screen saver's configuration dialog box. A screen saver that allows user configuration must define this function.

```
BOOL ScreenSaverConfigureDialog(
    HWND hDlg,
    UINT message,
    WPARAM wParam,
    LPARAM lParam
);
```

Parameters

hDlg

Identifier of the configuration dialog box.

message

Message that was sent to the screen saver's configuration dialog box.

wParam

Additional message-specific information.

lParam

Additional message-specific information.

Return Values

If the function successfully processes the message, it should return TRUE. If not, it should return FALSE, except in response to a **WM_INITDIALOG** message. In response to a **WM_INITDIALOG** message, **ScreenSaverConfigureDialog** should return FALSE if it calls the **SetFocus** function to set the keyboard focus to one of the controls in the dialog box. Otherwise, the function should return TRUE, in which case the system sets the keyboard focus to the first control in the dialog box that can be given the focus.

Remarks

The dialog-box template for the configuration dialog box must have the DLG_SCRNSAVECONFIGURE identifier.

The dialog-box procedure is used only if the application specifies the default window class (WC_DIALOG) for the dialog box. The application uses the default class if no explicit class is given in the dialog-box template. Although the dialog-box procedure is similar to a window procedure, it must not call the **DefWindowProc** function to process unwanted messages. Unwanted messages are processed internally by the default dialog-box procedure.

The **ScreenSaverConfigureDialog** function must be exported by including it in the **EXPORTS** statement in the application's module-definition (.def) file.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in scrnsave.h.
Import Library: scrnsave.lib.

See Also

RegisterDialogClasses

ScreenSaverProc

Receives messages sent to the specified screen-saver window.

```
LONG ScreenSaverProc(
    HWND hWnd,
    UINT message,
    WPARAM wParam,
    LPARAM lParam
);
```

Parameters

hWnd
Identifier of the window.

message
Message sent to the screen-saver window.

wParam
Additional message-specific information.

lParam
Additional message-specific information.

Return Value

The return value is the result of the message processing and depends on the message sent.

Remarks

A screen saver's **ScreenSaverProc** window procedure should use the **DefScreenSaverProc** function instead of the **DefWindowProc** function to provide default message processing. The **DefScreenSaverProc** function passes any messages that do not affect screen-saver operations to **DefWindowProc**.

The **ScreenSaverProc** function must be exported by including it in the **EXPORTS** statement in the application's module-definition (.def) file.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in scrnsave.h.
Import Library: scrnsave.lib.

SetMenuContextHelpId

Associates a Help context identifier with a menu.

```
BOOL SetMenuContextHelpId(
    HMENU hmenu,
    DWORD dwContextHelpId
);
```

Parameters

hmenu

Handle to the menu with which to associate the Help context identifier.

dwContextHelpId

Help context identifier.

Return Values

Returns nonzero if successful, or zero otherwise.

To get extended error information, call **GetLastError**.

Remarks

All items in the menu share this identifier. Help context identifiers cannot be attached to individual menu items.

! Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in winuser.h.
Import Library: user32.lib.

See Also

GetMenuContextHelpId

SetWindowContextHelpId

Associates a Help context identifier with the specified window.

```
BOOL SetWindowContextHelpId(
    HWND hwnd,
    DWORD dwContextHelpId
);
```

Parameters

hwnd

Handle to the window with which to associate the Help context identifier.

dwContextHelpId

Help context identifier.

Return Values

Returns nonzero if successful, or zero otherwise.

To get extended error information, call **GetLastError**.

Remarks

If a child window does not have a Help context identifier, it inherits the identifier of its parent window. Likewise, if an owned window does not have a Help context identifier, it inherits the identifier of its owner window. This inheritance of Help context identifiers allows an application to set just one identifier for a dialog box and all of its controls.

Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in winuser.h.
Import Library: user32.lib.

See Also

GetWindowContextHelpId

SHAddToRecentDocs

Adds a document to the shell's list of recently used documents, or clears all documents from the list.

```
VOID SHAddToRecentDocs(
    UINT uFlags,
    LPCVOID pv
);
```

Parameters

uFlags

[in] Flag that indicates the meaning of the *pv* parameter. This flag can be one of the following values:

SHARD_PATH
The *pv* parameter points to a NULL-terminated string with the path and filename of the object.

SHARD_PIDL
The *pv* parameter points to an **ITEMIDLIST** structure (PIDL) that identifies the document's file object. PIDLs that identify nonfile objects are not allowed.

pv

[in] A pointer to either a NULL terminated string with the path and file name of the document, or a PIDL that identifies the document's file object. Set this parameter to NULL, to clear all documents from the list.

Remarks

In addition to updating its list of recent documents, the shell adds a shortcut to the users Recent directory (CSIDL_RECENT) and the **Start** menu's **Documents** submenu.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in shlobj.h.
Import Library: shell32.lib.

+ See Also

SHGetFolderLocation, **SHGetFolderPath**

SHAppBarMessage

Sends an appbar message to the system.

```
UINT SHAppBarMessage(
    DWORD dwMessage,
    PAPPBARDATA pData
);
```

Parameters

dwMessage

Appbar message value to send. This parameter can be one of the following values:

ABM_ACTIVATE	Notifies the system that an appbar has been activated.
ABM_GETAUTOHIDEBAR	Retrieves the handle to the autohide appbar associated with a particular edge of the screen.
ABM_GETSTATE	Retrieves the autohide and always-on-top states of the Windows taskbar.
ABM_GETTASKBARPOS	Retrieves the bounding rectangle of the Windows taskbar.
ABM_NEW	Registers a new appbar and specifies the message identifier that the system should use to send notification messages to the appbar.
ABM_QUERYPOS	Requests a size and screen position for an appbar.
ABM_REMOVE	Unregisters an appbar, removing the bar from the system's internal list.
ABM_SETAUTOHIDEBAR	Registers or unregisters an autohide appbar for an edge of the screen.
ABM_SETPOS	Sets the size and screen position of an appbar.
ABM_WINDOWPOSCHANGED	Notifies the system when an appbar's position has changed.

pData

Address of an **APPBARDATA** structure. The content of the structure depends on the value set in the *dwMessage* parameter.

Return Values

Returns a message-dependent value. For more information, see the Microsoft Platform SDK documentation for the appbar message sent.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

SHBindToParent

This function takes the fully qualified PIDL of a namespace object, and returns a specified interface pointer on the parent object.

```
HRESULT SHBindToParent(
  LPCITEMIDLIST pidl,
  REFIID riid,
  VOID **ppv,
  LPCITEMIDLIST *ppidlLast
);
```

Parameters

pidl
[in] The item's PIDL.

riid
[in] The REFIID of one of the interfaces exposed by the item's parent object.

ppv
[out] A pointer to the interface specified by *riid*. You must release the object when you are finished.

ppidlLast
[out] The item's PIDL relative to the parent folder. This PIDL can be used with many of the methods supported by the parent folder's interfaces. If you set *ppidlLast* to NULL, the PIDL will not be returned.

Return Values

Returns S_OK if successful, an OLE-defined error value otherwise.

Example

The following code fragment uses **SHBindToParent** to get the display name from an item's PIDL. The **StrRetToBuf** function is used to convert the **STRRET** structure returned by **IShellFolder::GetDisplayNameOf** into a string:

```
IShellFolder *psfParent; //A pointer to the parent folder
                         // object's IShellFolder interface
LPITEMIDLIST pidlItem = NULL; //the item's PIDL
LPITEMIDLIST pidlRelative = NULL; //the item's PIDL
                                  // relative to the
```

```
                                        // parent folder
STRRET str; //the display name's STRRET structure
TCHAR szDisplayName[MAX_PATH]; //the display name's string

HRESULT hres = SHBindToParent(pidlItem, IID_IShellFolder,
&psfParent, &pidlRelative);
if(SUCCEEDED(hres))
{
   psfParent->GetDisplayNameOf(pidlRelative, SHGDN_NORMAL,
&str);
   psfParent->Release();
   StrRetToBuf(&str, pidlItem, szDisplayName,
ARRAYSIZE(szDisplayName));
}
```

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Windows CE: Unsupported.
Header: Declared in shlobj.h.
Import Library: shell32.lib.

SHBrowseForFolder

Displays a dialog box that enables the user to select a shell folder.

```
LPITEMIDLIST SHBrowseForFolder(
   LPBROWSEINFO lpbi
);
```

Parameters

lpbi

[in] Pointer to a **BROWSEINFO** structure that contains information used to display the dialog box.

Return Values

Returns a pointer to an **ITEMIDLIST** structure (PIDL) that specifies the location of the selected folder relative to the root of the namespace. If the user chooses the **Cancel** button in the dialog box, the return value is NULL.

Remarks

You must initialize COM with **CoInitialize** or **OLEInitialize** prior to calling **SHBrowseForFolder**.

The calling application is responsible for freeing the returned PIDL with the shell allocator's **IMalloc::Free method**. To get a handle to the shell allocator's **IMalloc** interface, call **SHGetMalloc**. See **The Shell Namespace** for further discussion of the shell allocator and PIDLs.

There are two styles of dialog box available. The older style is displayed by default, and is not resizable. To specify a new-style dialog box set the BIF_USENEWUI flag in the **ulFlags** member of the **BROWSEINFO** structure. It has a number of additional features, including: drag-and-drop capability within the dialog box, reordering, context menus, new folders, delete, and other context menu commands. Initially, it is larger than the old dialog box, but can be resized by the user.

If you implement a callback function, you will receive a handle to the dialog box. One use of this window handle is to modify the layout or contents of the dialog box. Because it is not resizable, modifying the old-style dialog box is relatively straightforward. Modifying the new-style dialog box is much more difficult, and not recommended. It not only has a different size and layout than the old style, its dimensions and the positions of its controls change every time it is resized by the user.

If the BIF_RETURNONLYFSDIRS flag is set in the **ulFlags** member of the **BROWSEINFO** structure, the OK button will remain enabled for "\\server" items, as well as "\\server\share" and directory items. However, if the user selects a "\\server" item, passing the PIDL returned by the dialog box to **SHGetPathFromIDList** will fail.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.
Import Library: shell32.lib.

+ See Also

Open and Save Common Dialog Boxes

SHChangeNotify

Notifies the system of an event that an application has performed. An application should use this function if it performs an action that might affect the shell.

```
VOID SHChangeNotify(
    LONG wEventId,
    UINT uFlags,
    LPCVOID dwItem1,
    LPCVOID dwItem2
);
```

Parameters

wEventId

Describes the event that has occurred. Typically, only one event is specified at a time. If more than one event is specified, the values contained in the *dwItem1* and *dwItem2* parameters must be the same, respectively, for all specified events. This parameter can be one or more of the following values:

SHCNE_ALLEVENTS	All events have occurred.
SHCNE_ASSOCCHANGED	A file-type association has changed. SHCNF_IDLIST must be specified in the *uFlags* parameter. *dwItem1* and *dwItem2* are not used and must be NULL.
SHCNE_ATTRIBUTES	The attributes of an item or folder have changed. SHCNF_IDLIST or SHCNF_PATH must be specified in *uFlags*. *dwItem1* contains the item or folder that has changed. *DwItem2* is not used and should be NULL.
SHCNE_CREATE	A nonfolder item has been created. SHCNF_IDLIST or SHCNF_PATH must be specified in *uFlags*. *dwItem1* contains the item that was created. *dwItem2* is not used and should be NULL.
SHCNE_DELETE	A nonfolder item has been deleted. SHCNF_IDLIST or SHCNF_PATH must be specified in *uFlags*. *dwItem1* contains the item that was deleted. *dwItem2* is not used and should be NULL.
SHCNE_DRIVEADD	A drive has been added. SHCNF_IDLIST or SHCNF_PATH must be specified in *uFlags*. *dwItem1* contains the root of the drive that was added. *dwItem2* is not used and should be NULL.
SHCNE_DRIVEADDGUI	A drive has been added and the shell should create a new window for the drive. SHCNF_IDLIST or SHCNF_PATH must be specified in *uFlags*. *dwItem1* contains the root of the drive that was added. *DwItem2* is not used and should be NULL.

(continued)

(continued)

SHCNE_DRIVEREMOVED	A drive has been removed. SHCNF_IDLIST or SHCNF_PATH must be specified in *uFlags*. *dwItem1* contains the root of the drive that was removed. *dwItem2* is not used and should be NULL.
SHCNE_EXTENDED_EVENT	Not used currently.
SHCNE_FREESPACE	The amount of free space on a drive has changed. SHCNF_IDLIST or SHCNF_PATH must be specified in *uFlags*. *DwItem1* contains the root of the drive on which the free space changed. *dwItem2* is not used and should be NULL.
SHCNE_MEDIAINSERTED	Storage media has been inserted into a drive. SHCNF_IDLIST or SHCNF_PATH must be specified in *uFlags*. *dwItem1* contains the root of the drive that contains the new media. *dwItem2* is not used and should be NULL.
SHCNE_MEDIAREMOVED	Storage media has been removed from a drive. SHCNF_IDLIST or SHCNF_PATH must be specified in *uFlags*. *dwItem1* contains the root of the drive from which the media was removed. *dwItem2* is not used and should be NULL.
SHCNE_MKDIR	A folder has been created. SHCNF_IDLIST or SHCNF_PATH must be specified in *uFlags*. *dwItem1* contains the folder that was created. *dwItem2* is not used and should be NULL.
SHCNE_NETSHARE	A folder on the local computer is being shared via the network. SHCNF_IDLIST or SHCNF_PATH must be specified in *uFlags*. *DwItem1* contains the folder that is being shared. *DwItem2* is not used and should be NULL.
SHCNE_NETUNSHARE	A folder on the local computer is no longer being shared via the network. SHCNF_IDLIST or SHCNF_PATH must be specified in *uFlags*. *dwItem1* contains the folder that is no longer being shared. *dwItem2* is not used and should be NULL.
SHCNE_RENAMEFOLDER	The name of a folder has changed. SHCNF_IDLIST or SHCNF_PATH must be specified in *uFlags*. *dwItem1* contains the previous PIDL or name of the folder. *dwItem2* contains the new PIDL or name of the folder.

SHCNE_RENAMEITEM
The name of a nonfolder item has changed. SHCNF_IDLIST or SHCNF_PATH must be specified in *uFlags*. *dwItem1* contains the previous PIDL or name of the item. *dwItem2* contains the new PIDL or name of the item.

SHCNE_RMDIR
A folder has been removed. SHCNF_IDLIST or SHCNF_PATH must be specified in *uFlags*. *dwItem1* contains the folder that was removed. *dwItem2* is not used and should be NULL.

SHCNE_SERVERDISCONNECT
The computer has disconnected from a server. SHCNF_IDLIST or SHCNF_PATH must be specified in *uFlags*. *dwItem1* contains the server from which the computer was disconnected. *dwItem2* is not used and should be NULL.

SHCNE_UPDATEDIR
The contents of an existing folder have changed, but the folder still exists and has not been renamed. SHCNF_IDLIST or SHCNF_PATH must be specified in *uFlags*. *DwItem1* contains the folder that has changed. *DwItem2* is not used and should be NULL. If a folder has been created, deleted, or renamed, use SHCNE_MKDIR, SHCNE_RMDIR, or SHCNE_RENAMEFOLDER, respectively, instead.

SHCNE_UPDATEIMAGE
An image in the system image list has changed. SHCNF_DWORD must be specified in *uFlags*. *dwItem1* contains the index in the system image list that has changed. *dwItem2* is not used and should be NULL.

SHCNE_UPDATEITEM
An existing nonfolder item has changed, but the item still exists and has not been renamed. SHCNF_IDLIST or SHCNF_PATH must be specified in *uFlags*. *dwItem1* contains the item that has changed. *dwItem2* is not used and should be NULL. If a nonfolder item has been created, deleted, or renamed, use SHCNE_CREATE, SHCNE_DELETE, or SHCNE_RENAMEITEM, respectively, instead.

The following values specify combinations of other events:

SHCNE_DISKEVENTS
Specifies a combination of all of the disk event identifiers.

SHCNE_GLOBALEVENT
Specifies a combination of all of the global event identifiers.

The following value modifies other event values and cannot be used alone:

SHCNE_INTERRUPT	The specified event occurred as a result of a system interrupt.

uFlags

Flags that indicate the meaning of the *dwItem1* and *dwItem2* parameters. The *uFlags* parameter must be one of the following values:

SHCNF_DWORD	The *dwItem1* and *dwItem2* parameters are DWORD values.
SHCNF_IDLIST	*dwItem1* and *dwItem2* are the addresses of **ITEMIDLIST** structures that represent the item(s) affected by the change. Each **ITEMIDLIST** must be relative to the desktop folder.
SHCNF_PATH	*dwItem1* and *dwItem2* are the addresses of NULL-terminated strings that contain the full path names of the items affected by the change.
SHCNF_PRINTER	*dwItem1* and *dwItem2* are the addresses of NULL-terminated strings that represent the friendly names of the printer(s) affected by the change.

The following flags modify other data-type flags and cannot be used by themselves:

SHCNF_FLUSH	The function should not return until the notification has been delivered to all affected components.
SHCNF_FLUSHNOWAIT	The function should begin delivering notifications to all affected components, but should return as soon as the notification process has begun.

dwItem1

First event-dependent value.

dwItem2

Second event-dependent value.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.
Import Library: shell32.lib.

SHCreateDirectoryEx

Creates a new file-system folder.

```
int SHCreateDirectoryEx(
    HWND hwnd,
    LPCSTR pszPath,
    SECURITY_ATTRIBUTES *psa
);
```

Parameters

hwnd

[in] Handle to a parent window. This parameter can be set to NULL if no user interface will be displayed.

pszPath

[in] Pointer to a string with the fully qualified path of the directory.

psa

[in] Pointer to a **SECURITY_ATTRIBUTES** structure with the directory's security attribute. Set this parameter to NULL if no security attributes need to be set.

Return Values

Returns ERROR_SUCCESS if successful. If the operation fails, one of the following error codes can be returned:

Error code	Description
ERROR_BAD_PATHNAME	The *pszPath* parameter was set to a relative path.
ERROR_FILENAME_EXCED_RANGE	The path pointed to by *pszPath* is too long.
ERROR_FILE_EXISTS	The directory exists.
ERROR_ALREADY_EXISTS	The directory exists.
ERROR_CANCELLED	The user canceled the operation.

Remarks

This function creates a file-system folder whose fully qualified path is given by *pszPath*. If one or more of the intermediate folders do not exist, they will be created as well. **SHCreateDirectoryEx** also verifies that the files will be visible. If not:

- If *hwnd* is set to a valid window handle, a message box is displayed warning users that they might not be able to access the files. If users choose not to proceed, the function returns ERROR_CANCELLED.
- If *hwnd* is set to NULL, no user interface is displayed and the function returns ERROR_CANCELLED.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.
Import Library: shell32.lib.

SHCreateProcessAsUser

Creates a new user-mode process and its primary thread to run a specified executable (.exe) file.

```
BOOL SHCreateProcessAsUser(
    PSHCREATEPROCESSINFOW pscpi
);
```

Parameters

pscpi

[in] Pointer to an **SHCREATEPROCESSINFOW** structure with information on how to create the process.

Return Values

Returns TRUE if successful, or FALSE if not. To get extended error information, call **GetLastError**.

Remarks

This function is similar to **ShellExecuteEx** with **runas** as the verb. However, **SHCreateProcessAsUser** creates a process that runs in the security context of the user represented by the **hUserToken** member of the structure pointed to by *pscpi*. The structure's **lpProcessInformation** member can be used to return a **PROCESS_INFORMATION** structure with information on the new process.

The **runas** verb must be supported by the executable file's file class. The .exe file class supports **runas**. Use the **AssocQueryString** function to check whether or not **runas** is supported by other file classes. The following code fragment illustrates the syntax:

```
AssocQueryString(0, ASSOCSTR_COMMAND, pszFile,
TEXT("runas"), NULL, &cchVerb)
```

For a discussion of how to use the shell to launch applications, see *Launching Applications*.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.
Import Library: shell32.lib.

See Also

CreateProcess, **CreateProcessAsUser**

Shell_NotifyIcon

Sends a message to the taskbar's status area.

```
BOOL Shell_NotifyIcon(
    DWORD dwMessage,
    PNOTIFYICONDATA pnid
);
```

Parameters

dwMessage

[in] A parameter that specifies the action to be taken. It can have one of the following values:

NIM_ADD

Add an icon to the status area. The **hWnd** and **uID** fields of the **NOTIFYICONDATA** structure pointed to by *pnid* will be used to identify the icon in future calls to **Shell_NotifyIcon**.

NIM_DELETE

Delete an icon from the status area. Use the **hWnd** and **uID** fields of the **NOTIFYICONDATA** structure pointed to by *pnid* to identify the icon to be deleted.

NIM_MODIFY

Modify an icon in the status area. Use the **hWnd** and **uID** fields of the **NOTIFYICONDATA** structure pointed to by *pnid* to identify the icon to be modified.

NIM_SETFOCUS

Version 5.0. Return focus to the taskbar notification area. Taskbar icons should use this message when they have completed their UI operation. For example, if the taskbar icon displays a context menu, but the user presses **Escape** to cancel it, NIM_SETFOCUS should be used to return focus to the taskbar notification area.

NIM_SETVERSION

Version 5.0. Instructs the taskbar to behave according to the version number specified in the **uVersion** member of the structure pointed to by *pnid*. This message allows you to specify whether you want the version 5.0 behavior found on Windows 2000 system, or the behavior found with earlier shell versions. See the Remarks for details.

pnid

[in] Pointer to a **NOTIFYICONDATA** structure. The content of the structure depends on the value of *dwMessage*.

Return Values

Returns TRUE if successful, or FALSE otherwise. If *dwMessage* is set to NIM_SETVERSION, the function will true if the version was successfully changed, or FALSE if the requested version is not supported.

Remarks

Version 5.0 of the shell, found on Windows 2000, handles **Shell_NotifyIcon** mouse and keyboard events differently than earlier shell versions, found on Windows NT 4.0, Windows 95, and Windows 98. The differences are:

- If a user requests a notify icon's context menu with the keyboard, the version 5.0 shell sends the associated application a **WM_CONTEXTMENU** message. Earlier versions send **WM_RBUTTONDOWN** and **WM_RBUTTONUP** messages.
- If a user selects a notify icon with the keyboard and activates it with the space bar or ENTER key, the version 5.0 shell sends the associated application a NIN_KEYSELECT notification. Earlier versions send **WM_RBUTTONDOWN** and **WM_RBUTTONUP** messages.
- If a user selects a notify icon with the mouse and activates it with the ENTER key, the version 5.0 shell sends the associated application a NIN_SELECT notification. Earlier versions send **WM_RBUTTONDOWN** and **WM_RBUTTONUP** messages.

You can select which way the shell should behave by calling **Shell_NotifyIcon** with *dwMessage* set to NIM_SETVERSION. Set the **uVersion** member of the **NOTIFYICONDATA** structure to indicate whether you want version 5.0 or pre-version 5.0 behavior.

Note The messages discussed above are not conventional Windows messages. They are sent as the *lParam* value of the application-defined message that is specified when the icon is added with NIM_ADD.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

ShellAbout

Displays a **ShellAbout** dialog box.

```
int ShellAbout (
    HWND hWnd,
    LPCTSTR szApp,
    LPCTSTR szOtherStuff,
    HICON hIcon
);
```

Parameters

hWnd

Window handle to a parent window. This parameter can be NULL.

szApp

Displays text in the title bar of the **ShellAbout** dialog box and on the first line of the dialog box after the text "Microsoft". If the text contains a separator (#) dividing it into two parts, the function displays the first part in the title bar and the second part on the first line after the text "Microsoft".

szOtherStuff

Displays text in the dialog box after the version and copyright information.

hIcon

Icon that the function displays in the dialog box. If this parameter is NULL, the function displays the Microsoft Windows or Windows NT icon.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

Note that the **ShellAbout** function dialog box uses text and a default icon that are specific to either Windows or Windows NT.

An example of a **ShellAbout** dialog box can be seen by selecting the **About Program Manager** command in Program Manager.

Requirements

Windows NT/2000: Requires Windows NT 3.5 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

ShellExecute

Performs an operation on a specified file.

```
HINSTANCE ShellExecute(
    HWND hwnd,
    LPCTSTR lpVerb,
    LPCTSTR lpFile,
    LPCTSTR lpParameters,
    LPCTSTR lpDirectory,
    INT nShowCmd
);
```

Parameters

hwnd

Handle to a parent window. This window receives any message boxes that an application produces. For example, an application can report an error by producing a message box.

lpVerb

A string, referred to as a *verb*, that specifies the action to be performed. The set of available verbs depends on the particular file or folder. It includes the commands listed in the context menu and the registry. See *Extending Context Menus* for further discussion of context menus. The following verbs are usually valid:

Verb	Description
edit	Opens an editor. If *lpFile* is not a document file, the function will fail.
explore	The function explores the folder specified by *lpFile*.
open	The function opens the file specified by the *lpFile* parameter. The file can be an executable file or a document file. It also can be a folder.
print	The function prints the document file specified by *lpFile*. If *lpFile* is not a document file, the function will fail.
properties	Displays the file or folder's properties.

If you set this paramater to NULL:

- For systems prior to Windows 2000, the "open" verb is used if available. If not, the *default verb* is used.
- For Windows 2000 and later systems, "open" is used if available. If not, the default verb is used. If neither verb is available, the system uses the first verb listed in the registry.

lpFile

Address of a null-terminated string that specifies the file to open or print or the folder to open or explore. The function can open an executable file or a document file. The function can print a document file.

lpParameters

If the *lpFile* parameter specifies an executable file, *lpParameters* is an address to a null-terminated string that specifies the parameters to be passed to the application.The format of this string is determined by the verb that is to be invoked. If *lpFile* specifies a document file, *lpParameters* should be NULL.

lpDirectory

Address of a null-terminated string that specifies the default directory.

nShowCmd

Flags that specify how an application is to be displayed when it is opened. If *lpFile* specifies a document file, the flag is passed to the associated application. It is up to the application to decide how to handle it.

SW_HIDE	Hides the window and activates another window.
SW_MAXIMIZE	Maximizes the specified window.
SW_MINIMIZE	Minimizes the specified window and activates the next top-level window in the z-order.
SW_RESTORE	Activates and displays the window. If the window is minimized or maximized, Windows restores it to its original size and position. An application should specify this flag when restoring a minimized window.
SW_SHOW	Activates the window and displays it in its current size and position.
SW_SHOWDEFAULT	Sets the show state based on the SW_ flag specified in the **STARTUPINFO** structure passed to the **CreateProcess** function by the program that started the application. An application should call **ShowWindow** with this flag to set the initial show state of its main window.
SW_SHOWMAXIMIZED	Activates the window and displays it as a maximized window.
SW_SHOWMINIMIZED	Activates the window and displays it as a minimized window.
SW_SHOWMINNOACTIVE	Displays the window as a minimized window. The active window remains active.
SW_SHOWNA	Displays the window in its current state. The active window remains active.
SW_SHOWNOACTIVATE	Displays a window in its most recent size and position. The active window remains active.
SW_SHOWNORMAL	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position. An application should specify this flag when displaying the window for the first time.

Return Values

Returns a value greater than 32 if successful, or an error value that is less than or equal to 32 otherwise. The following table lists the error values. The return value is cast as an HINSTANCE for backward compatibility with 16-bit Microsoft Windows applications. It is not a true HINSTANCE, however. The only thing that can be done with the returned HINSTANCE is to cast it to an integer and compare it with the value 32 or one of the error codes below:

0	The operating system is out of memory or resources.
ERROR_BAD_FORMAT	The .exe file is invalid (non-Win32 .exe or error in .exe image).
ERROR_FILE_NOT_FOUND	The specified file was not found.
ERROR_PATH_NOT_FOUND	The specified path was not found.
SE_ERR_ACCESSDENIED	The operating system denied access to the specified file.
SE_ERR_ASSOCINCOMPLETE	The file name association is incomplete or invalid.
SE_ERR_DDEBUSY	The DDE transaction could not be completed because other DDE transactions were being processed.
SE_ERR_DDEFAIL	The DDE transaction failed.
SE_ERR_DDETIMEOUT	The DDE transaction could not be completed because the request timed out.
SE_ERR_DLLNOTFOUND	The specified dynamic-link library was not found.
SE_ERR_FNF	The specified file was not found.
SE_ERR_NOASSOC	There is no application associated with the given file name extension. This error will also be returned if you attempt to print a file that is not printable.
SE_ERR_OOM	There was not enough memory to complete the operation.
SE_ERR_PNF	The specified path was not found.
SE_ERR_SHARE	A sharing violation occurred.

Remarks

This method allows you to execute any commands in a folder's context menu or stored in the registry.

To open a folder, use either of the following calls:

```
ShellExecute(handle, NULL, path_to_folder, NULL, NULL,SW_SHOWNORMAL);
```

or

```
ShellExecute(handle, "open", path_to_folder, NULL, NULL, SW_SHOWNORMAL);
```

To explore a folder, use:

```
ShellExecute(handle, "explore", path_to_folder, NULL,
NULL, SW_SHOWNORMAL);
```

To launch the shell's **Find** utility for a directory, use:

```
ShellExecute(handle, "find", path_to_folder, NULL, NULL, 0);
```

If *lpOperation* is NULL, the function opens the file specified by *lpFile*. If *lpOperation* is "open" or "explore", the function will attempt to open or explore the folder.

With multiple monitors, if you specify the window handle and set *lpOperation* to "Properties", any windows created by **ShellExecute** might not appear in the correct position.

To obtain information about the application that is launched as a result of calling **ShellExecute**, use **ShellExecuteEx**.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

+ See Also

IShellExecuteHook

ShellExecuteEx

Performs an action on a file.

```
BOOL ShellExecuteEx(
    LPSHELLEXECUTEINFO lpExecInfo
);
```

Parameters

lpExecInfo

Address of a **SHELLEXECUTEINFO** structure that contains and receives information about the application being executed.

Return Values

Returns TRUE if successful, or FALSE otherwise. Call **GetLastError** for error information.

Remarks

With multiple monitors, if you specify an hwnd and set the **lpVerb** member of *lpExecInfo* to "Properties" , any windows created by **ShellExecuteEx** may not appear in the correct position.

If the function succeeds, it sets the **hInstApp** member of the **SHELLEXECUTEINFO** structure to a value greater than 32. If the function fails, **hInstApp** is set to the **SE_ERR_XXX** error value that best indicates the cause of the failure. Although **hInstApp** is declared as an HINSTANCE for compatibility with 16-bit Microsoft Windows applications, it is not a true HINSTANCE. It can be cast only to an integer and compared to either 32 or the SE_ERR_XXX error codes.

Note The SE_ERR_XXX error values are provided for compatibility with **ShellExecute**. To retrieve more accurate error information, use **GetLastError**. It may return one of the following values:

ERROR_ACCESS_DENIED	Access to the specified file is denied.
ERROR_CANCELLED	The function prompted the user for additional information, but the user canceled the request.
ERROR_DDE_FAIL	The DDE transaction failed.
ERROR_DLL_NOT_FOUND	One of the library files necessary to run the application cannot be found.
ERROR_FILE_NOT_FOUND	The specified file was not found.
ERROR_NO_ASSOCIATION	There is no application associated with the given file-name extension.
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory to perform the specified action.
ERROR_PATH_NOT_FOUND	The specified path was not found.
ERROR_SHARING_VIOLATION	A sharing violation occurred.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

See Also

IShellExecuteHook

SHEmptyRecycleBin

Empties the Recycle Bin on the specified drive.

```
HRESULT SHEmptyRecycleBin(
    HWND hwnd,
    LPCTSTR pszRootPath,
    DWORD dwFlags
);
```

Parameters

hwnd

Handle to the parent window of any dialog boxes that might be displayed during the operation. This parameter can be NULL.

pszRootPath

Address of a NULL-terminated string that contains the path of the root drive on which the Recycle Bin is located. This parameter can contain the address of a string formatted with the drive, folder, and subfolder names (c:\windows\system . . .). It also can contain an empty string or NULL. If this value is an empty string or NULL, all Recycle Bins on all drives will be emptied.

dwFlags

One or more of the following values:

SHERB_NOCONFIRMATION	No dialog box confirming the deletion of the objects will be displayed.
SHERB_NOPROGRESSUI	No dialog box indicating the progress will be displayed.
SHERB_NOSOUND	No sound will be played when the operation is complete.

Return Values

Returns S_OK if successful, or an OLE-defined error value otherwise.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with
Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with
Internet Explorer 4.0 or later).

Windows CE: Unsupported.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

See Also

SHQueryRecycleBin

SHFileOperation

Copies, moves, renames, or deletes a file-system object.

```
int SHFileOperation(
    LPSHFILEOPSTRUCT lpFileOp
);
```

Parameters

lpFileOp
[in] Address of an **SHFILEOPSTRUCT** structure that contains information this function needs to carry out the specified operation.

Return Values

Returns zero if successful, or nonzero otherwise.

Remarks

You should use fully qualified path names with this function. Using it with relative path names is not thread-safe.

With two exceptions, you cannot use **SHFileOperation** to move special folders from a local drive to a remote computer by specifying a network path. The exceptions are the MyDocs and MyPics folders (CSIDL_PERSONAL and CSIDL_MYPICTURES, respectively).

When used to delete a file, **SHFileOperation** will attempt to place the deleted file in the Recycle Bin. If you want to delete a file and guarantee that it will *not* be placed in the Recycle Bin, use **DeleteFile**.

If a copy callback handler is exposed and registered, **SHFileOperation** will call it unless you set a flag such as FOF_NOCONFIRMATION in the **fFlags** member of the structure pointed to by *lpFileOp*. See **ICopyHook::CopyCallback** for details on implementing copy callback handlers.

File deletion is recursive unless you set the FOF_NORECURSION flag in *lpFileOp*.

With Microsoft Windows NT versions 5.0 and later, it is possible to *connect* an HTML file with a folder containing related files, such as GIF images or style sheets. If file connection is enabled, when you move or copy the HTML file, all of the files in the folder will be moved or copied, too. Conversely, if you move the folder with the related files, the HTML file also is moved.

The HTML file *must* have a .htm or .html extension. You create the connection to the related files by placing them in a subfolder of the folder containing the HTML file. The folder must have the name of the HTML file followed by " files". For example, if the HTML file is MyFile.htm, the folder should be named "MyFile files". Any files you place in the "MyFile files" subfolder will be connected to MyFile.htm.

File connection is enabled by default. It can be disabled by adding a REG_DWORD value, NoFileFolderConnection, to the HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer registry key. Setting NoFileFolderConnection to 1 disables file connection. If the value is set to zero or is missing, file connection is enabled.

To move only specified files from a group of connected files, set the FOF_NO_CONNECTED_ELEMENTS flag in the **fFlags** member of the structure pointed to by *lpFileOp*.

Note that the use of a folder with a name like "MyFile files" to define a connection might not be valid for localized versions of Windows NT. The term "files" might need be replaced by the equivalent word in the local language.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

SHFreeNameMappings

Frees a file-name mapping object that was retrieved by the **SHFileOperation** function.

```
VOID SHFreeNameMappings(;
    HANDLE hNameMappings
);
```

Parameters

hNameMappings
Handle to the file-name mapping object to be freed.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

SHGetDataFromIDList

Retrieves extended property data from a relative identifier list.

```
HRESULT SHGetDataFromIDList(
    IShellFolder *psf,
    LPCITEMIDLIST pidl,
    int nFormat,
    PVOID pv,
    int cb
);
```

Parameters

psf

Address of the parent **IShellFolder** interface. This must be the immediate parent of the **ITEMIDLIST** referenced by the **pidl** parameter.

pidl

Address of a simple **ITEMIDLIST** structure that identifies the object relative to the folder specified in **psf**.

nFormat

Format in which the data is being requested. This must be one of the following formats:

Format	Description
SHGDFIL_DESCRIPTIONID	**Version 4.71.** Format used for network resources. The *pv* parameter is the address of an **SHDESCRIPTIONID** structure.
SHGDFIL_FINDDATA	Format used for file-system objects. The *pv* parameter is the address of a **WIN32_FIND_DATA** structure.
SHGDFIL_NETRESOURCE	Format used for network resources. The *pv* parameter is the address of a **NETRESOURCE** structure.

pv

Address of a buffer that receives the requested data. The format of this buffer is determined by *nFormat*.

If *nFormat* is SHGDFIL_NETRESOURCE, there are two possible cases. If the buffer is large enough, the net resource's string information (fields for the network name, local name, provider, and comments) will be placed into the buffer. If the buffer is not large enough, only the net resource structure will be placed into the buffer and the string information pointers will be NULL.

cb

Size of the buffer at *pv*, in bytes.

Return Values

Returns NOERROR if successful, or E_INVALIDARG otherwise.

Remarks

E_INVALIDARG is returned if the *psf*, *pidl*, *pv*, or *cb* parameter does not match the *nFormat* parameter, or if *nFormat* is not one of the specific SHGDFIL_ values shown above.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.
Import Library: shell32.lib.

SHGetDesktopFolder

Retrieves the **IShellFolder** interface for the desktop folder, which is the root of the shell's namespace.

```
HRESULT SHGetDesktopFolder(
    IShellFolder **ppshf
);
```

Parameters

ppshf

Address that receives an **IShellFolder** interface pointer for the desktop folder. The calling application is responsible for eventually freeing the interface by calling its **Release** method.

Return Values

Returns NOERROR if successful, or an OLE-defined error result otherwise.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.
Import Library: shell32.lib.

SHGetDiskFreeSpace

Retrieves disk-space information for a disk volume.

```
BOOL SHGetDiskFreeSpace(
    LPCTSTR pszVolume,
    ULARGE_INTEGER *pqwFreeCaller,
    ULARGE_INTEGER *pqwTot,
    ULARGE_INTEGER *pqwFree
);
```

Parameters

pszVolume
A NULL-terminated string that specifies the volume for which size information will be retrieved. This can be a drive letter, UNC name, or the path of a folder. You cannot use NULL to represent the current drive.

pqwFreeCaller
Address of a **ULARGE_INTEGER** value that receives the number of bytes available to the caller on the volume. If the operating system implements per-user quotas, this value can be less than the total number of free bytes on the volume.

pqwTot
Address of a **ULARGE_INTEGER** value that receives the total size of the volume, in bytes.

pqwFree
Address of a **ULARGE_INTEGER** value that receives the number of bytes of free space on the volume.

Return Values

Returns nonzero if successful, or zero otherwise.

Remarks

SHGetDiskFreeSpace is nearly identical to the **GetDiskFreeSpaceEx** API. **SHGetDiskFreeSpace** will load the proper module, obtain the address of **GetDiskFreeSpaceEx**, and then call it. This approach prevents the caller from having to perform these operations themselves. See the operating system restrictions in the **GetDiskFreeSpaceEx** reference for more information.

This function is implemented in shell versions 4.71 and later. In order to maintain backward compatibility with previous shell versions, this function should not be used explicitly. Instead, the **LoadLibrary** and **GetProcAddress** functions should be used to obtain the function address.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with
Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with
Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

See Also

GetDiskFreeSpace

SHGetFileInfo

Retrieves information about an object in the file system, such as a file, folder, directory, or drive root.

```
DWORD_PTR SHGetFileInfo(
    LPCTSTR pszPath,
    DWORD dwFileAttributes,
    SHFILEINFO *psfi,
    UINT cbFileInfo,
    UINT uFlags
);
```

Parameters

pszPath

[in] Address of a buffer that contains the path and file name. Both absolute and relative paths are valid.

If the *uFlags* parameter includes the SHGFI_PIDL flag, this parameter must be the address of an **ITEMIDLIST** (PIDL) structure that contains the list of item identifiers that uniquely identifies the file within the shell's namespace. The PIDL must be a fully qualified PIDL. Relative PIDLs are not allowed.

If the *uFlags* parameter includes the SHGFI_USEFILEATTRIBUTES flag, this parameter does not have to be a valid file name. The function will proceed as if the file exists with the specified name and with the file attributes passed in the *dwFileAttributes* parameter. This allows you to obtain information about a file type by passing just the extension for *pszPath* and passing FILE_ATTRIBUTE_NORMAL in *dwFileAttributes*.

This string can use either short (the 8.3 form) or long file names.

dwFileAttributes

[in] Combination of one or more file attribute flags (FILE_ATTRIBUTE_ values). If *uFlags* does not include the SHGFI_USEFILEATTRIBUTES flag, this parameter is ignored.

psfi

[out] Address of a **SHFILEINFO** structure to receive the file information.

cbFileInfo

[in] Size, in bytes, of the **SHFILEINFO** structure pointed to by the *psfi* parameter.

uFlags

[in] Flags that specify the file information to retrieve. This parameter can be a combination of the following values:

SHGFI_ATTR_SPECIFIED

Modifies SHGFI_ATTRIBUTES. Indicates that the **dwAttributes** member of the **SHFILEINFO** structure at *psfi* contains the specific attributes that are desired. These attributes will be passed to **IShellFolder::GetAttributesOf**. If this flag is not specified, 0xFFFFFFFF will be passed to **GetAttributesOf**, requesting all attributes. This flag cannot be specified with the SHGFI_ICON flag.

SHGFI_ATTRIBUTES

Retrieve the item attributes. The attributes are copied to the **dwAttributes** member of the structure specified in the *psfi* parameter. These are the same attributes that are obtained from **IShellFolder::GetAttributesOf**.

SHGFI_DISPLAYNAME

Retrieve the display name for the file. The name is copied to the **szDisplayName** member of the structure specified in *psfi*. The returned display name uses the long file name, if there is one, instead of the 8.3 form of the file name.

SHGFI_EXETYPE

Retrieve the type of the executable file if *pszPath* identifies an executable file. The information is packed into the return value. This flag cannot be specified with any other flags.

SHGFI_ICON

Retrieve the handle to the icon that represents the file and the index of the icon within the system image list. The handle is copied to the **hIcon** member of the structure specified by *psfi*, and the index is copied to the **iIcon** member. The return value is the handle to the system image list.

SHGFI_ICONLOCATION
Retrieve the name of the file that contains the icon representing the file. The name is copied to the **szDisplayName** member of the structure specified in *psfi.*

SHGFI_LARGEICON
Modify SHGFI_ICON, causing the function to retrieve the file's large icon. The SHGFI_ICON flag also must be set.

SHGFI_LINKOVERLAY
Modify SHGFI_ICON, causing the function to add the link overlay to the file's icon. The SHGFI_ICON flag also must be set.

SHGFI_OPENICON
Modify SHGFI_ICON, causing the function to retrieve the file's open icon. A container object displays an open icon to indicate that the container is open. The SHGFI_ICON flag also must be set.

SHGFI_PIDL
Indicate that *pszPath* is the address of an ITEMIDLIST structure rather than a path name.

SHGFI_SELECTED
Modify SHGFI_ICON, causing the function to blend the file's icon with the system highlight color. The SHGFI_ICON flag must also be set.

SHGFI_SHELLICONSIZE
Modify SHGFI_ICON, causing the function to retrieve a shell-sized icon. If this flag is not specified, the function sizes the icon according to the system metric values. The SHGFI_ICON flag must also be set.

SHGFI_SMALLICON
Modify SHGFI_ICON, causing the function to retrieve the file's small icon. The SHGFI_ICON flag also must be set.

SHGFI_SYSICONINDEX
Retrieve the index of a system image list icon. If successful, the index is copied to the **iIcon** member of *psfi.* The return value is a handle to the system image list. Only those images whose indexes are successfully copied to **iIcon** are valid. Attempting to access other images in the system image list will result in undefined behavior.

SHGFI_TYPENAME
Retrieve the string that describes the file's type. The string is copied to the **szTypeName** member of the structure specified in *psfi.*

SHGFI_USEFILEATTRIBUTES
Indicates that the function should not attempt to access the file specified by *pszPath.* Instead, it should act as if the file specified by *pszPath* exists with the file attributes passed in *dwFileAttributes.* This flag cannot be combined with the SHGFI_ATTRIBUTES, SHGFI_EXETYPE, or SHGFI_PIDL flags.

Return Values

Returns a value whose meaning depends on the *uFlags* parameter. If *uFlags* contains the SHGFI_EXETYPE flag, the return value specifies the type of the executable file. It will be one of the following values:

Value	Executable file type
0	Nonexecutable file or an error condition
LOWORD = MZ and HIWORD = 0	MS-DOS .exe, .com, or .bat file
LOWORD = NE or PE and HIWORD = 3.0, 3.5, or 4.0	Windows application
LOWORD = PE and HIWORD = 0	Win32 console application

If *uFlags* contains SHGFI_SYSICONINDEX, the return value is a handle to an image list that contains the large icon images. If SHGFI_SMALLICON is included with

SHGFI_SYSICONINDEX, the return value is the handle to an image list that contains the small icon images.

If *uFlags* does not contain SHGFI_EXETYPE or SHGFI_SYSICONINDEX, the return value is nonzero if successful, or zero otherwise.

Remarks

If **SHGetFileInfo** returns an icon handle in the **hIcon** member of the **SHFILEINFO** structure pointed to by *psfi*, you are responsible for freeing it with **DestroyIcon** when you no longer need it.

Note Once you have a handle to a system image list, you can use the Image **List API** to manipulate it like any other image list. Because system image lists are created on a per-process basis, you should treat them as read-only objects. Writing to a system image list might overwrite or delete one of the system images, making it unavailable or incorrect for the remainder of the process.

You must initialize COM with **CoInitialize** or **OLEInitialize** prior to calling **SHGetFileInfo**.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

SHGetFolderLocation

Retrieves the path of a folder as an **ITEMIDLIST** structure.

```
HRESULT SHGetFolderLocation(
    HWND hwndOwner,
    int nFolder,
    HANDLE hToken,
    DWORD dwReserved,
    LPITEMIDLIST *ppidl
);
```

Parameters

hwndOwner

[in] Handle to the owner window. This parameter is typically set to NULL. If it is not NULL, and a dial-up connection needs to be made to access the folder, a UI prompt will appear in this window.

nFolder

[in] A CSIDL value that identifies the folder to be located. The folders associated with the CSIDLs will not necessarily exist on a particular system.

hToken

[in] Token that can be used to represent a particular user. It is usually set to NULL, but it may be needed when there are multiple users for those folders that are treated as belonging to a single user. The caller is responsible for correct impersonation when *hToken* is non-NULL. It must have appropriate security privileges for the particular user, and the user's registry hive must be mounted currently.

dwReserved

Reserved. Must be set to zero.

ppidl

[out] Address of a pointer to an item identifier list structure specifying the folder's location relative to the root of the namespace (the desktop). The *ppidl* parameter will be set to NULL on failure. The calling application is responsible for freeing this pointer with the shell's **IMalloc** interface (see **SHGetMalloc**).

Return Values

S_FALSE	The CSIDL in *nFolder* is valid, but the folder does not exist.
E_INVALIDARG	The CSIDL in *nFolder* is not valid.
S_OK	Success.

A standard OLE-defined error result also can be returned.

Remarks

The **SHGetFolderLocation**, **SHGetFolderPath**, **SHGetSpecialFolderLocation**, and **SHGetSpecialFolderPath** functions are the preferred ways to obtain handles to folders. Functions such as **ExpandEnvironmentStrings** that use the environment variable names directly, in the form %*VariableName*%, might not be reliable.

This function is a superset of **SHGetSpecialFolderLocation**, included with earlier versions of the shell. It is implemented also in a redistributable DLL, SHFolder.dll, that simulates many of the new shell folders on older platforms such as Windows 95 and Windows NT 4.0. This DLL always calls the current platform's version of this function. If that fails, it will try to simulate the appropriate behavior. Only some CSIDLs are supported, including:

CSIDL_APPDATA	c:\..\User\Application Data
CSIDL_LOCAL_APPDATA	\..\User\Local Application Data (nonroaming)
CSIDL_MYPICTURES	My Documents\My Pictures
CSIDL_PERSONAL	My Documents

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Windows CE: Unsupported.
Header: Declared in shlobj.h.
Import Library: shell32.lib.

SHGetFolderPath

Takes the CSIDL of a folder and returns the path name.

```
HRESULT SHGetFolderPath(
    HWND hwndOwner,
    int nFolder,
    HANDLE hToken,
    DWORD dwFlags,
    LPTSTR pszPath
);
```

Parameters

hwndOwner

Handle to an owner window. This parameter is set typically to NULL. If it is not NULL, and a dial-up connection needs to be made to access the folder, a UI prompt will appear in this window.

nFolder

A CSIDL value that identifies the folder whose path is to be retrieved. Only real folders are valid. If a virtual folder is specified, this function will fail. You can force creation of a folder with **SHGetFolderPath** by combining the folder's CSIDL with CSIDL_FLAG_CREATE.

hToken

An access token that can be used to represent a particular user. For systems earlier than Windows 2000, it should be set to NULL. For later systems, *hToken* is usually set to NULL. However, you might need to assign a value to *hToken* for those folders that can have multiple users but are treated as belonging to a single user. The most commonly used folder of this type is My Documents.

The caller is responsible for correct impersonation when *hToken* is non-NULL. It must have appropriate security privileges for the particular user, including TOKEN_QUERY and TOKEN_IMPERSONATE, and the user's registry hive must be mounted currently. See *Access Control* for further discussion of access control issues.

dwFlags

Flags to specify which path is to be returned. It is used for cases where the folder associated with a CSIDL might be moved or renamed by the user.

Flag	Description
SHGFP_TYPE_CURRENT	Return the folder's current path.
SHGFP_TYPE_DEFAULT	Return the folder's default path.

pszPath

Buffer of length [MAX_PATH] to receive the path. If an error occurs or S_FALSE is returned, this string will be empty.

Return Values

Value	Description
S_FALSE	The CSIDL in *nFolder* is valid, but the folder does not exist.
E_INVALIDARG	The CSIDL in *nFolder* is not valid.
S_OK	Success.

A standard OLE-defined error result also can be returned.

Example

The following code fragment uses **SHGetFolderPath** to find or create a folder and then creates a file in it:

```
TCHAR szPath[MAX_PATH]
...
if(SUCCEEDED(SHGetFolderPath(NULL, CSIDL_PERSONAL|CSIDL_
FLAG_CREATE, NULL, 0, szPath))
{
    PathAppend(szPath, TEXT("New Doc.txt"));
    HANDLE hFile = CreateFile(szPath, ...);
}
```

Remarks

This function is a superset of **SHGetSpecialFolderPath**, included with earlier versions of the shell. It is implemented in a redistributable DLL, SHFolder.dll, that also simulates many of the new shell folders on older platforms such as Windows 95, Windows 98, and Windows NT 4.0. This DLL always calls the current platform's version of this function. If that fails, it will try to simulate the appropriate behavior. Only some CSIDLs are supported, including:

- **CSIDL_APPDATA**
- **CSIDL_COMMON_APPDATA**
- **CSIDL_COOKIES**
- **CSIDL_HISTORY**
- **CSIDL_INTERNET_CACHE**
- **CSIDL_LOCAL_APPDATA**
- **CSIDL_MYPICTURES**
- **CSIDL_PERSONAL**
- **CSIDL_SYSTEM**
- **CSIDL_WINDOWS**

! Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.
Import Library: ShFolder.Lib.

SHGetIconOverlayIndex

Returns the index of the overlay icon in the system image list.

```
int SHGetIconOverlayIndex(
    LPCTSTR pszIconPath,
    int iIconIndex
);
```

Parameters

pszIconPath

[in] Fully qualified path of the file that contains the icon.

iIconIndex

[in] Icon's index in the file pointed to by *pszIconPath*. To request a standard overlay icon, set *pszIconPath* to NULL, and *iIconIndex* to one of the following:

IDO_SHGIOI_LINK	The overlay icon that indicates a linked folder or file.
IDO_SHGIOI_SHARE	The overlay icon that indicates a shared folder.
IDO_SHGIOI_SLOWFILE	The overlay icon that indicates a slow file.

Return Values

Returns the index of the overlay icon in the system image list if successful, or –1 otherwise.

Remarks

Icon overlays are part of the system image list. They have two identifiers. One is a one-based overlay index that identifies the overlay relative to other overlays in the image list. The other is an image index that identifies the actual image. These two indexes are equivalent to the values that you assign to the *iOverlay* and *iImage* parameters, respectively, when you add an icon overlay to a private image list with **ImageList_SetOverlayImage**. **ShGetIconOverlayIndex** returns the overlay index. To convert an overlay index to its equivalent image index, call **INDEXTOOVERLAYMASK**.

Note Once the image has been loaded into the system image list during initialization, it cannot be changed. The file name and index specified by *pszIconPath* and *iIconIndex* are used only to identify the icon overlay. **ShGetIconOverlayIndex** cannot be used to modify the system image list.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.

Header: Declared in shlobj.h.
Import Library: shell32.lib.

See Also

IShellIconOverlay, **IShellIconOverlayIdentifier**

SHGetInstanceExplorer

Allows components that run in a Web browser (Iexplore.exe) or a nondefault Windows Explorer (Explorer.exe) process to hold a reference to the process. The components can use the reference to prevent the process from closing prematurely.

```
HRESULT SHGetInstanceExplorer(
    IUnknown **ppunk
);
```

Parameters

ppunk

[out] Address of an Explorer.exe or Iexplore.exe **IUnknown** interface pointer. If the function succeeds, **SHGetInstanceExplorer** increments the host's reference count before it returns. Components should not call **ppunk->AddRef**. Components should call **ppunk->Release** to release the reference when the host process is no longer needed. If **SHGetInstanceExplorer** fails, *ppunk* is set to NULL.

Return Values

Returns S_OK if successful, or an OLE error code otherwise.

Remarks

There are a number of components, such as shell extension handlers, that are implemented as DLLs and run in the process space of Windows Explorer (Explorer.exe) or Internet Explorer (Iexplore.exe). Normally, when the user closes the host process, the component is shut down immediately as well. Such an abrupt termination can create problems for some components. For example, if a component is using a background thread to download data or run user-interface functions, it might need additional time to shut itself down safely.

The **SHGetInstanceExplorer** function allows components that run in an Iexplorer.exe or a nondefault Explorer.exe process to hold a reference on the host process. **SHGetInstanceExplorer** increments the host's reference count and returns a pointer to its **IUnknown** interface. By holding that reference, a component can prevent the host process from closing prematurely. Once the component has completed all necessary processing, it should call **ppunk->Release** to release the host's reference and allow the process to die.

Note If **SHGetInstanceExplorer** is successful, the component must release the host's reference when it is no longer needed. Otherwise, all resources associated with the process will remain in memory. The **IUnknown** interface pointed to by *ppunk* can be used only to release this reference. Components cannot use **ppunk->QueryInterface** to request other interface pointers.

SHGetInstanceExplorer succeeds only if it is called from within an Explorer.exe or Iexplorer.exe process. It is used typically by components that run in the context of the Web browser (Iexplore.exe). However, it is useful also when Explorer.exe has been configured to run all folders in a second process. **SHGetInstanceExplorer** fails if the component is running in the default Explorer.exe process. There is no need to hold a reference to this process, as it is shut down only when the user logs out.

In general, components can be loaded by either the default Explorer.exe process or a secondary Explorer.exe or Iexplore.exe process. The component must be able thus to handle either the success or failure of **SHGetInstanceExplorer**.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.
Import Library: shell32.lib.

SHGetMalloc

Retrieves a pointer to the shell's **IMalloc** interface.

```
HRESULT SHGetMalloc(
    LPMALLOC *ppMalloc
);
```

Parameters

ppMalloc
Address of a pointer that receives the shell's **IMalloc** interface pointer.

Return Values

Returns NOERROR if successful, or E_FAIL otherwise.

Remarks

This interface must be used to free memory that was allocated by the shell or to allocate memory that will be freed by the shell. Applications also can use this interface to allocate and free their own memory.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.
Import Library: shell32.lib.

SHGetNewLinkInfo

Creates the proper name for a new shortcut. This function does not actually create a shortcut.

```
BOOL SHGetNewLinkInfo(
    LPCTSTR pszLinkTo,
    LPCTSTR pszDir,
    LPTSTR pszName,
    BOOL *pfMustCopy,
    UINT uFlags
);
```

Parameters

pszLinkTo
[in] Specifies the path and file name of the target of the shortcut. If *uFlags* does not contain the SHGNLI_PIDL value, this parameter is the address of a NULL-terminated string that contains the target. If *uFlags* contains the SHGNLI_PIDL value, this parameter is a **PIDL** that represents the target.

pszDir
[in] Address of a NULL-terminated string that contains the path of the folder that the shortcut would be created in.

pszName
[out] Address of a string that receives the NULL-terminated path and file name for the shortcut. This buffer is assumed to be at least MAX_PATH characters in size.

pfMustCopy
[out] Address of a **BOOL** value that receives a flag that indicates if the shortcut would be copied. When a shortcut to another shortcut is created, the shell simply copies the target shortcut and modifies that copied shortcut appropriately. This value receives a nonzero value if the target specified in *pszLinkTo* specifies a shortcut that will cause

the target shortcut to be copied. This value receives zero if the target does not specify a shortcut that would be copied.

uFlags

[in] Specifies the options for the function. This can be zero or a combination of the following values:

Value	Description
SHGNLI_NOUNIQUE	The function will not create a unique name within the destination folder. If this flag is not included, the function create the shortcut name and then determine if the name is unique in the destination folder. If a file with the same name exists within the destination folder, the shortcut name will be modified. This process is repeated until a unique name is found.
SHGNLI_PIDL	*pszLinkTo* is a PIDL that represents the target. If this flag is not included, *pszLinkTo* is the address of a string that contains the path and file names of the target.
SHGNLI_PREFIXNAME	The created name will be preceded by the string "Shortcut to ".

Return Values

Returns nonzero if successful, or zero otherwise.

Remarks

SHGetNewLinkInfo will determine the executable type of the target. If *pszLinkTo* specifies a DOS application, the ".PIF" extension will be used, otherwise the ".LNK" extension will be used for the shortcut.

SHGetNewLinkInfo will determine also if the destination file system supports long file names. If the destination file system supports long file names, then a long file name will be used for the shortcut name. If the destination file system does not support long file names, then the shortcut name will be returned in an 8.3 format.

This function is implemented in shell versions 4.71 and later. In order to maintain backward compatibility with previous shell versions, this function should not be used explicitly. Instead, the **LoadLibrary** and **GetProcAddress** functions should be used to obtain the function address.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

SHGetPathFromIDList

Converts an item identifier list to a file-system path.

```
BOOL SHGetPathFromIDList(
    LPCITEMIDLIST pidl,
    LPSTR pszPath
);
```

Parameters

pidl

Address of an item identifier list that specifies a file or directory location relative to the root of the namespace (the desktop).

pszPath

Address of a buffer to receive the file-system path. This buffer must be at least MAX_PATH characters in size.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

If the location specified by the *pidl* parameter is not part of the file system, this function will fail.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.
Import Library: shell32.lib.

SHGetSettings

Retrieves the current shell-option settings.

```
VOID SHGetSettings(
    LPSHELLFLAGSTATE lpsfs,
    DWORD dwMask
);
```

Parameters

lpsfs

Address of a **SHELLFLAGSTATE** structure that receives the shell-option settings.

dwMask

Set of flags that determine which members of *lpsfs* are being requested. This can be one or more of the following values:

SSF_DESKTOPHTML	The **fDesktopHTML** member is being requested.
SSF_DONTPRETTYPATH	The **fDontPrettyPath** member is being requested.
SSF_DOUBLECLICKINWEBVIEW	The **fDoubleClickInWebView** member is being requested.
SSF_HIDEICONS	The **fHideIcons** member is being requested.
SSF_MAPNETDRVBUTTON	The **fMapNetDrvBtn** member is being requested.
SSF_NOCONFIRMRECYCLE	The **fNoConfirmRecycle** member is being requested.
SSF_SHOWALLOBJECTS	The **fShowAllObjects** member is being requested.
SSF_SHOWATTRIBCOL	The **fShowAttribCol** member is being requested.
SSF_SHOWCOMPCOLOR	The **fShowCompColor** member is being requested.
SSF_SHOWEXTENSIONS	The **fShowExtensions** member is being requested.
SSF_SHOWINFOTIP	The **fShowInfoTip** member is being requested.
SSF_SHOWSYSFILES	The **fShowSysFiles** member is being requested.
SSF_WIN95CLASSIC	The **fWin95Classic** member is being requested.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

SHGetSpecialFolderLocation

Retrieves a pointer to the **ITEMIDLIST** structure of a special folder.

```
HRESULT SHGetSpecialFolderLocation(
    HWND hwndOwner,
    int nFolder,
    LPITEMIDLIST *ppidl
);
```

Parameters

hwndOwner
Handle to the owner window the client should specify if it displays a dialog box or message box.

nFolder
A CSIDL value that identifies the folder of interest.

ppidl
A pointer to the folder's item identifier list (PIDL) specifying the folder's location relative to the root of the namespace (the desktop). The calling application is responsible for freeing this pointer with the shell's **IMalloc** interface (see **SHGetMalloc**).

Return Values

Returns NOERROR if successful, or an OLE-defined error result otherwise.

Remarks

With Windows 2000, this function is superseded by **ShGetFolderLocation**.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.

Header: Declared in shlobj.h.
Import Library: shell32.lib.

See Also

SHGetSpecialFolderPath

SHGetSpecialFolderPath

Retrieves the path of a special folder, identified by its CSIDL.

```
BOOL SHGetSpecialFolderPath(
    HWND hwndOwner,
    LPTSTR lpszPath,
    int nFolder,
    BOOL fCreate
);
```

Parameters

hwndOwner
Handle to the owner window the client should specify if it displays a dialog box or message box.

lpszPath
Address of a character buffer that receives the drive and path of the specified folder. This buffer must be at least MAX_PATH characters in size.

nFolder
A CSIDL that identifies the folder of interest. If a virtual folder is specified, this function will fail.

fCreate
Indicates if the folder should be created if it does not already exist. If this value is nonzero, the folder will be created. If this value is zero, the folder will not be created.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

The Internet Explorer 4.0 Desktop Update must be installed for this function to be available.

With Windows 2000, this function is superseded by **ShGetFolderPath**. You can use this function on earlier systems by including the redistributable DLL, ShFolder.dll.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.
Import Library: shell32.lib.

SHInvokePrinterCommand

Executes a command on a printer object.

```
BOOL SHInvokePrinterCommand(
    HWND hwnd,
    UINT uAction,
    LPCTSTR lpBuf1,
    LPCTSTR lpBuf2,
    BOOL fModal
);
```

Parameters

hwnd

Handle of the window that will be used as the parent of any windows or dialog boxes that are created during the operation.

uAction

A value that determines the type of printer operation that will be performed. This can be one of the following values:

PRINTACTION_DOCUMENTDEFAULTS

Windows NT only. The default document properties for the printer specified by the name in *lpBuf1* will be displayed. *lpBuf2* is ignored.

PRINTACTION_NETINSTALL

The network printer specified by the name in *lpBuf1* will be installed. *lpBuf2* is ignored.

PRINTACTION_NETINSTALLLINK	A shortcut to the network printer specified by the name in *lpBuf1* will be created. *lpBuf2* specifies the drive and path of the folder in which the shortcut will be created. The network printer must have been installed already on the local machine.
PRINTACTION_OPEN	The printer specified by the name in *lpBuf1* will be opened. *lpBuf2* is ignored.
PRINTACTION_OPENNETPRN	The network printer specified by the name in *lpBuf1* will be opened. *lpBuf2* is ignored.
PRINTACTION_PROPERTIES	The properties for the printer specified by the name in *lpBuf1* will be displayed. *lpBuf2* can be either NULL or specify.
PRINTACTION_SERVERPROPERTIES	**Windows NT only.** The properties for the server of the printer specified by the name in *lpBuf1* will be displayed. *lpBuf2* is ignored.
PRINTACTION_TESTPAGE	A test page will be printed on the printer specified by the name in *lpBuf1*. *lpBuf2* is ignored.

lpBuf1

Address of a string that contains additional information for the printer command. The information contained in this parameter depends upon the value of *uAction*.

lpBuf2

Address of a string that contains additional information for the printer command. The information contained in this parameter depends upon the value of *uAction*.

fModal

A value that determines whether **SHInvokePrinterCommand** should return after initializing the command or wait until the command is completed. If this value is nonzero, **SHInvokePrinterCommand** will not return until the command is completed. If this value is zero, **SHInvokePrinterCommand** will return as soon as the command is initialized.

Return Values

Returns nonzero if successful, or zero otherwise.

Remarks

When a printer name is specified by *lpBuf1*, the name can either be the name of a local printer or the server and share name of a network printer. When specifying a network printer name, the name must be specified in the format of "\\<server>\<shared printer name>".

This function is implemented in shell versions 4.71 and later. In order to maintain backward compatibility with previous shell versions, this function should not be used explicitly. Instead, the **LoadLibrary** and **GetProcAddress** functions should be used to obtain the function address.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

SHLoadInProc

Creates an instance of the specified object class from within the context of the shell's process.

```
HRESULT SHLoadInProc(
    REFCLSID rclsid
);
```

Parameters

rclsid
CLSID of the object class to be created.

Return Values

Returns NOERROR if successful, or an OLE-defined error result otherwise.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.
Import Library: shell32.lib.

SHQueryRecycleBin

Retrieves the size of the Recycle Bin, and the number of items in it, on the specified drive.

```
HRESULT SHQueryRecycleBin(
    LPCTSTR pszRootPath,
    LPSHQUERYRBINFO pSHQueryRBInfo
);
```

Parameters

pszRootPath

Address of a NULL-terminated string to contain the path of the root drive on which the Recycle Bin is located. This parameter can contain the address of a string formatted with the drive, folder, and subfolder names (c:\windows\system . . .). It also can contain an empty string or NULL. If this value is an empty string or NULL, information is retrieved for all Recycle Bins on all drives.

pSHQueryRBInfo

Address of a **SHQUERYRBINFO** structure that receives the Recycle Bin information. The **cbSize** member of the structure must be set to the size of the structure before calling this API.

Return Values

Returns S_OK if successful, or an OLE-defined error value otherwise.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shellapi.h.
Import Library: shell32.lib.

See Also

SHEmptyRecycleBin

SoftwareUpdateMessageBox

Displays a standard message box that can be used to notify a user that an application has been updated.

```
DWORD SoftwareUpdateMessageBox(
    HWND hWnd,
    LPCWSTR szDistUnit,
    DWORD dwFlags,
    LPSOFTDISTINFO psdi
);
```

Parameters

hWnd

Handle to the parent window.

szDistUnit

String value containing the identifier for the code distribution unit. For ActiveX controls, *szDistUnit* is typically a globally unique identifier (GUID).

dwFlags

Reserved. Must be set to zero.

psdi

Address of a **SOFTDISTINFO** structure in which to store the update information. The **cbSize** member must be initialized to the sizeof(SOFTDISTINFO).

Return Values

Returns one of the following values:

IDABORT	An error occurred.
IDIGNORE	There is no pending software update.
IDNO	The user clicked the **Don't Update** button on the dialog box.
IDYES	The user clicked the **Update Now** or **About Update** button. The application should move to the HTML page referred to by the **szHREF** member of the structure pointed to by *psdi*.

Remarks

The preferable way to handle updates is to author a Channel Definition Format with an Open Software Description (OSD) vocabulary inside and make the shortcut OSD-aware. Refer to the Channel Definition Format documentation for details (Site Builder Network).

The **SoftwareMessageBox** function is intended to be used in the case where shell shortcut hooks do not work. One example is an application that was not installed on the start menu. If that application needs to do it's own software update check it should use this function.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

TranslateURL

Applies common translations to a given URL string, creating a new URL string.

```
HRESULT TranslateURL(
  LPCSTR pcszURL,
  DWORD dwInFlags,
  LPSTR *ppszTranslatedURL,
);
```

Parameters

pcszURL
Address of the URL string to be translated.

dwInFlags
Bit flags that specify how the URL string is to be translated. This value can be a combination of the following:

TRANSLATEURL_FL_GUESS_PROTOCOL
If the protocol scheme is not specified in the *pcszURL* parameter to **TranslateURL**, the system automatically chooses a scheme and adds it to the URL.

TRANSLATEURL_FL_USE_DEFAULT_PROTOCOL
If the protocol scheme is not specified in the *pcszURL* parameter to **TranslateURL**, the system adds the default protocol to the URL.

ppszTranslatedURL
Pointer variable that receives the pointer to the newly created, translated URL string, if any. The **ppszTranslatedURL* parameter is valid only if the function returns S_OK.

Return Value

Returns S_OK upon success, or S_FALSE if the URL did not require translation. If an error occurs, the function returns one of the following values:

E_FLAGS
The flag combination passed in *dwInFlags* is invalid.

E_OUTOFMEMORY
There was insufficient memory to complete the operation.

E_POINTER
One of the input pointers is invalid.

Remarks

This function does not validate the input URL string. A successful return value does not indicate that the URL strings are valid URLs.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in intshcut.h.

URLAssociationDialog

Invokes the unregistered URL protocol dialog box. This dialog box allows the user to select an application to associate with a previously unknown protocol.

```
HRESULT URLAssociationDialog(
    HWND hwndParent,
    DWORD dwInFlags,
    LPCSTR pcszFile,
    LPCSTR pcszURL,
    LPSTR pszAppBuf,
    UINT ucAppBufLen,
);
```

Parameters

hwndParent
Handle to the parent window.

dwInFlags
Bit flags that specify the behavior of the function. This value can be a combination of the following:

URLASSOCDLG_FL_REGISTER_ASSOC
Register the selected application as the handler for the protocol specified in *pcszURL*. The application is registered only if this flag is set and the user indicates that a persistent association is desired.

URLASSOCDLG_FL_USE_DEFAULT_NAME
Use the default file name (that is, "Internet Shortcut").

pcszFile
Address of a constant zero-terminated string that contains the file name to associate with the URL's protocol.

pcszURL
Address of a constant zero-terminated string that contains the URL with an unknown protocol.

pszAppBuf
Address of a buffer that receives the path of the application specified by the user.

ucAppBufLen
The size of *pszAppBuf*, in characters.

Return Value

Returns S_OK if the application is registered with the URL protocol, or S_FALSE if nothing is registered. For example, the function returns S_FALSE when the user elects to perform a one-time execution via the selected application.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in intshcut.h.

WinHelp

Starts Microsoft Windows Help (Winhelp.exe) and passes additional data indicating the nature of the Help requested by the application.

```
BOOL WinHelp(
    HWND hWndMain,
    LPCTSTR lpszHelp,
    UINT uCommand,
    DWORD dwData
);
```

Parameters

hWndMain
Handle of the window requesting Help. The **WinHelp** function uses this handle to keep track of which applications have requested Help. If the *uCommand* parameter specifies HELP_CONTEXTMENU or HELP_WM_HELP, *hWndMain* identifies the control requesting Help.

lpszHelp
Address of a null-terminated string containing the path, if necessary, and the name of the Help file that **WinHelp** is to display.

The file name may be followed by an angle bracket (>) and the name of a secondary window if the topic is to be displayed in a secondary window instead of in the primary window. You must define the name of the secondary window in the [WINDOWS] section of the Help project (.hpj) file.

uCommand

Type of Help requested. For a list of possible values and how they affect the value to place in the *dwData* parameter, see the Remarks section.

dwData

Additional data. The value used depends on the value of the *uCommand* parameter. For a list of possible *dwData* values, see the Remarks section.

Return Values

Returns nonzero if successful, or zero otherwise.

To get extended error information, call **GetLastError**.

Remarks

Before closing the window that requested Help, the application must call **WinHelp** with the *uCommand* parameter set to HELP_QUIT. Until all applications have done this, Windows Help will not terminate. Note that calling Windows Help with the HELP_QUIT command is not necessary if you used the HELP_CONTEXTPOPUP command to start Windows Help.

The following table shows the possible values for the *uCommand* parameter and the corresponding formats of the *dwData* parameter:

uCommand	Action	dwData
HELP_COMMAND	Executes a Help macro or macro string.	Address of a string that specifies the name of the Help macro(s) to run. If the string specifies multiple macro names, the names must be separated by semicolons. You must use the short form of the macro name for some macros, because Windows Help does not support the long name.
HELP_CONTENTS	Displays the topic specified by the Contents option in the [OPTIONS] section of the .hpj file. This command is for backward compatibility. New applications should provide a .cnt file and use the HELP_FINDER command.	Ignored, set to 0.

uCommand	Action	dwData
HELP_CONTEXT	Displays the topic identified by the specified context identifier defined in the [MAP] section of the .hpj file.	Unsigned long integer containing the context identifier for the topic.
HELP_CONTEXTMENU	Displays the Help menu for the selected window, then displays the topic for the selected control in a pop-up window.	Address of an array of double-word pairs. The first double word in each pair is a control identifier, and the second is a context number for a topic. The array must be terminated by a pair of zeros{0,0}.
HELP_CONTEXTPOPUP	Displays the topic identified by the specified context identifier defined in the [MAP] section of the .hpj file in a pop-up window.	Unsigned long integer containing the context identifier for a topic.
HELP_FINDER	Displays the **Help Topics** dialog box.	Ignored, set to 0.
HELP_FORCEFILE	Ensures that Windows Help is displaying the correct Help file. If the incorrect Help file is being displayed, Windows Help opens the correct one; otherwise, there is no action.	Ignored, set to 0.
HELP_HELPONHELP	Displays Help on how to use Windows Help, if the **WINHLP32.HLP** file is available.	Ignored, set to 0.
HELP_INDEX	Displays the topic specified by the Contents option in the [OPTIONS] section of the .hpj file. This command is for backward compatibility. New applications should use the HELP_FINDER command.	Ignored, set to 0.
HELP_KEY	Displays the topic in the keyword table that matches the specified keyword, if there is an exact match. If there is more than one match, displays the Index with the topics listed in the **Topics Found** list box.	Address of a keyword string. Multiple keywords must be separated by semicolons.

(continued)

(continued)

uCommand	Action	dwData
HELP_MULTIKEY	Displays the topic specified by a keyword in an alternative keyword table.	Address of a **MULTIKEYHELP** structure that specifies a table footnote character and a keyword.
HELP_PARTIALKEY	Displays the topic in the keyword table that matches the specified keyword, if there is an exact match. If there is more than one match, displays the **Topics Found** dialog box. To display the index without passing a keyword, you should use a pointer to an empty string.	Address of a keyword string. Multiple keywords must be separated by semicolons.
HELP_QUIT	Informs Windows Help that it is no longer needed. If no other applications have asked for Help, Windows closes Windows Help.	Ignored, set to 0.
HELP_SETCONTENTS	Specifies the Contents topic. Windows Help displays this topic when the user clicks the Contents button if the Help file does not have an associated .cnt file.	Unsigned long integer containing the context identifier for the Contents topic.
HELP_SETPOPUP_POS	Sets the position of the subsequent pop-up window.	Address of a **POINTS** structure. The pop-up window is positioned as if the mouse cursor were at the specified point when the pop-up window was invoked.
HELP_SETWINPOS	Displays the Help window, if it is minimized or in memory, and sets its size and position as specified.	Address of a **HELPWININFO** structure that specifies the size and position of either a primary or secondary Help window.
HELP_TCARD	Indicates that a command is for a training card instance of Windows Help. Combine this command with other commands using the bitwise OR operator.	Depends on the command with which this command is combined.

uCommand	Action	dwData
HELP_WM_HELP	Displays the topic for the control identified by the *hWndMain* parameter in a pop-up window.	Address of an array of double-word pairs. The first double word in each pair is a control identifier, and the second is a context identifier for a topic. The array must be terminated by a pair of zeros {0,0}.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in winuser.h.
Import Library: user32.lib.

+ See Also

HELPWININFO, **MULTIKEYHELP**

Shell Callback Functions

BrowseCallbackProc

Specifies an application-defined callback function used with the **SHBrowseForFolder** function. The browse dialog box calls this function to notify it about events.

```
int CALLBACK BrowseCallbackProc(
    HWND hwnd,
    UINT uMsg,
    LPARAM lParam,
    LPARAM lpData
);
```

Parameters

hwnd

Window handle to the browse dialog box. **BrowseCallbackProc** can use this handle to send the following messages to the dialog box:

BFFM_ENABLEOK

Enables or disables the browse dialog box's **OK** button. To enable the **OK** button, set the message's *lParam* to a nonzero value. To disable the **OK** button, set message's *lParam* to zero.

BFFM_SETSELECTION
: Selects the specified folder. To use a PIDL to specify the folder, set the message's *lParam* to the PIDL, and set *wParam* to FALSE. To specify the folder's path, set the message's *lParam* value to point to a NULL-terminated string with the path, and set *wParam* to TRUE.

BFFM_SETSTATUSTEXT
: Sets the status text. Set the message's *lParam* value to point to a NULL-terminated string with the desired text.

uMsg

Value identifying the event. This can be one of the following values:

BFFM_INITIALIZED

Indicates the browse dialog box has finished initializing. The *lParam* value is zero.

BFFM_SELCHANGED

Indicates the selection has changed. The *lParam* parameter points to the item identifier list for the newly selected item.

BFFM_VALIDATEFAILED

Version 4.71. Indicates the user typed an invalid name into the edit box of the browse dialog. The *lParam* parameter is the address of a character buffer that contains the invalid name. An application can use this message to inform the user that the name entered was not valid. Return zero to allow the dialog to be dismissed, or nonzero to keep the dialog displayed.

lParam

Value dependent upon the message contained in the *uMsg* parameter.

lpData

Application-defined value that was specified in the **lParam** member of the **BROWSEINFO** structure.

Return Values

Returns zero.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.
Import Library: user-defined.

+ See Also

SHBrowseForFolder

FMExtensionProc

Application-defined callback function called by File Manager to communicate with a File Manager extension.

```
LONG FMExtensionProc(
    HWND hwnd,
    WORD wMsg,
    LONG lParam
);
```

Parameters

hwnd

Window handle to File Manager. An extension should use this handle to specify the parent window for any dialog box or message box it must display; it also should use this handle to send query messages to File Manager.

wMsg

One of the following File Manager messages:

1 through 99	User selected an item from the extension-supplied menu. The value is the identifier of the selected menu item.
FMEVENT_HELPMENUITEM	User pressed F1 while selecting an extension menu or toolbar command item. File Manager wants the extension to call **WinHelp** appropriately for the command item.
FMEVENT_HELPSTRING	User selected an extension menu or toolbar command item. File Manager wants the extension to supply a Help string.
FMEVENT_INITMENU	User selected the extension's menu. The extension should initialize items in the menu.
FMEVENT_LOAD	File Manager is loading the extension DLL and prompts the DLL for information about the menu that the DLL supplies.
FMEVENT_SELCHANGE	Selection in the File Manager directory window or Search Results window has changed.
FMEVENT_TOOLBARLOAD	File Manager is creating the toolbar and prompts the extension DLL for information about any buttons the DLL adds to the toolbar.
FMEVENT_UNLOAD	File Manager is unloading the extension DLL.
FMEVENT_USER_REFRESH	User selected the **Refresh** command from the **Window** menu. The extension should update items in the menu, if necessary.

lParam

Message-specific value.

Return Values

Returns a value dependent upon the *wMsg* parameter message.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.
Import Library: user-defined.

UndeleteFile

Application-defined callback function called by File Manager when the user chooses the **Undelete** command from the **File** menu.

```
DWORD UndeleteFile(
    HWND hwndOwner,
    LPSTR lpszDir
);
```

Parameters

hwndOwner
Window handle to File Manager. An undelete dynamic-link library (DLL) should use this handle to specify the owner window for any dialog box or message box the DLL might display.

lpszDir
Address of a null-terminated string that contains the name of the initial directory.

Return Values

Returns one of the following values:

–1	An error occurred.
IDCANCEL	No file was undeleted.
IDOK	A file was undeleted. File Manager repaints its window.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.
Import Library: user-defined.

CHAPTER 9

Shell Structures

Shell Structures

APPBARDATA

Contains information about a system appbar message. This structure is used with the **SHAppBarMessage** function.

```
typedef struct _AppBarData {
    DWORD  cbSize;
    HWND   hWnd;
    UINT   uCallbackMessage;
    UINT   uEdge;
    RECT   rc;
    LPARAM lParam;
} APPBARDATA, *PAPPBARDATA;
```

Members

cbSize

Contains the size of the structure, in bytes.

hWnd

Contains the handle to the appbar window.

uCallbackMessage

Application-defined message identifier. The application uses the specified identifier for notification messages that it sends to the appbar identified by the **hWnd** member. This member is used when sending the **ABM_NEW** message.

uEdge

Value that specifies an edge of the screen. This member can be one of the following values:

ABE_BOTTOM	Bottom edge.
ABE_LEFT	Left edge.
ABE_RIGHT	Right edge.
ABE_TOP	Top edge.

This member is used when sending the **ABM_GETAUTOHIDEBAR**, **ABM_QUERYPOS**, **ABM_SETAUTOHIDEBAR**, and **ABM_SETPOS** messages.

rc

RECT structure to contain the bounding rectangle, in screen coordinates, of an appbar or the Windows taskbar. This member is used when sending the **ABM_GETTASKBARPOS**, **ABM_QUERYPOS**, and **ABM_SETPOS** messages.

lParam

Message-dependent value. This member is used with the **ABM_SETAUTOHIDEBAR** message.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.

BROWSEINFO

Contains parameters for the **SHBrowseForFolder** function and receives information about the folder selected by the user.

```
typedef struct _browseinfo {
    HWND hwndOwner;
    LPCITEMIDLIST pidlRoot;
    LPTSTR pszDisplayName;
    LPCTSTR lpszTitle;
    UINT ulFlags;
    BFFCALLBACK lpfn;
    LPARAM lParam;
    int iImage;
} BROWSEINFO, *PBROWSEINFO, *LPBROWSEINFO;
```

Members

hwndOwner

Handle to the owner window for the dialog box.

pidlRoot

Pointer to an **ITEMIDLIST** structure (PIDL) specifying the location of the root folder from which to start browsing. Only the specified folder and any subfolders that are beneath it in the namespace hierarchy will appear in the dialog box. This member can be NULL; in that case, the namespace root (the desktop folder) is used.

pszDisplayName

Address of a buffer to receive the display name of the folder selected by the user. The size of this buffer is assumed to be MAX_PATH bytes.

lpszTitle

Address of a null-terminated string that is displayed above the tree view control in the dialog box. This string can be used to specify instructions to the user.

ulFlags

Flags specifying the options for the dialog box. This member can include zero or a combination of the following values:

BIF_BROWSEFORCOMPUTER
Only return computers. If the user selects anything other than a computer, the **OK** button appears dimmed.

BIF_BROWSEFORPRINTER
Only return printers. If the user selects anything other than a printer, the **OK** button appears dimmed.

BIF_BROWSEINCLUDEFILES
Version 4.71. The browse dialog box will display files as well as folders.

BIF_BROWSEINCLUDEURLS
Version 5.0. The browse dialog box can display URLs. The BIF_USENEWUI and BIF_BROWSEINCLUDEFILES flags also must be set. If these three flags are not set, the browser dialog box will reject URLs. Even when these flags are set, the browse dialog box will only display URLs if the folder that contains the selected item supports them. When the folder's **IShellFolder::GetAttributesOf** method is called to request the selected item's attributes, the folder must set the SFGAO_FOLDER attribute flag. Otherwise, the browse dialog box will not display the URL.

BIF_DONTGOBELOWDOMAIN
Do not include network folders below the domain level in the dialog box's tree-view control.

BIF_EDITBOX
Version 4.71. Includes an edit control in the browse dialog box that allows the user to type the name of an item.

BIF_NEWDIALOGSTYLE
Version 5.0. Use the new user-interface. Setting this flag provides the user with a larger dialog box that can be resized. It has several new capabilities including: drag-and-drop capability within the dialog box, reordering, context menus, new folders, delete, and other context menu commands. To use this flag, you must call **OleInitialize** or **CoInitialize** before calling **SHBrowseForFolder**.

(continued)

(continued)

BIF_RETURNFSANCESTORS	Only return file-system ancestors. An ancestor is a subfolder that is beneath the root folder in the namespace hierarchy. If the user selects an ancestor of the root folder that is not part of the file system, the **OK** button appears dimmed.
BIF_RETURNONLYFSDIRS	Only return file-system directories. If the user selects folders that are not part of the file system, the **OK** button appears dimmed.
BIF_SHAREABLE	**Version 5.0.** The browse dialog box can display shareable resources on remote systems. It is intended for applications that want to expose remote shares on a local system. The BIF_USENEWUI flag must also be set.
BIF_STATUSTEXT	Include a status area in the dialog box. The callback function can set the status text by sending messages to the dialog box.
BIF_USENEWUI	**Version 5.0**. Use the new user interface including an edit box. This flag is equivalent to BIF_EDITBOX | BIF_NEWDIALOGSTYLE. To use BIF_USENEWUI, you must call **OleInitialize** or **CoInitialize** before calling **SHBrowseForFolder**.
BIF_VALIDATE	**Version 4.71.** If the user types an invalid name into the edit box, the browse dialog will call the application's **BrowseCallbackProc** with the **BFFM_VALIDATEFAILED** message. This flag is ignored if BIF_EDITBOX is not specified.

lpfn

Address of an application-defined function that the dialog box calls when an event occurs. For more information, see the **BrowseCallbackProc** function. This member can be NULL.

lParam

Application-defined value that the dialog box passes to the callback function, if one is specified.

iImage

Variable to receive the image associated with the selected folder. The image is specified as an index to the system image list.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.

Windows CE: Unsupported.
Header: Declared in shlobj.h.

CIDA

This structure is used with the **CFSTR_SHELLIDLIST** clipboard format to transfer the PIDLs of one or more shell namespace objects.

```
typedef struct _IDA {
    UINT cidl;
    UINT aoffset[1];
} CIDA, * LPIDA;
```

Members

cidl

Number PIDLs that are being transferred, not counting the parent folder.

aoffset

An array of offsets, relative to the beginning of this structure. The array contains **cidl**+1 elements. The first element of **aoffset** contains an offset to the fully qualified PIDL of a parent folder. If this PIDL is empty, the parent folder is the desktop. Each of the remaining elements of the array contains an offset to one of the PIDLs to be transferred. All of these PIDLs are relative to the PIDL of the parent folder.

Remarks

To use this structure to get a particular PIDL, add the PIDLs **aoffset** value to the address of the structure. The following two macros can be used to retrive PIDLs from the structure. The first retrieves the PIDL of the parent folder. The second retrieves a PIDL, specified by its zero-based index:

```
#define HIDA_GetPIDLFolder(pida) (LPCITEMIDLIST)
(((LPBYTE)pida)+(pida)->aoffset[0])
#define HIDA_GetPIDLItem(pida, i) (LPCITEMIDLIST)
(((LPBYTE)pida)+(pida)->aoffset[i+1])
```

The value that is returned by these macros is a pointer to the PIDL's **ITEMIDLIST** structure. Since these structures vary in length, you must determine the end of the structure by parsing it. See *The Shell Namespace* for further discussion of PIDLs and the **ITEMIDLIST** structure

! Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.

Windows CE: Unsupported.
Header: Declared in shlobj.h.

CMINVOKECOMMANDINFO

Contains information needed by **IContextMenu::InvokeCommand** to invoke a context menu command.

```
typedef struct _CMInvokeCommandInfo{
    DWORD cbSize;
    DWORD fMask;
    HWND hwnd;
    LPCSTR lpVerb;
    LPCSTR lpParameters;
    LPCSTR lpDirectory;
    int nShow;
    DWORD dwHotKey;
    HANDLE hIcon;
} CMINVOKECOMMANDINFO, *LPCMINVOKECOMMANDINFO;
```

Members

cbSize

Contains the size of this structure, in bytes.

fMask

Zero, or one or more of the following flags:

CMIC_MASK_ASYNCOK	Wait for the DDE conversation to terminate before returning.
CMIC_MASK_FLAG_NO_UI	The system is prevented from displaying user-interface elements (for example, error messages) while carrying out a command.
CMIC_MASK_HOTKEY	The **dwHotKey** member is valid.
CMIC_MASK_ICON	The **hIcon** member is valid.
CMIC_MASK_NO_CONSOLE	If a context menu handler needs to create a new process, it normally will create a new console. Setting the CMIC_MASK_NO_CONSOLE flag suppresses the creation of a new console.

hwnd

Handle to the window that is the owner of the context menu. An extension also can use this handle as the owner of any message boxes or dialog boxes it displays.

lpVerb

32-bit value that contains zero in the high-order word and a menu-identifier offset of the command to carry out in the low-order word. The shell specifies this value (using the **MAKEINTRESOURCE** macro) when the user chooses a menu command.

If the high-order word is not zero, this member is the address of a null-terminated string specifying the language-independent name of the command to carry out. This member is typically a string when a command is being activated by an application. The system provides predefined constant values for the following command strings:

Value	String
CMDSTR_NEWFOLDER	"NewFolder"
CMDSTR_VIEWDETAILS	"ViewDetails"
CMDSTR_VIEWLIST	"ViewList"

lpParameters

An optional string containing parameters that are passed to the command. The format of this string is determined by the command that is to be invoked. This member is always NULL for menu items inserted by a shell extension.

lpDirectory

Optional working directory name. This member is always NULL for menu items inserted by a shell extension.

nShow

Set of SW_ values to pass to the **ShowWindow** function if the command displays a window or starts an application.

dwHotKey

Optional hot key to assign to any application activated by the command. If the **fMask** parameter does not specify CMIC_MASK_HOTKEY, this member is ignored.

hIcon

Icon to use for any application activated by the command. If the **fMask** member does not specify CMIC_MASK_ICON, this member is ignored.

Remarks

Although the **IContextMenu::InvokeCommand** declaration specifies a **CMINVOKECOMMANDINFO** structure for the *pici* parameter, it can also accept a **CMINVOKECOMMANDINFOEX** structure. If you are implementing this method, you must inspect **cbSize** to determine which structure has been passed.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.

Windows CE: Unsupported.
Header: Declared in shlobj.h.

CMINVOKECOMMANDINFOEX

Contains extended information about a context menu command. This structure is an extended version of **CMINVOKECOMMANDINFO** that allows the use of Unicode values.

```
typedef struct _CMInvokeCommandInfoEx {
    DWORD cbSize;
    DWORD fMask;
    HWND hwnd;
    LPCSTR lpVerb;
    LPCSTR lpParameters;
    LPCSTR lpDirectory;
    int nShow;
    DWORD dwHotKey;
    HANDLE hIcon;
    LPCSTR lpTitle;
    LPCWSTR lpVerbW;
    LPCWSTR lpParametersW;
    LPCWSTR lpDirectoryW;
    LPCWSTR lpTitleW;
    POINT ptInvoke;
} CMINVOKECOMMANDINFOEX, *LPCMINVOKECOMMANDINFOEX;
```

Members

cbSize

Contains the size of this structure, in bytes.

fMask

Zero, or one or more of the following flags:

CMIC_MASK_HOTKEY	The **dwHotKey** member is valid.
CMIC_MASK_ICON	The **hIcon** member is valid.
CMIC_MASK_FLAG_NO_UI	The system is prevented from displaying user-interface elements (for example, error messages) while carrying out a command.
CMIC_MASK_NO_CONSOLE	If a context menu handler needs to create a new process, it normally will create a new console. Setting the CMIC_MASK_NO_CONSOLE flag suppresses the creation of a new console.

CMIC_MASK_UNICODE	Indicates that the context menu handler should use **lpVerbW**, **lpParametersW**, **lpDirectoryW**, and **lpTitleW** members instead of their ANSI equivalents. Because some context menu handlers might not support Unicode, you also should pass valid ANSI strings in the **lpVerb**, **lpParameters**, **lpDirectory**, and **lpTitle** members.

hwnd

Handle to the window that is the owner of the context menu. An extension also can use this handle as the owner of any message boxes or dialog boxes it displays.

lpVerb

32-bit value that contains zero in the high-order word and a menu-identifier offset of the command to carry out in the low-order word. The shell specifies this value (using the **MAKEINTRESOURCE** macro) when the user chooses a menu command.

If the high-order word is not zero, this member is the address of a null-terminated string specifying the language-independent name of the command to carry out. This member is typically a string when a command is being activated by an application. The system provides predefined constant values for the following command strings:

Value	String
CMDSTR_NEWFOLDER	"NewFolder"
CMDSTR_VIEWDETAILS	"ViewDetails"
CMDSTR_VIEWLIST	"ViewList"

lpParameters

Optional parameters. This member is always NULL for menu items inserted by a shell extension.

lpDirectory

Optional working directory name. This member is always NULL for menu items inserted by a shell extension.

nShow

Set of SW_ values to pass to the **ShowWindow** function if the command displays a window or starts an application.

dwHotKey

Optional hot key to assign to any application activated by the command. If the **fMask** parameter does not specify CMIC_MASK_HOTKEY, this member is ignored.

hIcon

Icon to use for any application activated by the command. If the **fMask** member does not specify CMIC_MASK_ICON, this member is ignored.

lpTitle

ASCII title.

lpVerbW

Unicode verb, for those commands that can use it.

lpParametersW
Unicode parameters, for those commands that can use it.

lpDirectoryW
Unicode directory, for those commands that can use it.

lpTitleW
Unicode title.

ptInvoke
Point where the command is invoked. This member is not valid prior to Internet Explorer 4.0.

Remarks

Although the **IContextMenu::InvokeCommand** declaration specifies a **CMINVOKECOMMANDINFO** structure for the *pici* parameter, it can also accept a **CMINVOKECOMMANDINFOEX** structure. If you are implementing this method, you must inspect **cbSize** to determine which structure has been passed.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

COMPONENT

Used by Microsoft Windows 2000 to hold information about a component. This structure replaces the **IE4COMPONENT** structure.

```
typedef struct _tagCOMPONENT
{
    DWORD dwSize;
    DWORD dwID;
    int iComponentType;
    BOOL fChecked;
    BOOL fDirty;
    BOOL fNoScroll;
    COMPPOS cpPos;
    WCHAR wszFriendlyName[MAX_PATH];
    WCHAR wszSource[INTERNET_MAX_URL_LENGTH];
    WCHAR wszSubscribedURL[INTERNET_MAX_URL_LENGTH];
    DWORD dwCurItemState;
    COMPSTATEINFO csiOriginal;
    COMPSTATEINFO csiRestored;
}
COMPONENT, *LPCOMPONENT;
```

Members

dwSize
Size of the structure.

dwID
Reserved. Set to zero.

iComponentType
Component type. It can take one of the following values:

Value	Description
COMP_TYPE_CONTROL	Active X Control. This value is valid only for **IActiveDesktop::AddDesktopItem**.
COMP_TYPE_HTMLDOC	HTML document.
COMP_TYPE_PICTURE	Picture.
COMP_TYPE_WEBSITE	Web site.

fChecked
Value that is set to TRUE if the component is enabled, or FALSE if it is not.

fDirty
Value that is set to TRUE if the component has been modified and not yet saved to disk. It will be set to FALSE if the component has not been modified, or if it has been modified and saved to disk.

fNoScroll
Value that is set to TRUE if the component is scrollable, or FALSE if not.

cpPos
COMPPOS structure containing position and size information.

wszFriendlyName
Component's friendly name.

wszSource
Component's URL.

wszSubscribed
Subscribed URL.

dwCurItemState
Current state of the component. It can take one of the following values:

Value	Description
IS_FULLSCREEN	Full screen
IS_NORMAL	Normal screen
IS_SPLIT	Split screen

csiOriginal

COMPSTATEINFO structure with the state of the component when it first was added.

csiRestored

COMPSTATEINFO structure with the restored state of the component.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

COMPONENTSOPT

Contains the desktop-item options.

```
typedef struct _tagCOMPONENTSOPT {
    DWORD dwSize;
    BOOL fEnableComponents;
    BOOL fActiveDesktop;
} COMPONENTSOPT;
```

Members

dwSize

Unsigned long integer value that contains the size of the structure.

fEnableComponents

Boolean value that determines if desktop items are enabled.

fActiveDesktop

Boolean value that determines if the Active Desktop is enabled.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

COMPPOS

This structure holds information about a component's position and size.

```
typedef struct _tagCOMPPOS
{
    DWORD    dwSize;
    int    iLeft;
    int    iTop;
    DWORD    dwWidth;
    DWORD    dwHeight;
    int    izIndex;
    BOOL    fCanResize;
    BOOL    fCanResizeX;
    BOOL    fCanResizeY;
    int    iPreferredLeftPercent;
    int     iPreferredTopPercent;
}
COMPPOS, *LPCOMPPOS;
```

Members

dwSize
The size of the structure.

iLeft
The left edge of the top-left corner in screen coordinates.

iTop
The top of the top-left corner in screen coordinates.

dwWidth
The width, in pixels.

dwHeight
The height, in pixels.

izIndex
The z-order of the component.

fCanResize
Set to TRUE if the component is resizable, FALSE if not.

fCanResizeX
Set to TRUE if the component is resizable in the x-direction, FALSE if not.

fCanResizeY
Set to TRUE if the component is resizable in the y-direction, FALSE if not.

iPreferredLeftPercent
The left edge of the upper-left corner as a percentage of screen width.

iPreferredTopPercent
The top of the upper-left corner as a percentage of screen width.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

COMPSTATEINFO

Used by Microsoft Windows 2000 to hold information about a component's state.

```
typedef struct _tagCOMPSTATEINFO
{
    DWORD dwSize;
    int iLeft;
    int iTop;
    DWORD dwWidth;
    DWORD dwHeight;
    DWORD dwItemState;
}
COMPSTATEINFO, *LPCOMPSTATEINFO;
```

Members

dwSize
Size of the structure.

iLeft
Left edge of the top-left corner in screen coordinates.

iTop
Top of the top-left corner in screen coordinates.

dwWidth
Width, in pixels.

dwHeight
Height, in pixels.

dwItemState
State of the component.

IS_FULLSCREEN	Full screen
IS_NORMAL	Normal screen
IS_SPLIT	Split screen

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

CPLINFO

Contains resource information and an application-defined value for a dialog box supported by a Control Panel application. The **CPlApplet** function of the Control Panel application returns this information to the Control Panel in response to a **CPL_INQUIRE** message.

```
typedef struct tagCPLINFO {
    int  idIcon;
    int  idName;
    int  idInfo;
    LONG lData;
} CPLINFO;
```

Members

idIcon
Resource identifier of the icon that represents the dialog box.

idName
Resource identifier of the string containing the short name for the dialog box. This name is intended to be displayed below the icon.

idInfo
Resource identifier of the string containing the description for the dialog box that is intended to be displayed when the application icon is selected.

lData
Data defined by the application. When the Control Panel sends the **CPL_DBLCLK** and **CPL_STOP** messages, it passes this value back to your application.

Remarks

If the icon or display strings of the dialog box can change based on the state of the computer, you can specify the CPL_DYNAMIC_RES value for the **idIcon**, **idName**, or **idInfo** members, instead of specifying a valid resource identifier. This causes the Control Panel to send the **CPL_NEWINQUIRE** message each time it needs the icon and display strings. Using this technique is significantly slower, however, because the Control Panel will need to load your application each time it sends the **CPL_NEWINQUIRE** message.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.

Windows CE: Requires version 1.0 or later.
Header: Declared in cpl.h.

DATABLOCK_HEADER

Serves as the header for some of the extra data structures used by **IShellLinkDataList**.

```
typedef struct tagDATABLOCKHEADER
{
    DWORD cbSize;
    DWORD dwSignature;
} DATABLOCK_HEADER, *LPDATABLOCK_HEADER, *LPDBLIST;
```

Members

cbSize
Size of the extra data block.

dwSignature
Signature that identifies the type of data block that follows the header.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

EXP_DARWIN_LINK, **NT_CONSOLE_PROPS**, **NT_FE_CONSOLE_PROPS**

DESKBANDINFO

Contains and receives information for a band object. This structure is used with the **IDeskBand::GetBandInfo** method.

```
typedef struct {
    DWORD dwMask;
    POINTL ptMinSize;
    POINTL ptMaxSize;
    POINTL ptIntegral;
    POINTL ptActual;
    WCHAR wszTitle[256];
    DWORD dwModeFlags;
    COLORREF crBkgnd;
} DESKBANDINFO;
```

Members

dwMask

Set of flags that determine which members of this structure are being requested. This will be a combination of the following values:

DBIM_ACTUAL	**ptActual** is being requested.
DBIM_BKCOLOR	**crBkgnd** is being requested.
DBIM_INTEGRAL	**ptIntegral** is being requested.
DBIM_MAXSIZE	**ptMaxSize** is being requested.
DBIM_MINSIZE	**ptMinSize** is being requested.
DBIM_MODEFLAGS	**dwModeFlags** is being requested.
DBIM_TITLE	**wszTitle** is being requested.

ptMinSize

POINTL structure that receives the minimum size of the band object. The minimum width is placed in the **x** member, and the minimum height is placed in the **y** member.

ptMaxSize

POINTL structure that receives the maximum size of the band object. The maximum height is placed in the **y** member, and the **x** member is ignored. If there is no limit for the maximum height, (LONG)–1 should be used.

ptIntegral

POINTL structure that receives the sizing step value of the band object. The vertical step value is placed in the **y** member, and the **x** member is ignored. The step value determines in what increments the band will be resized. This member is ignored if **dwModeFlags** does not contain DBIMF_VARIABLEHEIGHT.

ptActual

POINTL structure that receives the ideal size of the band object. The ideal width is placed in the **x** member, and the ideal height is placed in the **y** member. The band container will attempt to use these values, but the band is not guaranteed to be this size.

wszTitle

WCHAR buffer that receives the title of the band.

dwModeFlags

A value that receives a set of flags that define the mode of operation for the band object. This must be one or a combination of the following values:

DBIMF_BKCOLOR
The band will be displayed with the background color specified in **crBkgnd**.

DBIMF_DEBOSSED
The band object is displayed with a sunken appearance.

DBIMF_NORMAL
The band is normal in all respects. The other mode flags modify this flag.

DBIMF_VARIABLEHEIGHT
The height of the band object can be changed. The **ptIntegral** member defines the step value by which the band object can be resized.

crBkgnd

A value that receives the background color of the band. This member is ignored if **dwModeFlags** does not contain the DBIMF_BKCOLOR flag.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

DLLVERSIONINFO

Receives DLL-specific version information. It is used with the DllGetVersion function.

```
typedef struct _DllVersionInfo
{
    DWORD cbSize;
    DWORD dwMajorVersion;
    DWORD dwMinorVersion;
    DWORD dwBuildNumber;
    DWORD dwPlatformID;
}DLLVERSIONINFO;
```

Members

cbSize

Size of the structure, in bytes. This member must be filled in before calling the function.

dwMajorVersion

Major version of the DLL. If the DLL's version is 4.0.950, this value will be 4.

dwMinorVersion

Minor version of the DLL. If the DLL's version is 4.0.950, this value will be 0.

dwBuildNumber

Build number of the DLL. If the DLL's version is 4.0.950, this value will be 950.

dwPlatformID

Identifies the platform for which the DLL was built. This can be one of the following values:

DLLVER_PLATFORM_NT	The DLL was built specifically for Windows NT.
DLLVER_PLATFORM_WINDOWS	The DLL was built for all Windows platforms.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.

DLLVERSIONINFO2

Receives DLL-specific version information. It is used with the **DllGetVersion** function.

```
typedef struct _DLLVERSIONINFO2
{
    DLLVERSIONINFO info1;
    DWORD dwFlags;
    ULONGLONG ullVersion;
}DLLVERSIONINFO2;
```

Members

info1

DLLVERSIONINFO structure. This member is included to provide backward compatibility with applications that are not expecting a **DLLVERSIONINFO2** structure.

dwFlags

Reserved for future use.

ullVersion

Value that contains the version information. It is divided into four 16-bit fields containing the major and minor version numbers, the build number, and the QFE version, in that order. Use the **MAKEDLLVERULL** macro to construct this value.

Remarks

Your application must set the **cbSize** member of the structure pointed to by **info1** to **sizeof(DLLGETVERSIONINFO2)** before calling **DllGetVersion**. Otherwise, no value will be assigned to the **dwFlags** or **ullVersion** member of the **DLLGETVERSIONINFO2** structure.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlwapi.h.

DROPFILES

Defines the CF_HDROP and CF_PRINTERS clipboard formats. In the CF_HDROP case, the data that follows is a double null-terminated list of file names. For CF_PRINTERS, the data that follows are the printer friendly names.

```
typedef struct _DROPFILES {
    DWORD pFiles;
    POINT pt;
    BOOL fNC;
    BOOL fWide;
} DROPFILES, FAR * LPDROPFILES;
```

Members

pFiles

Offset of the file list from the beginning of this structure, in bytes.

pt

Drop point. The coordinates depend on **fNC**.

fNC

Nonclient area flag. If this member is TRUE, **pt** specifies the screen coordinates of a point in a window's nonclient area. If it is FALSE, **pt** specifies the client coordinates of a point in the client area.

fWide

Value that indicates whether the file contains ANSI or Unicode characters. If it is zero, it contains ANSI characters. Otherwise, it contains Unicode characters.

Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

EXP_DARWIN_LINK

Holds an extra data block used by **IShellLinkDataList**. It holds the link's Windows Installer ID.

```
typedef struct {
    DATABLOCK_HEADER dbh;
    CHAR           szDarwinID[MAX_PATH];
    WCHAR          szwDarwinID[MAX_PATH];
} EXP_DARWIN_LINK, *LPEXP_DARWIN_LINK;
```

Members

dbh

DATABLOCK_HEADER structure with the **EXP_DARWIN_LINK** structure's size and signature. There are two available signatures.

EXP_DARWIN_ID_SIG	Contains a Windows Installer ID.
EXP_LOGO3_ID_SIG	Not currently supported.

szDarwinID

ANSI string with the the link's ID.

szwDarwinID

Unicode string with the link's ID.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

EXP_SPECIAL_FOLDER

Holds an extra data block used by **IShellLinkDataList**. It holds special folder information.

```
typedef struct
{
    DWORD cbSize;
    DWORD dwSignature;
    DWORD idSpecialFolder;
    DWORD cbOffset;
} EXP_SPECIAL_FOLDER, *LPEXP_SPECIAL_FOLDER;
```

Members

cbSize

Size of the **EXP_SPECIAL_FOLDER** structure.

dwSignature

Structure's signature. It should be set to EXP_SPECIAL_FOLDER_SIG.

idSpecialFolder

ID of the special folder into which the link points.

cbOffset

Offset into the saved PIDL.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

EXP_SZ_LINK

Holds an extra data block used by **IShellLinkDataList**. It holds expandable environment strings for the icon or target.

```
typedef struct
{
    DWORD       cbSize;
    DWORD       dwSignature;
    CHAR        szTarget[ MAX_PATH ];
    WCHAR       swzTarget[ MAX_PATH ];
} EXP_SZ_LINK, *LPEXP_SZ_LINK;
```

Members

cbSize

Size of the **EXP_SZ_LINK** structure.

dwSignature

Structure's signature. It can have one of the following values:

EXP_SZ_ICON_SIG	Contains the link's icon path.
EXP_SZ_LINK_SIG	Contains the link's target path.

szTarget

NULL-terminated ANSI string with the path of the target or icon.

swzTarget

NULL-terminated Unicode string with the path of the target or icon.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

EXT_BUTTON

Contains information about a button that a File Manager extension dynamic-link library is adding to the toolbar of File Manager.

```
typedef struct _EXT_BUTTON {
    WORD idCommand;
    WORD idsHelp;
    WORD fsStyle;
} EXT_BUTTON;
```

Members

idCommand

Command identifier for the button.

idsHelp

Identifier of the Help string for the button.

fsStyle

Style of the button.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

See Also

FMEVENT_TOOLBARLOAD, FMS_TOOLBARLOAD

EXTRASEARCH

Used by an **IEnumExtraSearch** enumerator object to return information on the search objects supported by a Shell Folder object.

```
typedef struct tagEXTRASEARCH
{
    GUID    guidSearch;
    WCHAR   wszFriendlyName[80];
    WCHAR   wszUrl[2084];
}EXTRASEARCH, *LPEXTRASEARCH;
```

Members

guidSearch
Search object's globally unique identifier (GUID).

wszFriendlyName
A Unicode string containing the search object's friendly name. It will be used to identify the search engine on the **Search Assistant** menu.

wszUrl
The URL that will be displayed in the search pane.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

FILEDESCRIPTOR

Describes the properties of a file that is being copied by means of the clipboard during an OLE drag-and-drop operation.

```
typedef struct _FILEDESCRIPTOR {
    DWORD    dwFlags;
    CLSID    clsid;
    SIZEL    sizel;
    POINTL   pointl;
    DWORD    dwFileAttributes;
    FILETIME ftCreationTime;
    FILETIME ftLastAccessTime;
    FILETIME ftLastWriteTime;
    DWORD    nFileSizeHigh;
    DWORD    nFileSizeLow;
    TCHAR    cFileName[MAX_PATH];
} FILEDESCRIPTOR, *LPFILEDESCRIPTOR;
```

Members

dwFlags

Array of flags that indicate which of the other structure members contain valid data. This member can be a combination of the following values:

Flag	Description
FD_ACCESSTIME	The **ftLastAccessTime** member is valid.
FD_ATTRIBUTES	The **dwFileAttributes** member is valid.
FD_CLSID	The **clsid** member is valid.
FD_CREATETIME	The **ftCreationTime** member is valid.
FD_FILESIZE	The **nFileSizeHigh** and **nFileSizeLow** members are valid.
FD_LINKUI	Treat the operation as “Link.”
FD_SIZEPOINT	The **sizel** and **pointl** members are valid.
FD_WRITESTIME	The **ftLastWriteTime** member is valid.

clsid

File class identifier.

sizel

Width and height of the file icon.

pointl

Screen coordinates of the file object.

dwFileAttributes

File attribute flags. This will be a combination of the FILE_ATTRIBUTE_ values described in **GetFileAttributes**.

ftCreationTime

FILETIME structure that contains the time of file creation.

ftLastAccessTime

FILETIME structure that contains the time that the file was last accessed.

ftLastWriteTime
FILETIME structure that contains the time of the last write operation.

nFileSizeHigh
High-order DWORD of the file size, in bytes.

nFileSizeLow
Low-order DWORD of the file size, in bytes.

cFileName
Null-terminated string that contains the name of the file.

Remarks

If the **CFSTR_FILECONTENTS** format that corresponds to this structure contains the file as a global memory object, **nFileSizeHigh** and **nFileSizeLow** specify the size of the associated memory block. If they are set, they can also be used if a user interface needs to be displayed. For instance, if a file is about to be overwritten, you normally would use information from this structure to display a dialog box containing the size, data, and name of the file.

To create a zero-length file, set the FD_FILESIZE flag in the **dwFlags member**, and set **nFileSizeHigh** and **nFileSizeLow** to zero. The **CFSTR_FILECONTENTS** format should represent the file as either a stream or global memory object (TYMED_ISTREAM or TYMED_HGLOBAL).

Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

FILEGROUPDESCRIPTOR

Defines the CF_FILEGROUPDESCRIPTOR clipboard format.

```
typedef struct _FILEGROUPDESCRIPTOR {
    UINT cItems;
    FILEDESCRIPTOR fgd[1];
} FILEGROUPDESCRIPTOR, * LPFILEGROUPDESCRIPTOR;
```

Members

cItems
Number of elements in **fgd**.

fgd
Array of **FILEDESCRIPTOR** structures that contain the file information.

Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

FMS_GETDRIVEINFO

Contains information about the drive selected in the active File Manager window (the directory window or the Search Results window).

```
typedef struct _FMS_GETDRIVEINFO {
    DWORD dwTotalSpace;
    DWORD dwFreeSpace;
    TCHAR  szPath[260];
    TCHAR  szVolume[14];
    TCHAR  szShare[128];
} FMS_GETDRIVEINFO;
```

Members

dwTotalSpace
Total amount of storage space, in bytes, on the disk associated with the drive.

dwFreeSpace
Amount of free storage space, in bytes, on the disk associated with the drive.

szPath
Null-terminated path of the current directory.

szVolume
Null-terminated volume label of the disk associated with the drive.

szShare
Null-terminated name of the network resource (if the drive is being accessed through a network).

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

See Also

FMExtensionProc, **FM_GETDRIVEINFO**

FMS_GETFILESEL

Contains information about a selected file in the active File Manager window (the directory window or the Search Results window).

```
typedef struct _FMS_GETFILESEL {
    FILETIME ftTime;
    DWORD    dwSize;
    BYTE     bAttr;
    TCHAR     szName[260];
} FMS_GETFILESEL;
```

Members

ftTime
Time and date the file was created.

dwSize
Size, in bytes, of the file.

bAttr
Attributes of the file.

szName
Null-terminated full path and file name of the selected file.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

+ See Also

FMExtensionProc

FMS_HELPSTRING

Contains information that File Manager uses to add a help string for a menu or toolbar command item.

```
typedef struct _FMS_HELPSTRING {
    INT   idCommand;
    HMENU hMenu;
    CHAR szHelp[128];
} FMS_HELPSTRING;
```

Members

idCommand
Identifier of the command item.

hMenu
Identifer of the menu bar.

szHelp
Null-terminated string that receives the help text.

Requirements

Windows NT/2000: Requires Windows NT 3.5 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

See Also

FMExtensionProc, **FMEVENT_HELPMENUITEM**

FMS_LOAD

Contains information that File Manager uses to add a custom menu provided by a File Manager extension DLL. The structure also provides a delta value that the extension DLL can use to manipulate the custom menu after File Manager has loaded the menu.

```
typedef struct _FMS_LOAD {
    DWORD dwSize;
    TCHAR  szMenuName[MENU_TEXT_LEN];
    HMENU hMenu;
    UINT  wMenuDelta;
} FMS_LOAD;
```

Members

dwSize
Length, in bytes, of the structure.

szMenuName
Null-terminated name for a menu item that appears on the menu bar in File Manager.

hMenu
Identifer of the pop-up menu added to the menu bar in File Manager.

wMenuDelta
Menu-item delta value. To avoid conflicts with its own menu items, File Manager renumbers the menu item identifiers in the pop-up menu identified by the **hMenu** member by adding this delta value to each identifier. An extension DLL that must modify a menu item must identify the item by adding the delta value to the menu item's identifier. The value of this member can vary from session to session.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

See Also

FMExtensionProc

FMS_TOOLBARLOAD

Contains information about custom buttons to be added to the File Manager toolbar. The buttons are provided by a File Manager extension DLL.

```
typedef struct _FMS_TOOLBARLOAD {
    DWORD        dwSize;
    LPEXT_BUTTON lpButtons;
    WORD         cButtons;
    WORD         cBitmaps;
    WORD         idBitmap;
    HBITMAP      hBitmap;
} FMS_TOOLBARLOAD;
```

Members

dwSize
Size, in bytes, of the structure. File Manager sets the size before calling the extension, and checks the size after the extension procedure returns.

lpButtons
Address of an array of **EXT_BUTTON** structures.

cButtons
Number of **EXT_BUTTON** structures in the array pointed to by the **lpButtons** member. This number equals the number of buttons and separators to add to the toolbar.

cBitmaps
Number of buttons represented by the given bitmap.

idBitmap
Identifier of a bitmap resource in the executable file for the extension DLL. The bitmap resource contains images for the number of buttons specified by **cBitmaps**. File Manager loads the bitmap resource and then uses it to display the buttons.

hBitmap
Handle to a bitmap that File Manager will use to obtain and display button images if **idBitmap** is 0.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

+ See Also

FMEVENT_TOOLBARLOAD

FOLDERSETTINGS

Contains folder view information.

```
typedef struct {
    UINT    ViewMode;
    UINT    fFlags;
}FOLDERSETTINGS, *LPFOLDERSETTINGS;
```

Members

ViewMode
Folder view mode. This can be one of the **FOLDERVIEWMODE** values.

fFlags
Set of flags that indicate the options for the folder. This can be zero or a combination of the **FOLDERFLAGS** values.

Remarks

These settings assume a particular user interface, which the shell's folder view has. A shell extension can use these settings if they apply to the view implemented by the extension.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IShellView::CreateViewWindow, **IShellView::GetCurrentInfo**

FVSHOWINFO

Contains information that the file viewer uses to display a file.

```
typedef struct {
    DWORD cbSize;
    HWND hwndOwner;
    int iShow;
    DWORD dwFlags;
    RECT rect;
    LPUNKNOWN punkRel;
    OLECHAR strNewFile[MAX_PATH];
} FVSHOWINFO, *LPFVSHOWINFO;
```

Members

cbSize

Size of the structure, in bytes.

hwndOwner

Window handle to the owner of the window in which the file will be displayed.

iShow

Show command for the window. This parameter is one of the SW_ values detailed in **ShowWindow**.

dwFlags

Flags that determine what the file viewer displays. This member can be one or more of the following values:

FVSIF_CANVIEWIT	The file viewer can display the file.
FVSIF_NEWFAILED	The file viewer specified a new file to display, but no viewer could display the file. The file viewer should either continue to display the previous file or terminate.
FVSIF_NEWFILE	A drag-and-drop operation has dropped a file on the file viewer window. The file viewer passes the name of the file to the shell by copying the name to the **strNewFile** member. The shell attempts to load a file viewer that can display the new file.
FVSIF_PINNED	A pinned window exists. A file viewer should either use the pinned window to display the file or set a new pinned window and display the file in it.
FVSIF_RECT	The **rect** member contains valid data.

rect

RECT structure that specifies the size and position of the file viewer's window. This member is valid only if the **dwFlags** member includes the FVSIF_RECT value.

punkRel

Address of an interface that has its **Release** method called by a new file viewer to release the previous file viewer. This member is used when a drag-and-drop operation drops a file on the file viewer's window.

strNewFile

Address of a string that specifies the name of a new file to display. A file viewer sets this member when a drag-and-drop operation drops a file on the file viewer's window.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

HELPINFO

Contains information about an item for which context-sensitive Help has been requested.

```
typedef struct tagHELPINFO {
    UINT      cbSize;
    int       iContextType
    int       iCtrlId;
    HANDLE    hItemHandle;
    DWORD     dwContextId;
    POINT     MousePos;
} HELPINFO, FAR *LPHELPINFO;
```

Members

cbSize

Structure size, in bytes.

iContextType

Type of context for which Help is requested. This member can be one of the following values:

HELPINFO_MENUITEM	Help requested for a menu item
HELPINFO_WINDOW	Help requested for a control or window

iCtrlId

Identifier of the window or control if **iContextType** is HELPINFO_WINDOW, or identifier of the menu item if **iContextType** is HELPINFO_MENUITEM.

hItemHandle

Identifier of the child window or control if **iContextType** is HELPINFO_WINDOW, or identifier of the associated menu if **iContextType** is HELPINFO_MENUITEM.

dwContextId
Help context identifier of the window or control.

MousePos
POINT structure that contains the screen coordinates of the mouse cursor. This is useful for providing Help based on the position of the mouse cursor.

! Requirements

Windows NT/2000: Requires Windows NT 3.5 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in winuser.h.

HELPWININFO

Contains the size and position of either a primary or a secondary Help window. An application can set this information by calling the **WinHelp** function with the HELP_SETWINPOS value.

Syntax

```
typedef struct {
    int   wStructSize;
    int   x;
    int   y;
    int   dx;
    int   dy;
    int   wMax;
    TCHAR rgchMember[2];
} HELPWININFO;
```

Members

wStructSize
Structure size, in bytes.

x
X-coordinate of the upper-left corner of the window, in screen coordinates.

y
Y-coordinate of the upper-left corner of the window, in screen coordinates.

dx
Window width, in pixels.

dy
Window height, in pixels.

wMax
How to show the window. This member must be one of the following values:

SW_HIDE	Hides the window and passes activation to another window.
SW_MINIMIZE	Minimizes the specified window and activates the top-level window in the z-order.
SW_RESTORE	Same as SW_SHOWNORMAL.
SW_SHOW	Activates a window and displays it in its current size and position.
SW_SHOWMAXIMIZED	Activates the window and displays it as a maximized window.
SW_SHOWMINIMIZED	Activates the window and displays it as an icon.
SW_SHOWMINNOACTIVE	Displays the window as an icon. The window that currently is active remains active.
SW_SHOWNA	Displays the window in its current state. The window that currently is active remains active.
SW_SHOWNOACTIVATE	Displays a window in its most recent size and position. The window that currently is active remains active.
SW_SHOWNORMAL	Activates and displays the window. Whether the window is minimized or maximized, Windows restores it to its original size and position.

rgchMember

Name of the window.

Remarks

Windows Help divides the display into 1024 units in both the x and y directions. To create a secondary window that fills the upper-left quadrant of the display, for example, an application would specify zero for the **x** and **y** members, and 512 for the **dx** and **dy** members.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in winuser.h.

IE4COMPONENT

Used by Microsoft Internet Explorer 4.0 and 4.01 to hold information about a component. With Microsoft Windows 2000, it is replaced by the **COMPONENT** structure.

```
typedef struct _tagIE4COMPONENT
{
    DWORD dwSize;
    DWORD dwID;
    int iComponentType;
    BOOL fChecked;
    BOOL fDirty;
    BOOL fNoScroll;
    COMPPOS cpPos;
    WCHAR wszFriendlyName[MAX_PATH];
    WCHAR wszSource[INTERNET_MAX_URL_LENGTH];
    WCHAR wszSubscribedURL[INTERNET_MAX_URL_LENGTH];
}
IE4COMPONENT, *LPIE4COMPONENT;
```

Members

dwSize

Size of the structure.

dwID

Reserved. Set to zero.

iComponentType

Component type. It can be set to one of these values:

COMP_TYPE_CONTROL	Control
COMP_TYPE_HTMLDOC	HTML document
COMP_TYPE_PICTURE	Picture
COMP_TYPE_WEBSITE	Web site

fChecked

Value that is set to TRUE if the component is enabled, or FALSE if not.

fDirty

Value that is set to TRUE if the component has been modified and not yet saved to disk. It will be set to FALSE if the component has not been modified, or if it has been modified and saved to disk.

fNoScroll

Value that is set to TRUE if the component is scrollable, or FALSE if it is not.

cpPos

COMPPOS structure containing position and size information.

wszFriendlyName

Component's friendly name.

wszSource

Component's URL.

wszSubscribed

URL to which a user has been subscribed.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

ITEMIDLIST

Contains a list of item identifiers.

```
typedef struct _ITEMIDLIST {
    SHITEMID mkid;
} ITEMIDLIST, * LPITEMIDLIST;
typedef const ITEMIDLIST * LPCITEMIDLIST;
```

Members

mkid

List of item identifiers.

Remarks

A pointer to this structure, called a *PIDL*, is used to identify objects in the shell namespace. See *The Shell Namespace* for a discussion of PIDLs.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in shlobj.h.

MULTIKEYHELP

Specifies a keyword to search for and the keyword table to be searched by Windows Help.

```
typedef struct tagMULTIKEYHELP {
    DWORD  mkSize;
    TCHAR  mkKeylist;
    TCHAR  szKeyphrase[1];
} MULTIKEYHELP;
```

Members

mkSize

Structure size, in bytes.

mkKeylist

Single character that identifies the keyword table to search.

szKeyphrase

Null-terminated text string that specifies the keyword to locate in the keyword table.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in winuser.h.

See Also

WinHelp

NEWCPLINFO

Contains resource information and an application-defined value for a dialog box supported by a Control Panel application.

```
typedef struct tagNEWCPLINFO {
    DWORD dwSize;
    DWORD dwFlags;
    DWORD dwHelpContext;
    LONG  lData;
    HICON hIcon;
    TCHAR  szName[32];
    TCHAR  szInfo[64];
    TCHAR  szHelpFile[128];
} NEWCPLINFO;
```

Members

dwSize

Length of the structure, in bytes.

dwFlags

This member is ignored.

dwHelpContext

This member is ignored.

lData

Data defined by the application. When the Control Panel sends the **CPL_DBLCLK** and **CPL_STOP** messages, it passes this value back to your application.

hIcon

Identifier of the icon that represents the dialog box. This icon is intended to be displayed by the application that controls the Control Panel application.

szName

Null-terminated string that contains the dialog-box name. The name is intended to be displayed below the icon.

szInfo

Null-terminated string containing the dialog-box description. The description is intended to be displayed when the icon for the dialog box is selected.

szHelpFile

This member is ignored.

Remarks

The **CPlApplet** function of the Control Panel application returns this information to the Control Panel in response to a **CPL_NEWINQUIRE** message.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in cpl.h.

+ See Also

CPLINFO

NOTIFYICONDATA

Contains information that the system needs to process taskbar status-area messages.

```
typedef struct _NOTIFYICONDATA {
    DWORD cbSize;
    HWND hWnd;
    UINT uID;
    UINT uFlags;
    UINT uCallbackMessage;
    HICON hIcon;
    TCHAR szTip[64];
    DWORD dwState; //Version 5.0
```

(continued)

(continued)

```
    DWORD dwStateMask; //Version 5.0
    TCHAR szInfo[256]; //Version 5.0
    union {
        UINT  uTimeout; //Version 5.0
        UINT  uVersion; //Version 5.0
    } DUMMYUNIONNAME;
    TCHAR szInfoTitle[64]; //Version 5.0
    DWORD dwInfoFlags; //Version 5.0
} NOTIFYICONDATA, *PNOTIFYICONDATA;
```

Members

cbSize

Size of this structure, in bytes.

hWnd

Handle to the window that will receive notification messages associated with an icon in the taskbar status area. The shell uses **hWnd** and **uID** to identify which icon is to be operated on when **Shell_NotifyIcon** is invoked.

uID

Application-defined identifier of the taskbar icon. The shell uses *hWnd* and **uID** to identify which icon is to be operated on when **Shell_NotifyIcon** is invoked. You can have multiple icons associated with a single **hWnd** by assigning each a diffent *uID*.

uFlags

Array of flags that indicate which of the other members contain valid data. This member can be a combination of the following:

Flag	Description
NIF_ICON	The **hIcon** member is valid.
NIF_INFO	Use a balloon-style tooltip instead of a standard tooltip. The **szInfo**, **uTimeout**, **szInfoTitle**, and **dwInfoFlags** members are valid.
NIF_MESSAGE	The **uCallbackMessage** member is valid.
NIF_STATE	The **dwState** and **dwStateMask** members are valid.
NIF_TIP	The **szTip** member is valid.

uCallbackMessage

Application-defined message identifier. The system uses this identifier to send notifications to the window identified in **hWnd**. These notifications are sent when a mouse or event occurs in the bounding rectangle of the icon, or the icon is selected or activated with the keyboard. The *wParam* parameter of the message contains the identifier of the taskbar icon in which the event occurred. The *lParam* parameter holds the mouse or keyboard message associated with the event. For example, when the mouse cursor moves over a taskbar icon, *lParam* is set to **WM_MOUSEMOVE**. See the *Taskbar* guide chapter for further discussion.

hIcon

Handle to the icon to be added, modified, or deleted.

szTip

Pointer to a NULL-terminated string with the text for a standard tooltip. It can have a maximum of 64 characters, including the terminating NULL.

Version 5.0 and later. *szTip* can have a maximum of 128 characters, including the terminating NULL.

dwState

Version 5.0. State of the icon. There are two flags that can be set independently:

Flag	Description
NIS_HIDDEN	The icon is hidden.
NIS_SHAREDICON	The icon is shared.

dwStateMask

Version 5.0. A value that specifies which bits of the state member will be retrieved or modified. For example, setting this member to NIS_HIDDEN will cause only the item's hidden state to be retrieved.

szInfo

Version 5.0. Pointer to a NULL-terminated string with the text for a balloon-style tooltip. It can have a maximum of 255 characters. To remove the tooltip, set the NIF_INFO flag in **uFlags**, and set **szInfo** to an empty string.

uTimeout

The timeout value, in milliseconds, for a balloon-style tooltip. The system enforces minimum and maximum timeout values. **uTimeout** values that are too large are set to the maximum value and values that are too small default to the minimum value. The system minimum and maximum timeout values are set currently at 10 seconds and 30 seconds, respectively. These values can change in future versions of Windows. See the remarks for further discussion of **uTimeout**. Union with **uVersion**.

uVersion

Version 5.0. Used to specify whether the shell notify icon interface should use Windows 95 or Windows 2000 behavior. This member is only used when using **Shell_NotifyIcon** to send an **NIM_VERSION** message. Union with **uTimeout**.

Value	Description
0	Use the Windows 95 behavior.
NOTIFYICON_VERSION	Use the Windows 2000 behavior.

szInfoTitle

Version 5.0. Pointer to a NULL-terminated string containing a title for a balloon tooltip. This title will be in boldface, and placed above the text. It can have a maximum of 63 characters.

dwInfoFlags

Version 5.0. Flags that can be set to add an icon to a balloon tooltip; it will be placed to the left of the title. If the **szTitleInfo** member is zero-length, the icon will not be shown.

Flag	Description
NIIF_ERROR	An error icon
NIIF_INFO	An information icon
NIIF_WARNING	A warning icon

Remarks

If you set the NIF_INFO flag in the **uFlags** member, the standard tooltip, will be replaced by a balloon tooltip. For more discussion of balloon tooltips, see *Tooltip Controls*.

No more than one balloon tooltip at at time will be displayed for the taskbar. If an application attempts to display a tooltip when one is already being displayed, it will not appear until the existing balloon tooltip has been visible for at least the system minum timeout value. For example, a balloon tooltip with **uTimeout** set to 30 seconds has been visible for seven seconds when another application attempt to display a balloon tooltip. If the system minimum timeout is ten seconds, the first tooltip will be displayed for an additional three seconds before being replaced by the second tooltip.

Note that several members of this structure are only supported for Shell32.dll versions 5.0 and later. To enable these members, include the following in your header:

```
#define _WIN32_IE 0x0500
```

However, you must initialize the structure with its size. If you use the size of the currently defined structure, the application may not run with the earlier versions of Shell32.dll, which expect a smaller structure. You can run your application on pre-5.0 versions of Shell32.dll by defining the appropriate version number. However, this might cause problems if your application also needs to run on systems with more recent versions.

Your application can remain compatible with all Shell32.dll versions while still using the current header files by setting the size of the **NOTIFYICONDATA** structure appropriately. Before initializing the structure, use the **DllGetVersion** function to determine which Shell32.dll version is installed on the system. If it is version 5.0 or greater, initialize the **cbSize** member with:

```
nid.cbSize = sizeof(NOTIFYICONDATA);
```

Setting **cbSize** to this value will enable all the version 5.0 enhancements. For earlier versions, the size of the pre-5.0 structure is given by the NOTIFYICONDATA_V1_SIZE constant. Initialize the **cbSize** member with:

```
nid.cbSize = NOTIFYICONDATA_V1_SIZE;
```

Using this value for **cbSize** will allow your application to use **NOTIFYICONDATA** with earlier Shell32.dll versions, although without the version 5.0 enhancements.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in shellapi.h.

NRESARRAY

Defines the CF_NETRESOURCE clipboard format.

```
typedef struct _NRESARRAY {
    UINT        cItems;
    NETRESOURCE nr[1];
} NRESARRAY, * LPNRESARRAY;
```

Members

cItems

Number of elements in the **nr** array.

nr

Array of **NETRESOURCE** structures that contain information about network resources. The string members (LPSTR types) in the structure contain offsets instead of addresses.

! Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

NT_CONSOLE_PROPS

Holds an extra data block used by **IShellLinkDataList**. It holds console properties.

```
typedef struct {
    DATABLOCK_HEADER dbh;
    WORD      wFillAttribute;
    WORD      wPopupFillAttribute;
    COORD     dwScreenBufferSize;
    COORD     dwWindowSize;
    COORD     dwWindowOrigin;
```

(continued)

(continued)

```
    DWORD    nFont;
    DWORD    nInputBufferSize;
    COORD    dwFontSize;
    UINT     uFontFamily;
    UINT     uFontWeight;
    WCHAR    FaceName[LF_FACESIZE];
    UINT     uCursorSize;
    BOOL     bFullScreen;
    BOOL     bQuickEdit;
    BOOL     bInsertMode;
    BOOL     bAutoPosition;
    UINT     uHistoryBufferSize;
    UINT     uNumberOfHistoryBuffers;
    BOOL     bHistoryNoDup;
    COLORREF ColorTable[ 16 ];
} NT_CONSOLE_PROPS, *LPNT_CONSOLE_PROPS;
```

Members

dbh

DATABLOCK_HEADER structure with the **NT_CONSOLE_PROPS** structure's size and signature. The signature for an **NT_CONSOLE_PROPS** structure is NT_CONSOLE_PROPS_SIG.

wFillAttribute

Fill attribute for the console.

wPopupFillAttribute

Fill attribute for console pop-up windows.

dwScreenBufferSize

COORD structure with the console's screen-buffer size.

dwWindowSize

COORD structure with the console's window size.

dwWindowOrigin

COORD structure with the console's window origin.

nFont

Font.

nInputBufferSize

Input-buffer size.

dwFontSize

COORD structure with the font size.

uFontFamily

Font family.

uFontWeight

Font weight.

FaceName

Character array that contains the font's face name.

uCursorSize

Cursor size.

bFullScreen

Boolean value that is set to TRUE if the console is in full-screen mode, or FALSE otherwise.

bQuickEdit

Boolean value that is set to TRUE if the console is in quick-edit mode, or FALSE otherwise.

bInsertMode

Boolean value that is set to TRUE if the console is in insert mode, or FALSE otherwise.

bAutoPosition

Boolean value that is set to TRUE if the console is in auto-position mode, or FALSE otherwise.

uHistoryBufferSize

Size of the history buffer.

uNumberOfHistoryBuffers

Number of history buffers.

bHistoryNoDup

Boolean value that is set to TRUE if old duplicate history lists should be discarded, or FALSE otherwise.

ColorTable

Array of **COLORREF** values with the console's color settings.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

NT_FE_CONSOLE_PROPS

Holds an extra data block used by **IShellLinkDataList**. It holds the console's code page.

```
typedef struct {
    DATABLOCK_HEADER dbh;
    UINT      uCodePage;
} NT_FE_CONSOLE_PROPS, *LPNT_FE_CONSOLE_PROPS;
```

Members

dbh

DATABLOCK_HEADER structure with the **NT_FE_CONSOLE_PROPS** structure's size and signature. The signature for an **NT_FE_CONSOLE_PROPS** structure is NT_FE_CONSOLE_PROPS_SIG.

uCodePage

Console's code page.

Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

PERSIST_FOLDER_TARGET_INFO

Specifies a folder shortcut's target folder and its attributes. This structure is used by **IPersistFileSystemFolder::GetFolderTargetInfo** and **IPersistFileSystemFolder::InitializeEx**.

```
typedef struct
{
    LPITEMIDLIST  pidlTargetFolder;
    WCHAR         szTargetParsingName[MAX_PATH];
    WCHAR         szNetworkProvider[MAX_PATH];
    DWORD         dwAttributes;
    int           csidl;
} PERSIST_FOLDER_TARGET_INFO;
```

Members

pidlTargetFolder

Fully qualified **PIDL** of the target folder. Set **pidlTargetFolder** to NULL if not specified.

szTargetParsingName

NULL-terminated Unicode string with the target folder's parsing name. Set **szTargetParsingName** to an empty string if not specified.

szNetworkProvider

NULL-terminated Unicode string that specifies the type of network provider that will be used when binding to the target folder. The format is the same as that used by the **WNet API**. Set **szNetworkProvider** to an empty string if not specified.

dwAttributes

DWORD value that contains SFGAO_XXX file attribute flags. For a complete list of available flags, see **IShellFolder::ParseDisplayName**. Set **dwAttributes** to –1 if not specified.

csidl

Target folder's **CSIDL** value, if it has one. Set **csidl** to –1 if the target folder does not have a CSIDL. In addition to the CSIDL value, you can also set the following two flags:

CSIDL_FLAG_CREATE
Indicates that the target folder should be created if it does not exist already.

CSIDL_FLAG_PFTI_TRACKTARGET
Indicates that the target folder should change if the user changes the target folder's underlying CSIDL value.

Remarks

Any or all of the **pidlTargetFolder**, **szTargetParsingName**, and **csidl** members can be used to specify the target folder's location.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IPersistFolder3

SHCOLUMNDATA

Contains information that identifies a particular file. It is used by **IColumnProvider::GetItemData** when requesting data for a particular file.

```
typedef struct {
    ULONG   dwFlags;
    DWORD   dwFileAttributes;
    ULONG   dwReserved;
    WCHAR   *pwszExt;
    WCHAR   wszFile[MAX_PATH];
} SHCOLUMNDATA, *LPSHCOLUMNDATA;
typedef const SHCOLUMNDATA* LPCSHCOLUMNDATA;
```

Members

dwFlags

Flags used to specify the nature of the request.

Flag	Description
SHCDF_UPDATEITEM	The file specified by *wszFile* is a new file or it has changed since the last call to **IColumnProvider::GetItemData**. Any cached data should be flushed and recalculated. Column handlers that do not cache data or that display data that is stored separately from the file can ignore this flag.

dwFileAttributes

File attribute flags. It will be one or more of the following values:

Flag	Description
FILE_ATTRIBUTE_ARCHIVE	The file or directory is an archive file or directory. Applications use this attribute to mark files for backup or removal.
FILE_ATTRIBUTE_COMPRESSED	The file or directory is compressed. For a file, this means that all data in the file is compressed. For a directory, this means that compression is the default for newly created files and subdirectories.
FILE_ATTRIBUTE_DIRECTORY	The handle identifies a directory.
FILE_ATTRIBUTE_ENCRYPTED	The file or directory is encrypted. For a file, this means that all data streams in the file are encrypted. For a directory, this means that encryption is the default for newly created files and subdirectories.
FILE_ATTRIBUTE_HIDDEN	The file or directory is hidden. It is not included in an ordinary directory listing.
FILE_ATTRIBUTE_NORMAL	The file or directory has no other attributes set. This attribute is valid only if used alone.

Flag	Description
FILE_ATTRIBUTE_OFFLINE	The data of the file is not immediately available. This attribute indicates that the file data has been physically moved to offline storage. This attribute is used by Remote Storage, the hierarchical storage management software in Microsoft Windows 2000. Applications should not change arbitrarily this attribute.
FILE_ATTRIBUTE_READONLY	The file or directory is read-only. Applications can read the file but cannot write to it or delete it. In the case of a directory, applications cannot delete it.
FILE_ATTRIBUTE_REPARSE_POINT	The file has an associated reparse point.
FILE_ATTRIBUTE_SPARSE_FILE	The file is a sparse file.
FILE_ATTRIBUTE_SYSTEM	The file or directory is part of, or is used exclusively by, the operating system.
FILE_ATTRIBUTE_TEMPORARY	The file is being used for temporary storage. File systems attempt to keep all of the data in memory for quicker access instead of flushing the data back to mass storage. A temporary file should be deleted by the application as soon as it is no longer needed.

dwReserved

Reserved for future use. Set to NULL.

pwszExt

Pointer to a NULL-terminated Unicode string with a file-name extension.

wszFile

NULL-terminated Unicode string containing a fully qualified file path.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

GetFileAttributes

SHCOLUMNID

Specifies the FMTID/PID identifier of a column that will be displayed by the Windows Explorer details view.

```
typedef struct {
    GUID fmtid;
    DWORD pid;
} SHCOLUMNID, *LPSHCOLUMNID;
```

Members

fmtid

String form of a property set format identifier or FMTID (a GUID). The shell supports the storage, shell details, and summary information property sets. Other property sets can be supported by particular folders.

FMTID	Description
{B725F130-47EF-101A-A5F1-02608C9EEBAC}	Storage property set. Defined in Ntquery.h.
{28636AA6-953D-11D2-B5D6-00C04FD918D0}	Shell details property set. Defined in Shlguid.h.
{9B174B33-40FF-11D2-A27E-00C04FC30871}	Displaced property set. Defined in Shlguid.h.
{9B174B34-40FF-11D2-A27E-00C04FC30871}	Miscellaneous property set. Defined in Shlguid.h.
{9B174B35-40FF-11D2-A27E-00C04FC30871}	Volume property set. Defined in Shlguid.h.
{49691C90-7E17-101A-A91C-08002B2ECDA9}	Query property set. Defined in Shlguid.h.
{F29F85E0-4FF9-1068-AB91-08002B27B3D9}	Summary information property set. Defined in Ntquery.h.

pid

Column's property identifier (PID).

The storage property set supports the five PIDs, defined in Ntquery.h.

PID	Value	Description	Type
PID_STG_ATTRIBUTES	13	The object's attributes	VT_BSTR
PID_STG_NAME	10	The object's display name	VT_BSTR
PID_STG_SIZE	12	The object's size	VT_BSTR
PID_STG_STORAGETYPE	4	The object's type	VT_BSTR
PID_STG_WRITETIME	14	The object's modified attribute	VT_BSTR

The shell details property set supports three PIDs, defined in Shlguid.h.

PID	Value	Description	Type
PID_DESCRIPTIONID	2	An SHDESCRIPTIONID	VT_ARRAY I VT_UI1
PID_FINDDATA	0	A WIN32_FIND_DATA	VT_ARRAY I VT_UI1
PID_NETRESOURCE	1	A NETRESOURCE	VT_ARRAY I VT_UI1

The displaced property set supports files that have been deleted and moved to the Recycle Bin. There are two PIDs, defined in Shlguid.h.

PID	Value	Description
PID_DISPLACED_DATE	3	Date that the file was deleted
PID_DISPLACED_FROM	2	Location from which the file was deleted

The miscellaneous property set is used to support synchronization of briefcases or offline files. It has three PIDs, defined in Shlguid.h.

PID	Value	Description
PID_MISC_ACCESSCOUNT	3	The number of times the file has been accessed
PID_MISC_OWNER	4	Ownership of the file (for the NTFS file system)
PID_MISC_STATUS	2	The synchronization status

The query property set is used to support file searches. It has one PID, defined in Shlguid.h.

PID	Value	Description
PID_QUERY_RANK	2	The rank of the file

The volume property set provides volume information. It has one PID, defined in Shlguid.h.

PID	Value	Description
PID_VOLUME_FREE	2	The amount of free space

The summary information property set is a standard OLE property set that supports the following PIDs:

PID	Property name	Type
2	Title	VT_LPSTR
3	Subject	VT_LPSTR
4	Author	VT_LPSTR
5	Keywords	VT_LPSTR
6	Comments	VT_LPSTR
7	Template	VT_LPSTR
8	Last Saved By	VT_LPSTR
9	Revision Number	VT_LPSTR
10	Total Editing Time	VT_FILETIME
11	Last Printed	VT_FILETIME
12	Create Time/Date	VT_FILETIME
13	Last Saved Time/Date	VT_FILETIME
14	Number of Pages	VT_I4
15	Number of Words	VT_I4
16	Number of Characters	VT_I4
17	Thumbnail	VT_CF
18	Name of Creating Application	VT_LPSTR
19	Security	VT_I4

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellFolder2::GetDetailsEx

SHCOLUMNINFO

Contains information about the properties of a column. It is used by **IColumnProvider::GetColumnInfo**.

```
typedef struct {
    SHCOLUMNID  scid;
    VARTYPE     vt;
    DWORD       fmt;
    UINT        cChars;
    DWORD       csFlags;
    WCHAR wszTitle[MAX_COLUMN_NAME_LEN];
    WCHAR wszDescription[MAX_COLUMN_DESC_LEN];
} SHCOLUMNINFO, *LPSHCOLUMNINFO;
typedef const SHCOLUMNINFO* LPCSHCOLUMNINFO;
```

Members

scid

[out] **SHCOLUMNID** structure that uniquely identifies the column.

vt

[out] Native VARIANT type of the column's data.

fmt

[out] List-view format. This member normally is set to LVCFMT_LEFT.

cChars

[out] Default width of the column, in characters.

csFlags

[out] Flags indicating the default column state. It can be a combination of the following flags:

Flag	Description
SHCOLSTATE_EXTENDED	Provided by a handler, not the folder object.
SHCOLSTATE_HIDDEN	Not displayed in the user interface.
SHCOLSTATE_ONBYDEFAULT	Will be shown by default in Microsoft Windows Explorer Details view, even if the user has not selected the column. If this flag is set, the column will be displayed for all folders. There is no way to force a column to be displayed on a per-folder basis.
SHCOLSTATE_SECONDARYUI	Not displayed in the context menu, but listed in the **More** dialog box.
SHCOLSTATE_SLOW	Will be slow to compute. Windows Explorer should get the data asynchronously and do the computation on a background thread.
SHCOLSTATE_TYPE_DATE	A date.
SHCOLSTATE_TYPE_INT	An integer.
SHCOLSTATE_TYPE_STR	A string.

wszTitle

[out] Unicode string with the column's title. It must contain no more than MAX_COLUMN_NAME_LEN characters, including the terminating NULL.

wszDescription

[out] Unicode string with the column's description. It must contain no more than MAX_COLUMN_DESC_LEN characters, including the terminating NULL.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IColumnProvider::GetColumnInfo

SHCOLUMNINIT

Passes initialization information to **IColumnProvider::Initialize**.

```
typedef struct {
    ULONG    dwFlags;
    ULONG    dwReserved;
    WCHAR    wszFolder[MAX_PATH];
} SHCOLUMNINIT, *LPSHCOLUMNINIT;
typedef const SHCOLUMNINFO* LPCSHCOLUMNINFO;
```

Members

dwFlags

Initialization flags. Reserved for future use. Set to NULL

dwReserved

Reserved for future use. Set to NULL.

wszFolder

Pointer to a NULL-terminated Unicode string with a fully qualified folder path. The string will be empty if multiple folders are specified.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

SHCREATEPROCESSINFOW

Contains the information needed by **SHCreateProcessAsUser** to create a process.

```
typedef struct _SHCREATEPROCESSINFOW
{
    DWORD cbSize;
    ULONG fMask;
    HWND hwnd;
    LPCWSTR  pwszFile;
    LPCWSTR  pwszParameters;
    LPCWSTR  pwszCurrentDirectory;
    HANDLE hUserToken;
    LPSECURITY_ATTRIBUTES lpProcessAttributes;
    LPSECURITY_ATTRIBUTES lpThreadAttributes;
    BOOL bInheritHandles;
    DWORD dwCreationFlags;
    LPSTARTUPINFOW lpStartupInfo;
    LPPROCESS_INFORMATION lpProcessInformation;
} SHCREATEPROCESSINFOW, *PSHCREATEPROCESSINFOW;
```

Members

cbSize

[in] Size, in bytes, of this structure.

fMask

[in] Array of flags that indicates the content and validity of the other structure members. This can be a combination of the following values:

Flag	Description
SEE_MASK_CLASSKEY	Use the file's class registry key.
SEE_MASK_CLASSNAME	Use the file's class name.
SEE_MASK_CONNECTNETDRV	Validate the share and connect to a drive letter. The **pwszFile** member is a Universal Naming Convention (UNC) path of a file on a network.
SEE_MASK_DOENVSUBST	Expand any environment variables.
SEE_MASK_FLAG_DDEWAIT	Wait for the DDE conversation to terminate before returning.
SEE_MASK_FLAG_NO_UI	Do not display an error message box if an error occurs.
SEE_MASK_HMONITOR	Use this flag when specifying a monitor on multimonitor systems.

(continued)

(continued)

Flag	Description
SEE_MASK_NOCLOSEPROCESS	The application will close the process. If the **lpProcessInformation** member is a valid **PROCESS_INFORMATION** pointer, and SEE_MASK_NOCLOSEPROCESS is set, the process will remain open when **SHCreateProcessAsUser** returns. The **hProcess** and **hThread** members of the **PROCESS_INFORMATION** structure will hold the process and thread handles, respectively. This flag typically is set to allow an application to find out when a process created with **SHCreateProcessAsUser** terminates. In some cases, such as when execution is satisfied through a DDE conversation, no handle will be returned. The calling application is responsible for closing the handle when it is no longer needed. If this flag is not set, the process will be closed before SHCreateProcessAsUser returns, even if **lpProcessInformation** is a valid pointer.
SEE_MASK_NO_CONSOLE	Create a console for the new process instead of having it inherit the parent's console. It is equivalent to using a CREATE_NEW_CONSOLE flag with **CreateProcess**.
SEE_MASK_UNICODE	Indicates a Unicode application.

hwnd

[in] Parent window handle.

pwszFile

[in] Pointer to a NULL-terminated Unicode string that specifies the executable file on which **SHCreateProcessAsUser** will perform the action specified by the **runas** verb. The **runas** verb must be supported by the file's class.

Note If the path is not included with the file name, the current directory is assumed.

pwszParameters

[in] Pointer to a NULL-terminated Unicode string containing the application parameters. The parameters must be separated by spaces. To include double quotation marks, you must enclose each mark in a pair of quotation marks, as in the following example:

```
sei.lpParameters = "An example: \"\"\"quoted text\"\"\"";
```

In this case, the application receives three parameters: An, example:, and “quoted text”.

pwszCurrentDirectory

[in] NULL-terminated Unicode string that contains the current directory.

hUserToken

[in] Access token that can be used to represent a particular user. It is needed when there are multiple users for those folders that are treated as belonging to a single user. The caller must have appropriate security privileges for the particular user, including TOKEN_QUERY and TOKEN_IMPERSONATE, and the user’s registry hive currently must be mounted. For further discussion of access control issues, see *Access Control*.

lpProcessAttributes

[in] Pointer to a **SECURITY_ATTRIBUTES** structure with the security descriptor for the new process. It also specifies whether a child process can be inherited. If this parameter is set to NULL, the process will have a default security descriptor and the handle cannot be inherited.

lpThreadAttributes

[in] Pointer to a **SECURITY_ATTRIBUTES** structure with the security descriptor for the new thread. It also specifies whether a child process can be inherited. If this parameter is set to NULL, the process will have a default security descriptor and the handle cannot be inherited.

bInheritHandles

[in] Indicator as to whether the new process inherits handles from the calling process. If set to TRUE, each inheritable open handle in the calling process is inherited by the new process. Inherited handles have the same value and access privileges as the original handles.

dwCreationFlags

[in] Flags that control the creation of the process and the priority class. For a list of the available flags, see **CreateProcessAsUser**.

lpStartupInfo

[in] Pointer to a **STARTUPINFO** structure that specifies how the main window for the new process should appear.

lpProcessInformation

[out] Pointer to a **PROCESS_INFORMATION** structure that receives information about the new process. Set this member to a valid structure pointer, and set the SEE_MASK_NOCLOSEPROCESS flag in the **fMask** member, and the process will remain open when the function returns. The structure’s **hProcess** and **hThread** members then will hold the process and thread handles, respectively. Set this member to NULL, and the process will be closed before the function returns.

QuickInfo

Version 5.00 and later of shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.
Import Library: shell32.lib.

See Also

SHELLEXECUTEINFO

SHDESCRIPTIONID

Receives item data in response to a call to **SHGetDataFromIDList**.

```
typedef struct SHDESCRIPTIONID{
    DWORD   dwDescriptionId;
    CLSID   clsid;
} SHDESCRIPTIONID, *LPSHDESCRIPTIONID;
```

Members

dwDescriptionId

Receives a value that determines what type the item is. This can be one of the following values:

SHDID_COMPUTER_CDROM	The item is a CD-ROM drive.
SHDID_COMPUTER_DRIVE35	The item is a 3.5-inch floppy drive.
SHDID_COMPUTER_DRIVE525	The item is a 5.25-inch floppy drive.
SHDID_COMPUTER_FIXED	The item is a fixed disk drive.
SHDID_COMPUTER_NETDRIVE	The item is a drive that is mapped to a network share.
SHDID_COMPUTER_OTHER	The item is an unidentified system device.
SHDID_COMPUTER_RAMDISK	The item is a RAM disk.
SHDID_COMPUTER_REMOVABLE	The item is a removable disk drive.
SHDID_FS_DIRECTORY	The item is a folder.
SHDID_FS_FILE	The item is a file.
SHDID_FS_OTHER	The item is an unidentified item in the file system.
SHDID_NET_DOMAIN	The item is a network domain.
SHDID_NET_OTHER	The item is an unidentified network resource.

SHDID_NET_RESTOFNET	Not currently used.
SHDID_NET_SERVER	The item is a network server.
SHDID_NET_SHARE	The item is a network share.
SHDID_ROOT_REGITEM	The item is a registered item on the desktop.

clsid

Receives the CLSID of the object to which the item belongs.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

SHDRAGIMAGE

Contains the information needed to create a drag image.

```
typedef struct
{
    SIZE        sizeDragImage;
    POINT       ptOffset;
    HBITMAP     hbmpDragImage;
    COLORREF    crColorKey;
} SHDRAGIMAGE, *LPSHDRAGIMAGE;
```

Members

sizeDragImage

SIZE structure with the length and width of the drag image.

ptOffset

POINT structure that specifies the location of the cursor within the drag image. The structure should contain the offset from the upper-left corner of the drag image to the location of the cursor.

hbmpDragImage

Drag image's bitmap handle.

crColorKey

Color used by the control to fill the background of the drag image.

Remarks

Use the following procedure to create the drag image:

1. Create a bitmap of the size specified by **sizeDragImage** with an HDC that is compatible with the screen.
2. Draw the bitmap.
3. Select the bitmap out of the HDC with which it was created.
4. Destroy the HDC.
5. Assign the bitmap handle to **hbmpDragImage**.

Note Turn off antialiasing when drawing text. Otherwise, artifacts could occur at the edges, between the text color and the color key.

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

+ See Also

IDragSourceHelper::InitializeFromBitmap, **IDragSourceHelper::InitializeFromWindow**

SHELLEXECUTEINFO

Contains information used by **ShellExecuteEx**.

```
typedef struct _SHELLEXECUTEINFO{
    DWORD cbSize;
    ULONG fMask;
    HWND hwnd;
    LPCTSTR lpVerb;
    LPCTSTR lpFile;
    LPCTSTR lpParameters;
    LPCTSTR lpDirectory;
    int nShow;
    HINSTANCE hInstApp;

    // Optional members
    LPVOID lpIDList;
    LPCSTR lpClass;
    HKEY hkeyClass;
    DWORD dwHotKey;
    union {
        HANDLE hIcon;
        HANDLE hMonitor;
    };
    HANDLE hProcess;
} SHELLEXECUTEINFO, *LPSHELLEXECUTEINFO;
```

Members

cbSize

Size of the structure, in bytes.

fMask

Array of flags that indicate the content and validity of the other structure members. This can be a combination of the following values:

Flag	Description
SEE_MASK_CLASSKEY	Use the class key given by the **hkeyClass** member.
SEE_MASK_CLASSNAME	Use the class name given by the **lpClass** member.
SEE_MASK_CONNECTNETDRV	Validate the share and connect to a drive letter. The **lpFile** member is a Universal Naming Convention (UNC) path of a file on a network.
SEE_MASK_DOENVSUBST	Expand any environment variables specified in the string given by the **lpDirectory** or **lpFile** member.
SEE_MASK_FLAG_DDEWAIT	Wait for the DDE conversation to terminate before returning (if the **ShellExecuteEx** function causes a DDE conversation to start).
SEE_MASK_FLAG_NO_UI	Do not display an error message box if an error occurs.
SEE_MASK_HMONITOR	Use this flag when specifying a monitor on multimonitor systems. The monitor is specified in the **hMonitor** member.
SEE_MASK_HOTKEY	Use the hot key given by the **dwHotKey** member.
SEE_MASK_ICON	Use the icon given by the **hIcon** member.
SEE_MASK_IDLIST	Use the item identifier list given by the **lpIDList** member. The **lpIDList** member must point to an **ITEMIDLIST** structure.

(continued)

(continued)

Flag	Description
SEE_MASK_INVOKEIDLIST	Use the **IContextMenu** interface of the selected item's **context menu handler**. Use either lpFile to identify the item by its file-system path, or lpIDList to identify the item by its PIDL. This flag allows applications to use **ShellExecuteEx** to invoke verbs from context menu extensions, instead of the static verbs listed in the registry.
SEE_MASK_NOCLOSEPROCESS	The **hProcess** member receives the process handle. This handle is typically used to allow an application to find out when a process created with **ShellExecuteEx** terminates. In some cases, such as when execution is satisfied through a DDE conversation, no handle will be returned. The calling application is responsible for closing the handle when it is no longer needed.
SEE_MASK_NO_CONSOLE	Create a console for the new process instead of having it inherit the parent's console. It is equivalent to using a CREATE_NEW_CONSOLE flag with **CreateProcess**.
SEE_MASK_UNICODE	Use this flag to indicate a Unicode application.

hwnd

Window handle to any message boxes that the system may produce while executing this function.

lpVerb

A string, referred to as a *verb*, that specifies the action to be performed. The set of available verbs depends on the particular file or folder. It includes the commands listed in the context menu and the registry. See *Extending Context Menus* for further discussion of context menus. The following verbs are usually valid:

Verb	Description
edit	Opens an editor. If *lpFile* is not a document file, the function will fail.
explore	The function explores the folder specified by *lpFile.*
open	The function opens the file specified by the *lpFile* parameter. The file can be an executable file or a document file. It also can be a folder.
print	The function prints the document file specified by *lpFile*. If *lpFile* is not a document file, the function will fail.
properties	Displays the file or folder's properties.

If you set this paramater to NULL:

- For systems prior to Windows 2000, the “open” verb is used if available. If not, the *default verb* is used.
- For Windows 2000 and later systems, “open” is used if available. If not, the default verb is used. If neither verb is available, the system uses the first verb listed in the registry.

lpFile

Address of a null-terminated string that specifies the name of the file on which **ShellExecuteEx** will perform the action specified by the *lpVerb* parameter. The system registry verbs that are supported by the **ShellExecuteEx** function include “open” for executable files and document files and “print” for document files for which a print handler has been registered. Other applications may have added shell verbs through the system registry, such as “play” for AVI and WAV files.

Note If the path is not included with the name, the current directory is assumed.

lpParameters

Address of a null-terminated string containing the application parameters. The parameters must be separated by spaces. To include double quotation marks, you must enclose each mark in a pair of quotation marks, as in the following example:

```
sei.lpParameters = "An example: \"\"\"quoted text\"\"\"";
```

In this case, the application receives three parameters: An, example:, and “quoted text”.

If the **lpFile** member specifies a document file, this member should be NULL.

lpDirectory

Address of a null-terminated string that specifies the name of the working directory. If this member is not specified, the current directory is used as the working directory.

nShow

Flags that specify how an application is to be shown when it is opened. It can be one of the SW_ values listed for the **ShellExecute** function. If **lpFile** specifies a document file, the flag is simply passed to the associated application. It is up to the application to decide how to handle it.

hInstApp

If the function succeeds, it sets this member to a value greater than 32. If the function fails, it is set to an SE_ERR_XXX error value that indicates the cause of the failure. Although **hInstApp** is declared as an HINSTANCE for compatibility with 16-bit Microsoft Windows applications, it is not a true HINSTANCE. It only can be cast to an integer, and compared to either 32 or the SE_ERR_XXX error codes:

Error value	Description
SE_ERR_ACCESSDENIED	Access denied.
SE_ERR_ASSOCINCOMPLETE	File association information not complete.
SE_ERR_DDEBUSY	DDE operation is busy.
SE_ERR_DDEFAIL	DDE operation failed.
SE_ERR_DDETIMEOUT	DDE operation timed out.
SE_ERR_DLLNOTFOUND	Dynamic-link library not found.
SE_ERR_FNF	File not found.
SE_ERR_NOASSOC	File association not available.
SE_ERR_OOM	Out of memory.
SE_ERR_PNF	Path not found.
SE_ERR_SHARE	Cannot share an open file.

lpIDList

Address of an **ITEMIDLIST** structure to contain an item-identifier list uniquely identifying the file to execute. This member is ignored if the **fMask** member is not set to SEE_MASK_IDLIST.

lpClass

Address of a null-terminated string specifying the name of a file class or a globally unique identifier (GUID). This member is ignored if **fMask** is not set to SEE_MASK_CLASSNAME.

hkeyClass

Handle to the registry key for the file class. This member is ignored if **fMask** is not set to SEE_MASK_CLASSKEY.

dwHotKey

Hot key to associate with the application. The low-order word is the virtual key code, and the high-order word is a modifier flag (HOTKEYF_). For a list of modifier flags, see the description of the **WM_SETHOTKEY** message. This member is ignored if **fMask** is not set to SEE_MASK_HOTKEY.

hIcon

Handle to the icon for the file class. This member is ignored if **fMask** is not set to SEE_MASK_ICON.

hProcess

Handle to the newly started application. This member is set on return and is always NULL unless **fMask** is set to SEE_MASK_NOCLOSEPROCESS. Even if **fMask** is set to SEE_MASK_NOCLOSEPROCESS, *hProcess* will be NULL if no process was launched. For example, if a document to be launched is a URL and an instance of Microsoft Internet Explorer already is running, it will display the document. No new process is launched, and *hProcess* will be NULL.

Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in shellapi.h.

SHELLFLAGSTATE

Contains a set of flags that indicate the current shell settings. This structure is used with the **SHGetSettings** function.

```
typedef struct {
    BOOL fShowAllObjects : 1;
    BOOL fShowExtensions : 1;
    BOOL fNoConfirmRecycle : 1;
    BOOL fShowSysFiles : 1;
    BOOL fShowCompColor : 1;
    BOOL fDoubleClickInWebView : 1;
    BOOL fDesktopHTML : 1;
    BOOL fWin95Classic : 1;
    BOOL fDontPrettyPath : 1;
    BOOL fShowAttribCol : 1;
    BOOL fMapNetDrvBtn : 1;
    BOOL fShowInfoTip : 1;
    BOOL fHideIcons : 1;
    UINT fRestFlags : 3;
} SHELLFLAGSTATE, * LPSHELLFLAGSTATE;
```

Members

fShowAllObjects

Nonzero if the **Show All Files** option is enabled, or zero otherwise.

fShowExtensions

Nonzero if the **Hide File Extensions for Known File Types** option is disabled, or zero otherwise.

fNoConfirmRecycle

Nonzero if the **Display Delete Confirmation** dialog box in the Recycle Bin is enabled, or zero otherwise.

fShowSysFiles

Nonzero if the **Do Not Show Hidden Files** option is selected, or zero otherwise.

fShowCompColor

Nonzero if the **Display Compressed Files and Folders with Alternate Color** option is enabled, or zero otherwise.

fDoubleClickInWebView
Nonzero if the **Double-Click to Open an Item** option is enabled, or zero otherwise.

fDesktopHTML
Nonzero if the **Active Desktop**, **View as Web Page** option is enabled, or zero otherwise.

fWin95Classic
Nonzero if the **Classic Style** option is enabled, or zero otherwise.

fDontPrettyPath
Nonzero if the **Allow All Uppercase Names** option is enabled, or zero otherwise.

fShowAttribCol
Nonzero if the **Show File Attributes in Detail View** option is enabled, or zero otherwise.

fMapNetDrvBtn
Nonzero if the **Show Map Network Drive Button in Toolbar** option is enabled, or zero otherwise.

fShowInfoTip
Nonzero if the **Show Info Tips for Items in Folders & Desktop** option is enabled, or zero otherwise.

fHideIcons
Not used.

fRestFlags
Not used.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

SHFILEINFO

Contains information about a file object.

```
typedef struct _SHFILEINFO{
    HICON hIcon;
    int   iIcon;
    DWORD dwAttributes;
    TCHAR szDisplayName[MAX_PATH];
    TCHAR szTypeName[80];
} SHFILEINFO;
```

Members

hIcon

Handle to the icon that represents the file. You are responsible for destroying this handle with **DestroyIcon** when you no longer need it.

iIcon

Index of the icon image within the system image list.

dwAttributes

Array of values that indicates the attributes of the file object. For information about these values, see the **IShellFolder::GetAttributesOf** method.

szDisplayName

String that contains the name of the file as it appears in the Windows shell, or the path and file name of the file that contains the icon representing the file.

szTypeName

String that describes the type of file.

Remarks

This structure is used with the **SHGetFileInfo** function.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in shellapi.h.

SHFILEOPSTRUCT

Contains information that the **SHFileOperation** function uses to perform file operations.

```
typedef struct _SHFILEOPSTRUCT{
    HWND hwnd;
    UINT wFunc;
    LPCTSTR pFrom;
    LPCTSTR pTo;
    FILEOP_FLAGS fFlags;
    BOOL fAnyOperationsAborted;
    LPVOID hNameMappings;
    LPCSTR lpszProgressTitle;
} SHFILEOPSTRUCT, *LPSHFILEOPSTRUCT;
```

Members

hwnd

Window handle to the dialog box to display information about the status of the file operation.

wFunc

Value that indicates which operation to perform. This member can be one of the following values:

FO_COPY

Copy the files specified in the **pFrom** member to the location specified in the **pTo** member.

FO_DELETE

Delete the files specified in **pFrom**.

FO_MOVE

Move the files specified in **pFrom** to the location specified in **pTo**.

FO_RENAME

Rename the file specified in **pFrom**. You cannot use this flag to rename multiple files with a single function call. Use FO_MOVE, instead.

pFrom

Address of a buffer to specify one or more source file names. Multiple names must be null-separated. The list of names must be double null-terminated. This member can specify a wild-card path.

pTo

Address of a buffer to contain the name of the destination file or directory. This parameter must be set to NULL if it is not used. The name string must meet the following specifications:

- Wildcard characters are not supported.
- **Copy** and **Move** operations can specify destination directories that do not exist and the system will attempt to create them. The system normally displays a dialog box to ask the user if they want to create the new directory. To suppress this dialog box and have the directories created silently, set the FOF_NOCONFIRMDIR flag in **fFiles**.
- For **Copy** and **Move** operations, the buffer can contain multiple destination file names if the **fFlags** member specifies FOF_MULTIDESTFILES.
- Multiple names must be null-separated. The list of names must be double null-terminated.
- Use fully qualified path names with long file names. If you use a relative path name, long file names will be truncated to the 8.3 format. For example, if you set *pFrom* = "New Text Document.txt" and *pTo* = "temp\0", the file will appear in the temp subdirectory as NewTextD.txt.

fFlags

Flags that control the file operation. This member can take a combination of the following flags:

Flag	Description
FOF_ALLOWUNDO	Preserve **Undo** information, if possible. If **pFrom** does not contain fully qualified path and file names, this flag is ignored.
FOF_FILESONLY	Perform the operation on files only if a wildcard file name (*.*) is specified.
FOF_MULTIDESTFILES	The **pTo** member specifies multiple destination files (one for each source file), instead of one directory where all source files are to be deposited.
FOF_NOCONFIRMATION	Respond with "Yes to All" for any dialog box that is displayed.
FOF_NOCONFIRMMKDIR	Do not confirm the creation of a new directory if the operation requires one to be created.
FOF_NO_CONNECTED_ELEMENTS	**Version 5.0.** Do not move connected files as a group. Only move the specified files.
FOF_NOCOPYSECURITYATTRIBS	**Version 4.71.** Do not copy the security attributes of the file.
FOF_NOERRORUI	Do not display a user interface if an error occurs.
FOF_NORECURSION	Only operate in the local directory. Do not operate recursively into subdirectories.
FOF_RENAMEONCOLLISION	Give the file being operated on a new name in a move, copy, or rename operation if a file with the target name already exists.
FOF_SILENT	Do not display a progress dialog box.
FOF_SIMPLEPROGRESS	Display a progress dialog box but do not show the file names.
FOF_WANTMAPPINGHANDLE	If FOF_RENAMEONCOLLISION is specified and any files were renamed, assign a name mapping object containing their old and new names to the **hNameMappings** member.
FOF_WANTNUKEWARNING	**Version 5.0.** Send a warning if a file is being destroyed during a delete operation instead of recycled. This flag partially overrides FOF_NOCONFIRMATION.

fAnyOperationsAborted

Value that receives TRUE if the user cancelled any file operations before they were completed, or FALSE otherwise.

hNameMappings

A handle to a name mapping object containing the old and new names of the renamed files. This member is used only if the **fFlags** member includes the FOF_WANTMAPPINGHANDLE flag. Treat **hNameMappings** as a pointer to a structure whose first member is an INT value, followed by an array of **SHNAMEMAPPING** structures. The INT value will be set to the number of structures in the array. Each **SHNAMEMAPPING** structure will contain the old and new path name for one of the renamed files.

Note The handle must be freed with **SHFreeNameMappings**.

lpszProgressTitle

Address of a string to use as the title of a progress dialog box. This member is used only if **fFlags** includes the FOF_SIMPLEPROGRESS flag.

Remarks

If the **pFrom** or **pTo** members are unqualified names, the current directories are taken from the global current drive and directory settings, as managed by the **GetCurrentDirectory** and **SetCurrentDirectory** functions.

If **pFrom** is set to a filename, deleting the file with FO_DELETE will not move it to the Recycle Bin, even if the FOF_ALLOWUNDO flag is set. You must use a full pathname.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.

SHITEMID

Defines an item identifier.

```
typedef struct _SHITEMID {
    USHORT cb;
    BYTE   abID[1];
} SHITEMID, * LPSHITEMID;
typedef const SHITEMID * LPCSHITEMID;
```

Members

cb

Size of identifier, in bytes, including **cb** itself.

abID

Variable-length item identifier.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

SHNAMEMAPPING

Contains the old and new path names for each file that was moved, copied, or renamed by the **SHFileOperation** function.

```
typedef struct _SHNAMEMAPPING {
    LPSTR pszOldPath;
    LPSTR pszNewPath;
    int   cchOldPath;
    int   cchNewPath;
} SHNAMEMAPPING, *LPSHNAMEMAPPING;
```

Members

pszOldPath

Address of a character buffer that contains the old path name.

pszNewPath

Address of a character buffer that contains the new path name.

cchOldPath

Number of characters in **pszOldPath**.

cchNewPath

Number of characters in **pszNewPath**.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.

See Also

SHFILEOPSTRUCT

SHQUERYRBINFO

Contains the size and item count information retrieved by the **SHQueryRecycleBin** function.

```
typedef struct _SHQUERYRBINFO {
    DWORD cbSize;
    __int64 i64Size;
    __int64 i64NumItems;
} SHQUERYRBINFO, *LPSHQUERYRBINFO;
```

Members

cbSize

Size of the structure, in bytes. This member must be filled in prior to calling the function.

i64Size

Total size of all the objects in the specified Recycle Bin, in bytes.

i64NumItems

Total number of items in the specified Recycle Bin.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.

STRRET

Contains strings returned from the **IShellFolder** interface methods.

```
typedef struct _STRRET {
    UINT uType;
    union
    {
        LPWSTR pOleStr;
        LPSTR pStr;
        UINT uOffset;
        char cStr[MAX_PATH];
    } DUMMYUNIONNAME;
} STRRET, *LPSTRRET;
```

Members

uType

Value that specifies the desired format of the string. This can be one of the following values:

STRRET_CSTR	The string is returned in the **cStr** member.
STRRET_OFFSET	The **uOffset** member value indicates the number of bytes from the beginning of the item identifier list where the string is located.
STRRET_WSTR	The string is at the address pointed to, in the **pOleStr** member.

pOleStr

Address of the OLE string. This memory must be allocated with the shell's allocator (see **SHGetMalloc**). It is the calling application's responsibility to free this memory when it is no longer needed. The shell's allocator must be used to free the memory.

pStr

This member is not used.

uOffset

Offset into item identifier list.

cStr

Buffer to receive the display name.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

See Also

IShellFolder::GetDisplayNameOf

SV2CVW2_PARAMS

Holds the parameters for the **IShellView2::CreateViewWindow2** method.

```
typedef struct _SV2CVW2_PARAMS
{
    DWORD cbSize;
    IShellView *psvPrev;
    FOLDERSETTINGS const *pfs;
    IShellBrowser *psbOwner;
    RECT *prcView;
    SHELLVIEWID const *pvid;
    HWND hwndView;
} SV2CVW2_PARAMS, *LPSV2CVW2_PARAMS;
```

Members

cbSize
Size of the structure.

psvPrev
Pointer to the **IShellView** interface of the previous view. A Shell View can use this parameter to communicate with a previous view with the same implementation. It also can be used to optimize browsing between like views. This parameter can be NULL.

pfs
FOLDERSETTINGS structure with information needed to create the view.

psbOwner
Pointer to the current instance of the **IShellBrowser** interface of the parent shell browser. **CreateViewWindow2** should call this interface's **AddRef** method and store the interface pointer. It can be used for communication with the Windows Explorer window.

prcView
RECT structure that defines the view's display region.

pvid
View mode's globally unique identifier (GUID).

hwndView
Window handle for the new Shell View.

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with
Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

URLINVOKECOMMANDINFO

Contains information for use with the **IUniformResourceLocator::InvokeCommand** method.

```
typedef struct {
    DWORD   dwcbSize;
    DWORD   dwFlags;
    HWND    hwndParent;
    LPCSTR  pcszVerb;
} URLINVOKECOMMANDINFO, *PURLINVOKECOMMANDINFO;
```

Members

dwcbSize

Size of this structure, in bytes.

dwFlags

Flag value that specifies how the **IUniformResourceLocator::InvokeCommand** method will execute. This value can be a combination of the following:

IURL_INVOKECOMMAND_FL_ALLOW_UI

Interaction with the user is allowed and the **hwndParent** member of this structure is valid. If this is not set, interaction with the user is not allowed, and the **hwndParent** member is ignored.

IURL_INVOKECOMMAND_FL_USE_DEFAULT_VERB

Default verb for the Internet Shortcut's protocol is to be used, and the **pcszVerb** member is ignored. If this bit is not set, the verb is specified by **pcszVerb**.

hwndParent

Handle to the parent window. If **dwFlags** is set to IURL_INVOKECOMMAND_FL_USE_DEFAULT_VERB, this member is ignored.

pcszVerb

Address of a zero-terminated string that contains the verb to be invoked by **IUniformResourceLocator::InvokeCommand**. If **dwFlags** is set to IURL_INVOKECOMMAND_FL_USE_DEFAULT_VERB, this member is ignored.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in intshcut.h.

WALLPAPEROPT

Contains the wallpaper options.

```
typedef struct _tagWALLPAPEROPT {
    DWORD dwSize;
    DWORD dwStyle;
} WALLPAPEROPT;
```

Members

dwSize

Unsigned long integer value that contains the size of the **WALLPAPEROPT** structure.

dwStyle

Unsigned long integer value that contains the wallpaper style. It can be one of the following values:

- **WPSTYLE_CENTER**
- **WPSTYLE_TILE**
- **WPSTYLE_STRETCH**
- **WPSTYLE_MAX**

! Requirements

Version 4.71 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlobj.h.

CHAPTER 10

Shell Enumerations and Macros

Shell Enumerations

ASSOCDATA

Used by **IQueryAssociations::GetData** to define the type of data that is to be returned.

```
typedef enum {
    ASSOCDATA_MSIDESCRIPTOR = 1,
    ASSOCDATA_NOACTIVATEHANDLER,
    ASSOCDATA_QUERYCLASSSTORE,
} ASSOCDATA;
```

Members

ASSOCDATA_MSIDESCRIPTOR
Component descriptor to pass to the Microsoft Installation API.

ASSOCDATA_NOACTIVATEHANDLER
Attempts to activate a window are restricted. There is no data associated with this value.

ASSOCDATA_QUERYCLASSSTORE
If this value is present, applications should check the Microsoft Windows 2000 class store.

ASSOCF

Used to provide information to the **IQueryAssociations** interface methods.

```
enum {
    ASSOCF_INIT_BYEXENAME,
    ASSOCF_OPEN_BYEXENAME,
    ASSOCF_INIT_DEFAULTTOSTAR,
    ASSOCF_INIT_DEFAULTTOFOLDER,
    ASSOCF_NOUSERSETTINGS,
    ASSOCF_NOTRUNCATE,
```

(continued)

(continued)

```
    ASSOCF_VERIFY,
    ASSOCF_REMAPRUNDLL,
    ASSOCF_NOFIXUPS,
    ASSOCF_IGNOREBASECLASS
};

typedef DWORD ASSOCF;
```

Members

ASSOCF_INIT_BYEXENAME

Used when the *pwszAssoc* parameter of **IQueryAssociations::Init** is set to an executable file name. If this flag is not set in this case, the root key will be set to the ProgID associated with the **.exe** key instead of the executable file's ProgID.

ASSOCF_OPEN_BYEXENAME

This value is identical to ASSOCF_INIT_BYEXENAME.

ASSOCF_INIT_DEFAULTTOSTAR

If an **IQueryAssociations** method does not find the requested value under the root key, it will attempt to retrieve the comparable value from the * subkey.

ASSOCF_INIT_DEFAULTTOFOLDER

If an **IQueryInterface** method does not find the requested value under the root key, it will attempt to retrieve the comparable value from the **Folder** subkey.

ASSOCF_NOUSERSETTINGS

Only search HKEY_CLASSES_ROOT. Ignore HKEY_CURRENT_USER.

ASSOCF_NOTRUNCATE

Don't truncate the return string. Return an error value and the required size for the complete string.

ASSOCF_VERIFY

Verify that data is accurate. This setting allows **IQueryAssociations** methods to read data from the user's hard drive. For example, they can check the friendly name in the registry against the one stored in the .exe file. Setting this flag normally reduces the efficiency of the method.

ASSOCF_REMAPRUNDLL

IQueryAssociations methods normally return information about the first .exe or .dll in a command string. If a command uses Rundll.exe, setting this flag tells the method to ignore Rundll.exe and return information about its target.

ASSOCF_NOFIXUPS

Do not fix errors in the registry, such as the friendly name of a function not matching the one found in the .exe file.

ASSOCF_IGNOREBASECLASS

Ignore the BaseClass value.

See Also

AssocQueryKey, **AssocQueryString**, **AssocQueryStringByKey**

ASSOCKEY

Used to specify the type of key to be returned by **IQueryAssociations::GetKey**.

```
typedef enum {
    ASSOCKEY_SHELLEXECCLASS = 1,
    ASSOCKEY_APP,
    ASSOCKEY_CLASS,
    ASSOCKEY_BASECLASS,
} ASSOCKEY;
```

Members

ASSOCKEY_SHELLEXECCLASS
Key that is passed to ShellExec(hkeyClass).

ASSOCKEY_APP
Application key for the file class.

ASSOCKEY_CLASS
ProgID or class key.

ASSOCKEY_BASECLASS
BaseClass value.

ASSOCSTR

Used by **IQueryAssociations::GetString** to define the type of string that is to be returned.

```
typedef enum {
    ASSOCSTR_COMMAND,
    ASSOCSTR_EXECUTABLE,
    ASSOCSTR_FRIENDLYDOCNAME,
    ASSOCSTR_FRIENDLYAPPNAME,
    ASSOCSTR_NOOPEN,
    ASSOCSTR_SHELLNEWVALUE,
    ASSOCSTR_DDECOMMAND,
    ASSOCSTR_DDEIFEXEC,
    ASSOCSTR_DDEAPPLICATION,
    ASSOCSTR_DDETOPIC,
} ASSOCSTR;
```

Members

ASSOCSTR_COMMAND
Command string associated with a shell verb.

ASSOCSTR_EXECUTABLE
Executable from a shell verb command string. If the command uses Rundll.exe, set the ASSOCF_REMAPRUNDLL flag in the *flags* parameter of **IQueryAssociations::GetString** to get the target executable.

ASSOCSTR_FRIENDLYDOCNAME
Friendly name of a document type.

ASSOCSTR_FRIENDLYAPPNAME
Friendly name of an executable.

ASSOCSTR_NOOPEN
Ignore the information associated with the **open** subkey.

ASSOCSTR_SHELLNEWVALUE
Look under the **ShellNew** subkey.

ASSOCSTR_DDECOMMAND
Template for DDE commands.

ASSOCSTR_DDEIFEXEC
DDECOMMAND to use to just create a process.

ASSOCSTR_DDEAPPLICATION
Application name in a DDE broadcast.

ASSOCSTR_DDETOPIC
Topic name in a DDE broadcast.

FOLDERFLAGS

Set of flags used to specify folder view options. They are independent of each other, and can be used in any combination.

```
typedef enum{
    FWF_AUTOARRANGE,
    FWF_ABBREVIATEDNAMES,
    FWF_SNAPTOGRID,
    FWF_OWNERDATA,
    FWF_BESTFITWINDOW,
    FWF_DESKTOP,
    FWF_SINGLESEL,
    FWF_NOSUBFOLDERS,
    FWF_TRANSPARENT,
    FWF_NOCLIENTEDGE,
    FWF_NOSCROLL,
    FWF_ALIGNLEFT,
    FWF_NOICONS,
    FWF_SHOWSELALWAYS
    FWF_SINGLECLICKACTIVATE,
} FOLDERFLAGS;
```

Members

FWF_AUTOARRANGE

Automatically arrange the elements in the view. This implies LVS_AUTOARRANGE if the list view control is used to implement the view.

FWF_ABBREVIATEDNAMES

Names should be abbreviated. This flag is not currently supported.

FWF_SNAPTOGRID

Items should be arranged on a grid. This flag is not currently supported.

FWF_OWNERDATA

This flag is not currently supported.

FWF_BESTFITWINDOW

Enable the best-fit window mode. Let the view size the window so that its contents fit inside the view window in the best possible manner.

FWF_DESKTOP

Make the folder behave like the desktop. This value applies only to the desktop view and is not used for typical shell folders.

FWF_SINGLESEL

Do not allow more than a single item to be selected. This is used in the common dialogs.

FWF_NOSUBFOLDERS

Do not show subfolders.

FWF_TRANSPARENT

Draw transparently. This is used only for the desktop.

FWF_NOCLIENTEDGE

Do not add the WS_EX_CLIENTEDGE value to the view. This is used only for the desktop.

FWF_NOSCROLL

Do not add scroll bars. This is used only for the desktop.

FWF_ALIGNLEFT

The view should be left-aligned. This implies LVS_ALIGNLEFT if the list view control is used to implement the view.

FWF_NOICONS

The view should not display icons.

FWF_SHOWSELALWAYS

Always show the selection.

FWF_SINGLECLICKACTIVATE

This flag is not currently supported.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

FOLDERVIEWMODE

Set of constants used to specify the folder view type.

```
typedef enum {
    FVM_ICON      = 1,
    FVM_SMALLICON = 2,
    FVM_LIST      = 3,
    FVM_DETAILS   = 4
} FOLDERVIEWMODE;
```

Members

FVM_ICON
The view should display large icons.

FVM_SMALLICON
The view should display small icons.

FVM_LIST
Object names are displayed in a list view.

FVM_DETAILS
Object names and other selected information, such as the size or date last updated, are shown.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

IURL_SETURL_FLAGS

The values of this enumeration are used with the **IUniformResourceLocator::SetURL** to specify the protocol scheme.

```
typedef enum iurl_seturl_flags{
    IURL_SETURL_FL_GUESS_PROTOCOL,
    IURL_SETURL_FL_USE_DEFAULT_PROTOCOL
} IURL_SETURL_FLAGS;
```

Members

IURL_SETURL_FL_GUESS_PROTOCOL

If the protocol scheme is not specified in the *pcszURL* parameter to **IUniformResourceLocator::SetURL**, the system automatically chooses a scheme and adds it to the URL.

IURL_SETURL_FL_USE_DEFAULT_PROTOCOL

If the protocol scheme is not specified in the *pcszURL* parameter to **IUniformResourceLocator::SetURL**, the system adds the default protocol to the URL.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in intshcut.h.

IURL_SETURL_INVOKECOMMAND_FLAGS

The following enumerated values are used in the **dwFlags** member of the **URLINVOKECOMMANDINFO** structure.

```
typedef enum iurl_invokecommand_flags{
    IURL_INVOKECOMMAND_FL_ALLOW_UI              = 0x0001,
    IURL_INVOKECOMMAND_FL_USE_DEFAULT_VERB      = 0x0002,
} IURL_INVOKECOMMAND_FLAGS;
```

Members

IURL_INVOKECOMMAND_FL_ALLOW_UI

If this bit is set, interaction with the user is allowed and the **hwndParent** member of the **URLINVOKECOMMANDINFO** structure is valid. If this bit is clear, interaction with the user is not allowed and the **hwndParent** member is ignored.

IURL_INVOKECOMMAND_FL_USE_DEFAULT_VERB

If this bit is set, the default verb for the Internet Shortcut's protocol is to be used and the **pcszVerb** member of the **URLINVOKECOMMANDINFO** structure is ignored. If this bit is clear, the verb is specified by **pcszVerb**.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in intshcut.h.

SHCONTF

Determines the type of items included in an enumeration. These values are used with the **IShellFolder::EnumObjects** method.

```
typedef enum tagSHCONTF{
    SHCONTF_FOLDERS,
    SHCONTF_NONFOLDERS,
    SHCONTF_INCLUDEHIDDEN,
    SHCONTF_INIT_ON_FIRST_NEXT,
    SHCONTF_NETPRINTERSRCH,
} SHCONTF;
```

Members

SHCONTF_FOLDERS
Include items that are folders in the enumeration.

SHCONTF_NONFOLDERS
Include items that are not folders in the enumeration.

SHCONTF_INCLUDEHIDDEN
Include hidden items in the enumeration.

SHCONTF_INIT_ON_FIRST_NEXT
IShellFolder::EnumObjects can return without validating the enumeration object. Validation can be postponed until the first call to **IEnumIDList::Next**. This flag is intended to be used when a user-interface may be displayed prior to the first **IEnumIDList::Next** call. For a user-interface to be presented, *hwndOwner* must be set to a valid window handle.

SHCONTF_NETPRINTERSRCH
The caller is looking for printer objects.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.

Windows CE: Unsupported.
Header: Declared in shlobj.h.

SHGNO

Defines the values used with the **IShellFolder::GetDisplayNameOf** and **IShellFolder::SetNameOf** methods.

```
typedef enum tagSHGDN {
    SHGDN_NORMAL,
    SHGDN_INFOLDER,
    SHGDN_FOREDITING,
    SHGDN_FORADDRESSBAR,
    SHGDN_FORPARSING,
} SHGNO;
```

Members

This enumeration consists of two groups of values. The first specifies the name's type and has two possible values:

Value	Description
SHGDN_NORMAL	Full name. The name is relative to the desktop and not to a specific folder. This name will be used for generic display.
SHGDN_INFOLDER	Relative name. The name is relative to the folder that is processing it.

The second group consists of modifiers to the first group that specify name retrieval options. It has the following values:

Value	Description
SHGDN_FOREDITING	The name will be used for in-place editing when the user renames the item.
SHGDN_FORADDRESSBAR	The name will be displayed in an address bar combo box.
SHGDN_FORPARSING	The name will be used for parsing. That is, it can be passed to **ParseDisplayName** to recover the objects PIDL. The form this name takes depends on the particular object.

Remarks

Note while the parsing name returned by file system objects is the object's fully-qualified path, virtual folders may use something quite different. For example, some virtual folders use a GUID as the parsing name and will return a string of the form "::{GUID}". To check whether or not the object is part of the file system, call **IShellFolder::GetAttributesOf** and see if the SFGAO_FILESYSTEM flag is set.

The numeric value of SHGDN_NORMAL is zero, so you cannot test for the presence of this bit. Consider SHGDN_NORMAL a default setting that will be used if no other flag in that group is set.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlobj.h.

TRANSLATEURL_IN_FLAGS

The following enumerated values are used with the **TranslateURL** function to determine how it will execute.

```
typedef enum translateurl_in_flags {
    TRANSLATEURL_FL_GUESS_PROTOCOL,
    TRANSLATEURL_FL_USE_DEFAULT_PROTOCOL,
} TRANSLATEURL_IN_FLAGS;
```

Members

TRANSLATEURL_FL_GUESS_PROTOCOL
If the protocol scheme is not specified in the *pcszURL* parameter to **TranslateURL**, the system automatically chooses a scheme and adds it to the URL.

TRANSLATEURL_FL_USE_DEFAULT_PROTOCOL
If the protocol scheme is not specified in the *pcszURL* parameter to **TranslateURL**, the system adds the default protocol to the URL.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.

Windows CE: Unsupported.
Header: Declared in intshcut.h.

URLASSOCIATIONDIALOG_IN_FLAGS

The following enumerated values are used with **URLAssociationDialog** to determine how it executes.

```
typedef enum urlassociationdialog_in_flags {
    URLASSOCDLG_FL_USE_DEFAULT_NAME ,
    URLASSOCDLG_FL_REGISTER_ASSOC
} URLASSOCIATIONDIALOG_IN_FLAGS;
```

Members

URLASSOCDLG_FL_USE_DEFAULT_NAME
Use the default file name (that is, "Internet Shortcut").

URLASSOCDLG_FL_REGISTER_ASSOC
Register the selected application as the handler for the protocol specified in the *pcszURL* parameter of **URLAssociationDialog**. The application is registered only if this flag is set and the user indicates that a persistent association is desired.

! Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in intshcut.h.

Shell Macros

MAKEDLLVERULL

Used to pack DLL version information into a ULONGLONG value.

```
ULONGLONG MAKEDLLVERULL(
  WORD wMajorVersion,
  WORD wMinorVersion,
  WORD wBuild,
  WORD wQFE
);
```

Parameters

wMajorVersion
The major version number.

wMinorVersion
The minor version number.

wBuild
The build number.

wQFE
The QFE number that identifies the service pack.

Return Values

Returns the version information packed into a ULONGLONG value

Remarks

This macro is used in conjunction with **DllGetVersion** to pack version information into a form that can easily be compared to the **ullVersion** member of a **DLLVERSIONINFO2** structure. It is defined as:

```
#define MAKEDLLVERULL(major, minor, build, qfe) \
        (((ULONGLONG)(major) << 48) | \
         ((ULONGLONG)(minor) << 32) | \
         ((ULONGLONG)(build) << 16) | \
         ((ULONGLONG)(  qfe) <<  0))
```

For most purposes, you only need to assign values to the major and minor version numbers. The remaining two parameters can be set to zero. The following code fragment illustrates how to use **MAKEDLLVERULL** to determine whether a DLL is version 4.71 or later. The VersionInfo structure is the **DLLVERSIONINFO2** structure returned by **DllGetVersion**.

```
if(VersionInfo.ullVersion >= MAKEDLLVERULL(4, 71, 0,0))
{
    ...
}
```

! Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.

SOANGLETENTHS

Sets the angle in tenths of a degree.

```
SOANGLETENTHS(AngleInTenthsOfADegree)
```

Parameters

AngleInTenthsOfADegree
New angle, in tenths of a degree.

! Requirements

Windows NT/2000: Not supported by version 5.0 or later.
Windows: Supported by Windows 95. Not supported by Windows 98.
Windows CE: Unsupported.

SOPALETTEINDEX

Creates a palette-index color value.

```
SOPALETTEINDEX(Index)
```

Parameters

Index
Valid palette entry index. When this color value is used, the system uses the color from the given palette entry.

! Requirements

Windows NT/2000: Not supported by version 5.0 or later.
Windows: Supported by Windows 95. Not supported by Windows 98.
Windows CE: Unsupported.

SOPALETTERGB

Creates a palette-relative RGB color value. When this color value is specified, the system uses the palette entry that has the color that most closely matches this value.

```
SOPALETTERGB(Red, Green, Blue)
```

Parameters

Red
Red color intensity in the range 0 to 255.

Green
Green color intensity in the range 0 to 255.

Blue
Blue color intensity in the range 0 to 255.

! Requirements

Windows NT/2000: Not supported by version 5.0 or later.
Windows: Supported by Windows 95. Not supported by Windows 98.
Windows CE: Unsupported.

SORGB

Creates an RGB color value.

```
SORGB(Red, Green, Blue)
```

Parameters

Red
Red color intensity in the range 0 to 255.

Green
Green color intensity in the range 0 to 255.

Blue
Blue color intensity in the range 0 to 255.

! Requirements

Windows NT/2000: Not supported by version 5.0 or later.
Windows: Supported by Windows 95. Not supported by Windows 98.
Windows CE: Unsupported.

SOSETRATIO

Sets the ratio.

```
SOSETRATIO(Numerator, Denominator)
```

Parameters

Numerator
Numerator of the ratio factor. This value must be in the range 0 to 65,535.

Denominator
Denominator of the ratio factor. This value must be in the range 0 to 65,535.

! Requirements

Windows NT/2000: Not supported by version 5.0 or later.
Windows: Supported by Windows 95. Not supported by Windows 98.
Windows CE: Unsupported.

CHAPTER 11

Shell Lightweight Utility APIs

String Functions

ChrCmpI

Performs a comparison between two characters. The comparison is not case sensitive.

```
BOOL ChrCmpI(
  TCHAR c1,
  TCHAR c2
);
```

Parameters

c1
First character to be compared.

c2
Second character to be compared.

Return Values

Returns zero if the two characters are the same, or nonzero otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

IntlStrEqN

Performs a case-sensitive comparison of a specified number of characters from the beginning of two localized strings.

```
BOOL IntlStrEqN(
  LPCTSTR pszStr1,
  LPCTSTR pszStr2,
  int nChar
);
```

Parameters

pszStr1
  [in] Pointer to a NULL-terminated string.

pszStr2
  [in] Pointer to a NULL-terminated string.

nChar
  [in] Number of characters to be compared, starting from the beginning of the strings.

Return Values

Returns TRUE if the first *nChar* characters are identical, or FALSE otherwise.

Remarks

This function gets the thread locale and uses **CompareString** to do a case-sensitive comparison of the first *nChar* characters. It is equivalent to:

```
IntlStrEqWorker( TRUE, pszStr1, pszStr2, nChar)
```

Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

See Also

IntlStrEqWorker

IntlStrEqNI

Performs a case-insensitive comparison of a specified number of characters from the beginning of two localized strings.

```
BOOL IntlStrEqNI(
  LPCTSTR pszStr1,
  LPCTSTR pszStr2,
  int nChar
);
```

Parameters

pszStr1

[in] Pointer to a NULL-terminated string.

pszStr2

[in] Pointer to a NULL-terminated string.

nChar

[in] Number of characters to be compared, starting from the beginning of the strings.

Return Values

Returns TRUE if the first *nChar* characters are identical, or FALSE otherwise.

Remarks

This function gets the thread locale and uses **CompareString** to do a case-insensitive comparison of the first *nChar* characters. It is equivalent to:

```
IntlStrEqWorker( FALSE, pszStr1, pszStr2, nChar)
```

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

+ See Also

IntlStrEqWorker

IntlStrEqWorker

Compares a specified number of characters from the beginning of two localized strings.

```
BOOL IntlStrEqWorker(
  BOOL fCaseSens,
  LPCTSTR pszStr1,
  LPCTSTR pszStr2,
  int nChar
);
```

Parameters

fCaseSens

[in] Value that is set to TRUE for a case-sensitive comparison, or to FALSE for a case-insensitive comparison.

pszStr1

[in] Pointer to a NULL-terminated string.

pszStr2

[in] Pointer to a NULL-terminated string.

nChar

[in] Number of characters to be compared, starting from the beginning of the strings.

Return Values

Returns TRUE if the first *nChar* characters are identical, or FALSE otherwise.

Remarks

This function gets the thread locale and uses **CompareString** to determine whether the first *nChar* characters are identical.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

MLLoadLibrary

Maps an executable module, specified by its file name and registry subkey, into the address space of the calling function. It supports multilanguage applications by determining the LCID and loading the appropriate DLL or EXE.

```
HINSTANCE MLLoadLibrary(
  LPCTSTR lpLibFileName,
  HANDLE hFile,
  DWORD dwFlags,
  LPCTSTR lpComponent,
  BOOL bCrossCodePage
);
```

Parameters

lpszLibFileName

Null-terminated string with the file name of the EXE or DLL to be loaded. It should not include any path information. **MLLoadLibrary** will extract that from the registry.

hFile

Reserved. NULL must be passed.

dwFlags

Action to be taken when loading the module. This can be one of the following:

DONT_RESOLVE_DLL_REFERENCE
If the executable module is a DLL, the system does not call **DllMain** for process and thread initialization and termination.

LOAD_LIBRARY_AS_DATAFILE
The system maps the file into the calling process's virtual address space as if it were a data file.

lpComponent

Name of the component's subkey. The function assumes that the component is located under HKEY_LOCAL_MACHINE\\Software\\Microsoft\\Windows\\CurrentVersion\\App Paths.

bCrossCodePage

Reserved. Null must be passed.

Return Values

Returns a handle to the module if successful, or NULL otherwise.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.

SHStrDup

Makes a copy of a string in newly allocated memory.

```
HRESULT SHStrDup(
  LPCTSTR pszSource,
  LPWSTR *ppwszTarget
);
```

Parameters

pszSource

[in] Pointer to the string to be copied.

ppwszTarget

[out] Pointer to an allocated Unicode string containing the result. **SHStrDup** allocates memory for this string with **CoTaskMemAllocate**. You should free the string with **CoTaskMemFree** when it is no longer needed.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

This function will take either Unicode or ANSI strings as input, but the copied string is always Unicode.

This function uses **CoTaskMemAlloc** to allocate memory for the copied string. You must free this memory with **CoTaskMemFree** when it is no longer needed.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

+ See Also

StrDup

StrCat

Appends one string to another.

```
LPTSTR StrCat(
  LPTSTR psz1,
  LPCTSTR psz2,
);
```

Parameters

psz1

[in/out] Address of a null-terminated string to be appended to. It must be large enough to hold both strings.

psz2

[in] Address of the string to be appended to *psz1*.

Return Values

Returns a pointer to *psz1*, which holds the combined strings.

! Requirements

Version 4.71 and later of shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with
Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrCatBuff

Copies and appends characters from one string to the end of another.

```
LPTSTR StrCatBuff(
  LPTSTR pszDestination,
  LPCTSTR pszSource,
  int cchDestBuffSize
);
```

Parameters

pszDestination

[in/out] Pointer to a null-terminated string. When the function returns, the characters from *pszSource* will be appended to it.

pszSource

Pointer to the string to be appended to *pszDestination*.

cchDestBuffSize

Size of the buffer pointed to by *pszDestination*. This value must be greater than the length of the combined string, or it will be truncated to fit.

Return Values

Returns a pointer to the destination string.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrChr

Searches a string for the first occurrence of a character that matches the specified character. The comparison is case sensitive.

```
LPTSTR StrChr(
  LPCTSTR lpStart,
  TCHAR wMatch
);
```

Parameters

lpStart

Address of the string to be searched.

wMatch

Character to be used for comparison.

Return Values

Returns the address of the first occurrence of the character in the string if successful, or NULL otherwise.

Remarks

The comparison assumes *lpStart* points to the start of a null-terminated string.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrChrI

Searches a string for the first occurrence of a character that matches the specified character. The comparison is not case sensitive.

```
LPTSTR StrChrI(
  LPCTSTR lpStart,
  TCHAR wMatch
);
```

Parameters

lpStart
Address of the string to be searched.

wMatch
Character to be used for comparison.

Return Values

Returns the address of the first occurrence of the character in the string if successful, or NULL otherwise.

Remarks

The comparison assumes *lpStart* points to the start of a null-terminated string.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.

Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrCmp

Compares two strings to determine if they are the same. The comparison *is* case-sensitive.

```
int StrCmp(
  LPCTSTR lpStr1,
  LPCTSTR lpStr2
);
```

Parameters

lpStr1
[in] Pointer to the first string to be compared.

lpStr2
[in] Pointer to the second string to be compared.

Return Values

Returns zero if the strings are identical. Returns a positive value if the string pointed to by *lpStr1* is greater than that pointed to by *lpStr2*. Returns a negative value if the string pointed to by *lpStr1* is less than that pointed to by *lpStr2*.

Remarks

This function returns the difference in value of the first unequal characters it encounters, or zero if they are all equal. For example, if *lpStr1*=“abczb” and *lpStr2* = “abcdefg”, **StrCmp** determines that “abczb” is greater than “abcdefg” and returns z - d.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrCmpI

Compares two strings to determine if they are the same. The comparison is *not* case-sensitive.

```
int StrCmpI(
  LPCTSTR lpStr1,
  LPCTSTR lpStr2
);
```

Parameters

lpStr1
[in] Address of the first string to be compared.

lpStr2
[in] Address of the second string to be compared.

Return Values

Returns zero if the strings are identical. Returns a positive value if the string pointed to by *lpStr1* is greater than that pointed to by *lpStr2*. Returns a negative value if the string pointed to by *lpStr1* is less than that pointed to by *lpStr2*.

Remarks

This function returns the difference in value of the first unequal characters it encounters, or zero if they are all equal. For example, if *lpStr1*="Abczb" and *lpStr2* = "abCdefg", **StrCmpI** determines that "Abczb" is greater than "abCdefg" and returns z - d.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrCmpN

Compares a specified number of characters from the beginning of two strings to determine if they are the same. The comparison *is* case-sensitive.

```
int StrCmpN(
  LPCTSTR lpStr1,
  LPCTSTR lpStr2,
  int nChar
);
```

Parameters

lpStr1
[in] Pointer to the first string to be compared.

lpStr2
[in] Pointer to the second string to be compared.

nChar
[in] Number of characters from the beginning of each string to be compared.

Return Values

Returns zero if the strings are identical. Returns a positive value if the first *nChar* characters of the string pointed to by *lpStr1*are greater than those from the string pointed to by *lpStr2*. It returns a negative value if the first *nChar* characters of the string pointed to by *lpStr1* are less than those from the string pointed to by *lpStr2*.

Remarks

The **StrNCmp** macro differs from this function in name only.

This function compares the first *nChar* characters of the two strings. It returns zero if they are all equal or the difference in value of the first unequal characters it encounters. For example, if *lpStr1*="Abczb", *lpStr2* = "abCdefg", and *nChar* = 4, **StrCmpN** determines that "abCdefg" is greater than "Abczb" and returns A-a.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrCmpNI

Compares a specified number of characters from the beginning of two strings to determine if they are the same. The comparison is *not* case-sensitive.

```
int StrCmpNI(
  LPCTSTR lpStr1,
  LPCTSTR lpStr2,
  int nChar
);
```

Parameters

lpStr1
[in] Pointer to the first string to be compared.

lpStr2
[in] Pointer to the second string to be compared.

nChar
[in] Number of characters from the beginning of each string to be compared.

Return Values

Returns zero if the strings are identical. Returns a positive value if the first *nChar* characters of the string pointed to by *lpStr1*are greater than those from the string pointed to by *lpStr2*. It returns a negative value if the first *nChar* characters of the string pointed to by *lpStr1* are less than those from the string pointed to by *lpStr2*.

Remarks

The **StrNCmpI** macro differs from this function in name only.

This function compares the first *nChar* characters of the two strings. It returns zero if they are all equal or the difference in value of the first unequal characters it encounters. For example, if *lpStr1*="Abczb", *lpStr2* = "abCdefg", and *nChar* = 4, **StrCmpNI** determines that "Abczb" is greater than "abCdefg" and returns z - d.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrCpy

Copies one string to another.

```
LPTSTR StrCpy(
  LPTSTR psz1,
  LPCTSTR psz2,
);
```

Parameters

psz1
[out] Address of the destination string.

psz2
[in] Address of the source string.

Return Values

Returns a pointer to *psz1*.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrCpyN

Copies a specified number of characters from the beginning of one string to another. The **StrNCpy** macro differs from this function in name only.

```
LPTSTR StrCpyN(
  LPTSTR psz1,
  LPCTSTR psz2,
  int cchMax
);

#define StrNCpy StrCpyN
```

Parameters

psz1
[out] Pointer to a NULL-terminated string to hold the copied characters.

psz2
[in] Pointer to the source string.

cchMax

[in] Number of characters to be copied, including the terminating NULL character.

Return Values

Returns a pointer to *psz1*.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrCSpn

Searches a string for the first occurrence of any of a group of characters. The search method *is* case-sensitive, and the NULL terminator is included within the search pattern match.

```
int StrCSpn(
  LPCTSTR lpStr,
  LPCTSTR lpSet
);
```

Parameters

lpStr

[in] Pointer to the null-terminated string to be searched.

lpSet

[in] Pointer to a null-terminated string containing the characters to search for.

Return Values

Returns the index of the first occurrence in *lpStr* of any character from *lpSet*, or the length of *lpStr* if no match is found.

Remarks

The return value of this function is equal to the length of the initial substring in *lpStr* that does not include any characters from *lpSet*.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrCSpnI

Searches a string for the first occurrence of any of a group of characters. The search method is *not* case-sensitive, and the NULL terminator is included within the search pattern match.

```
int StrCSpnI(
  LPCTSTR lpStr,
  LPCTSTR lpSet
);
```

Parameters

lpStr
[in] Pointer to the null-terminated string to be searched.

lpSet
[in] Pointer to a null-terminated string containing the characters to search for.

Return Values

Returns the index of the first occurrence in *lpStr* of any character from *lpSet*, or the length of *lpStr* if no match is found.

Remarks

The return value of this function is equal to the length of the initial substring in *lpStr* that does not include any characters from *lpSet*.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).

Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrDup

Duplicates a string.

```
LPTSTR StrDup(
  LPCTSTR lpsz
);
```

Parameters

lpsz
Address of a constant null-terminated character string.

Return Values

Returns the address of the string that was copied, or NULL if the string cannot be copied.

Remarks

Note This function uses LocalAlloc to allocate storage space for the copy of the string. The calling application must free this memory by calling the LocalFree function on the pointer returned by the call to StrDup.

StrDup will allocate storage the size of the original string. If storage allocation is successful, the original string is copied to the duplicate string.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrFormatByteSize

Converts a numeric value into a string that represents the number expressed as a size value in bytes, kilobytes, megabytes, or gigabytes, depending on the size.

```
LPTSTR StrFormatByteSizeA(
  DWORD dw,
  LPSTR pszBuf,
  UINT cchBuf
);
LPTSTR StrFormatByteSizeW(
  LONGLONG qdw,
  LPWSTR pwszBuf,
  UINT cchBuf
);
```

Parameters

dw / qdw
  [in] Numeric value to be converted.

pszBuf / pwszBuf
  [out] Pointer to the converted string.

cchBuf
  [in] Size of *pszBuf*, in characters.

Return Values

Returns the address of the converted string, or NULL if the conversion fails.

Remarks

The first parameter of this function has a different type for the ANSI and Unicode versions. If your numeric value is a DWORD, you can use **StrFormatByteSize** with text macros for both cases. The compiler will cast the numerical value to a LONGLONG for the Unicode case. If your numerical value is a LONGLONG, you should use **StrFormatByteSizeW** explicitly.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrFormatByteSize64A

Converts a numeric value into a string that represents the number expressed as a size value in bytes, kilobytes, megabytes, or gigabytes, depending on the size.

```
LPSTR StrFormatByteSize64A(
  LONGLONG qdw,
  LPSTR pszBuf,
  UINT uiBufSize
);
```

Parameters

qdw

[in] Numeric value to be converted.

pszBuf

[out] Pointer a buffer to hold the converted number.

uiBufSize

[in] Size of *pszBuf*, in characters.

Return Values

Returns the address of the converted string, or NULL if the conversion fails.

Remarks

The following table illustrates how this function converts a numeric value into a text string.

Numeric value	Text string
532	532 bytes
1340	1.30KB
23506	22.9KB
2400016	2.29MB
2400000000	2.23GB

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

See Also

StrFormatByteSize

StrFormatKBSize

Converts a numeric value into a string that represents the number expressed as a size value in kilobytes.

```
LPTSTR StrFormatKBSize(
  LONGLONG qdw,
  LPTSTR pszBuf,
  UINT uiBufSize
);
```

Parameters

qdw

[in] Numeric value to be converted.

pszBuf

[out] Pointer to a buffer to hold the converted number.

uiBufSize

[in] Size of *pszBuf*, in characters.

Return Values

Returns a pointer to the converted string, or NULL if the conversion fails.

Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

See Also

StrFormatByteSize

StrFromTimeInterval

Converts a time interval, specified in milliseconds, to a string.

```
int StrFromTimeInterval(
  LPTSTR pszOut,
  UINT cchMax,
  DWORD dwTimeMS,
  int digits
);
```

Parameters

pszOut

[out] Pointer to a character buffer that receives the converted string.

cchMax

[in] Size of *pszOut*, in characters. If *cchMax* is set to zero, **StrFromTimeInterval** will return the minimum size of the character buffer needed to hold the converted string. In this case, *pszOut* will not contain the converted string.

dwTimeMS

[in] Time interval, in milliseconds.

digits

[in] Maximum number of digits to be represented in *pszOut*. Some examples are:

dwTimeMS	digits	pszOut
34000	3	34 sec
34000	2	34 sec
34000	1	30 sec
74000	3	1 min 14 sec
74000	2	1 min 10 sec
74000	1	1 min

Return Values

Returns the number of characters in *pszOut*, excluding the NULL terminator.

Remarks

The time value returned in *pzsOut* will always be in the form *hh* hours *mm* minutes *ss* seconds. Times that exceed twenty four hours are not converted to days or months. Fractions of seconds are ignored.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrIsIntlEqual

Compares a specified number of characters from the beginning of two strings to determine if they are equal.

```
BOOL StrIsIntlEqual(
  BOOL fCaseSens,
  LPCTSTR lpString1,
  LPCTSTR lpString2,
  int nChar
);
```

Parameters

fCaseSens
Determines the case sensitivity of the comparison. If this value is nonzero, the comparison is case-sensitive. If this value is zero, the comparison is not case-sensitive.

lpString1
Pointer to the first string to be compared.

lpString2
Pointer to the second string to be compared.

nChar
Number of characters from the beginning of each string to be compared.

Return Values

Returns TRUE if the first *nChar* characters from the two strings are equal, or FALSE otherwise.

Remarks

You can set case sensitivity with the **StrIntlEqN** and **StrIntlEqNI** macros. **StrIntlEqN** performs a case-sensitive comparison, and **StrIntlEqNI** performs a case-insensitive comparison.

The syntax of the two macros is:

```
#define StrIntlEqN(s1, s2, nChar) StrIsIntlEqual(TRUE, s1, s2, nChar)
#define StrIntlEqNI(s1, s2, nChar) StrIsIntlEqual(FALSE, s1, s2, nChar)
```

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrNCat

Appends a specified number of characters from the beginning of one string to the end of another. The **StrCatN** macro is identical to this function.

```
LPTSTR StrNCat(
  LPTSTR pszFront,
  LPCTSTR pszBack,
  int cchMax
);
```

Parameters

pszFront
  [in/out] Address of a NULL-terminated string to which the characters from *pszBack* will be appended. It must be large enough to hold the combined strings.

pszBack
  [in] Address of the string to be appended.

cchMax
  [in] Number of characters to be appended to *pszFront* from the beginning of *pszBack*.

Return Values

Returns a pointer to *pszFront*, which holds the combined string.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrPBrk

Searches a string for the first occurrence of a character contained in a specified buffer. This search does not include the null terminator.

```
LPTSTR StrPBrk(
  LPCTSTR psz,
  LPCTSTR pszSet
);
```

Parameters

psz
Address of the string to be searched.

pszSet
Address of a null-terminated character buffer that contains the characters for which to search.

Return Values

Returns the address in *psz* of the first occurrence of a character contained in the buffer at *pszSet*, or NULL if no match is found.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrRChr

Searches a string for the last occurrence of a specified character. The comparison *is* case-sensitive.

```
LPTSTR StrRChr(
  LPCTSTR pszStart,
  LPCTSTR pszEnd,
  TCHAR wMatch
```

Parameters

lpStart

[in] Pointer to the null-terminated string to be searched.

lpEnd

[in] Pointer into the source string that defines the range of the search. Set *lpEnd* to point to a character in the string and the search will stop with the preceding character. Set *lpEnd* to NULL to search the entire string.

wMatch

[in] Character to search for.

Return Values

Returns a pointer to the last occurrence of the character in the string, if successful, or NULL if not.

Remarks

The comparison assumes that *lpEnd* points to the end of the string.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrRChrI

Searches a string for the last occurrence of a specified character. The comparison is *not* case-sensitive.

```
LPTSTR StrRChrI(
  LPCTSTR pszStart,
  LPCTSTR pszEnd,
  TCHAR wMatch
);
```

Parameters

pszStart

[in] Pointer to the NULL-terminated string to be searched.

pszEnd

[in] Pointer into the source string that defines the range of the search. Set *lpEnd* to point to a character in the string and the search will stop with the preceding character. Set *lpEnd* to NULL to search the entire string.

wMatch

[in] Character to search for.

Return Values

Returns a pointer to the last occurrence of the character in the string if successful, or NULL if not.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrRetToBuf

Takes a **STRRET** structure returned by **IShellFolder::GetDisplayNameOf**, converts it to a string, and places the result in a buffer.

```
HRESULT StrRetToBuf(
  LPSTRRET pstr,
  LPCITEMIDLIST pidl,
  LPTSTR pszBuf,
  UINT cchBuf
);
```

Parameters

pstr

[in] Pointer to the **STRRET** structure. When the function returns, this pointer will no longer be valid.

pidl

[in] Pointer to the item's ITEMIDLIST structure.

pszBuf

[out] Buffer to hold the display name. It will be returned as a null-terminated string. If *cchBuf* is too small, the name will be truncated to fit.

cchBuf

[in] Size of *pszBuf*, in characters. If *cchBuf* is too small, the string will be truncated to fit.

Return Values

Returns S_OK if successful, or an OLE error code otherwise.

Remarks

If the **uType** member of the structure pointed to by *pstr* is set to STRRET_WSTR, the **pOleStr** member of that structure will be freed on return.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

+ See Also

StrRetToStr

StrRetToStr

Takes a **STRRET** structure returned by **IShellFolder::GetDisplayNameOf** and returns a pointer to an allocated string containing the display name.

```
HRESULT StrRetToStr(
  LPSTRRET pstr,
  LPCITEMIDLIST pidl,
  LPTSTR *ppszName
);
```

Parameters

pstr

[in] Pointer to the **STRRET** structure. When the function returns, this pointer will no longer be valid.

pidl
[in] Pointer to the item's **ITEMIDLIST** structure.

ppszName
[out] Pointer to an allocated string containing the result. **StrRetToStr** allocates memory for this string with **CoTaskMemAllocate**. You should free the string with **CoTaskMemFree** when it is no longer needed.

Return Values

Returns S_OK if successful, or an OLE error code otherwise.

Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

See Also

StrRetToBuf

StrRStrI

Searches for the last occurrence of a specified substring within a string. The comparison is *not* case sensitive.

```
LPTSTR StrRStrI(
  LPCTSTR pszSource,
  LPCTSTR pszLast,
  LPCTSTR pszSrch
);
```

Parameters

pszSource
[in] Pointer to a NULL-terminated source string.

pszLast
[in] Pointer into the source string that defines the range of the search. Set *pszLast* to point to a character in the source string and the search will stop with the preceding character. Set *pszLast* to NULL to search the entire source string.

pszSrch

[in] Pointer to the substring to search for.

Return Values

Returns the address of the last occurrence of the substring if successful, or NULL otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrSpn

Obtains the length of a substring within a string that consists entirely of characters contained in a specified buffer.

```
int StrSpn(
  LPCTSTR psz,
  LPCTSTR pszSet
);
```

Parameters

psz

Address of the string that is to be searched.

pszSet

Address of a null-terminated character buffer that contains the set of characters for which to search.

Return Values

Returns the length, in characters, of the matching string or zero if no match is found.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrStr

Finds the first occurrence of a substring within a string. The comparison is case sensitive.

```
LPTSTR StrStr(
  LPCTSTR lpFirst,
  LPCTSTR lpSrch
);
```

Parameters

lpFirst
Address of the string being searched.

lpSrch
Substring to search for.

Return Values

Returns the address of the first occurrence of the matching substring if successful, or NULL otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with
Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrStrI

Finds the first occurrence of a substring within a string. The comparison is not case sensitive.

```
LPTSTR StrStrI(
  LPCTSTR lpFirst,
  LPCTSTR lpSrch
);
```

Parameters

lpFirst

Address of the string being searched.

lpSrch

Substring to search for.

Return Values

Returns the address of the first occurrence of the matching substring if successful, or NULL otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrToInt

Converts a decimal string to an integer. The **StrToLong** macro differs from this function in name only.

```
int StrToInt(
  LPCTSTR lpSrc
);

#define StrToLong StrToInt
```

Parameters

lpSrc

Address of the null-terminated string to be converted.

Return Values

Returns the INT value of a string.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).

Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrToIntEx

Converts a decimal or hexadecimal string to an integer.

```
BOOL StrToIntEx(
  LPCTSTR pszString,
  DWORD dwFlags,
  int FAR *piRet
);
```

Parameters

pszString
Address of a null-terminated string to be converted.

dwFlags
Specifies if *pszString* contains a decimal or hexadecimal value. This can be one of the following values:

STIF_DEFAULT	*pszString* contains a decimal value.
STIF_SUPPORT_HEX	*pszString* contains a hexadecimal value.

piRet
Address of an integer variable that receives the converted string.

Return Values

Returns TRUE if the string is converted, or FALSE otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

StrTrim

Removes (trims) specified leading and trailing characters from a string.

```
BOOL StrTrim(
  LPTSTR pszSource,
  LPCTSTR pszTrimChars
);
```

Parameters

pszSource
[in/out] Pointer to the string to be trimmed. On return, *pszSource* will hold the trimmed string.

pszTrimChars
[in] Pointer to a null-terminated string containing the characters that will be trimmed from *psz*.

Return Values

Returns TRUE if any characters were removed, or FALSE otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

wnsprintf

Takes a variable-length argument list and returns the values of the arguments as a **printf**-style formatted string.

```
int wnsprintf(
  LPTSTR lpOut,
  int cchLimitIn
  LPCTSTR pszFmt,
...
);
```

Parameters

lpOut
 [out] Buffer to hold the output string.

cchLimitIn
 [in] Maximum number of characters allowed in *lpOut*.

pszFmt
 [in] **printf**-style format string.

Return Values

Returns the number of characters written to the buffer, excluding any terminating NULL characters. A negative value is returned if an error occurs.

Remarks

This is a Windows version of **sprintf**. It does not support floating point or pointer types. It supports only the left alignment flag.

The Unicode version of **wsprintf** () is not implemented for Windows 95. Use the Unicode version of this function (**wnsprintfW**) instead of **wsprintfW**.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

wvnsprintf

Takes list of arguments and returns the values of the arguments as a **printf**-style formatted string.

```
int wvnsprintf(
  LPTSTR lpOut,
  int cchLimitIn,
  LPCTSTR pszFmt,
  va_list arglist
);
```

Parameters

lpOut

[out] Buffer to hold the output string.

cchLimitIn

[in] Maximum number of characters allowed in *lpOut*.

pszFmt

[in] **printf**-style format string.

arglist

[in] Pointer to a list of parameters.

Return Values

Returns the number of characters written to the buffer, excluding any terminating NULL characters. A negative value is returned if an error occurs.

Remarks

This is a Windows version of **vsprintf()**. It does not support floating point or pointer types. It supports only the left alignment flag.

Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

Path Functions

PathAddBackslash

Adds a backslash to the end of a string to create the correct syntax for a path. If the source path already has a trailing backslash, no backslash will be added.

```
LPTSTR PathAddBackslash(
  LPTSTR lpszPath
);
```

Parameters

lpszPath

[in/out] Pointer to a buffer with a string that represents a path. The size of this buffer should be set to MAX_PATH to ensure that it is large enough to hold the returned string.

Return Values

Returns the address of the NULL that terminates the string.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathAddExtension

Adds a file extension to a path string.

```
BOOL PathAddExtension(
  LPTSTR  pszPath,
  LPCTSTR pszExtension
);
```

Parameters

pszPath

[in/out] Pointer to a buffer with the NULL-terminated string to which the file extension will be appended. The size of this buffer should be set to MAX_PATH to ensure that it is large enough to hold the returned string.

pszExtension

[in] Pointer to the string that contains the file extension.

Return Values

Returns TRUE if an extension was added, or FALSE otherwise.

Remarks

If there is already a file extension present, no extension will be added. If the *pszPath* points to a NULL string, the result will be the file extension only. If *pszExtension* points to a NULL string, an ".exe" extension will be added.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with
Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathAppend

Appends one path to the end of another.

```
BOOL PathAppend(
  LPTSTR  pszPath,
  LPCTSTR pszMore
);
```

Parameters

pszPath

[in/out] Pointer to a buffer with the NULL-terminated string to which the file name will be appended. The size of this buffer should be set to MAX_PATH to ensure that it is large enough to hold the returned string.

pszMore

[in] Address of the string that represents the file name.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

This function automatically inserts a backslash between the two strings, if one is not already present. If *pszPath* is set to "c:", no backslash will be inserted to allow drive-relative paths to be used.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathBuildRoot

Creates a root path from a given drive number.

```
LPTSTR PathBuildRoot(
  LPTSTR szRoot,
  int iDrive
);
```

Parameters

szRoot
Address of the string that receives the constructed root path. This buffer must be at least four characters in size.

iDrive
Integer value that indicates the desired drive number. It should be between 0 and 25.

Return Values

Returns the address of the constructed root path, or NULL otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathCanonicalize

Canonicalizes a path.

```
BOOL PathCanonicalize(
  LPTSTR lpszDst,
  LPCTSTR lpszSrc
);
```

Parameters

lpszDst
Address of a string that receives the canonicalized path. The size of this buffer should be set to MAX_PATH to ensure that it is large enough to hold the returned string.

lpszSrc
Address of the path to be canonicalized.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

This function allows the user to specify what to remove from a path by inserting special character sequences into the path. The "." sequence indicates to remove the path part from the current position to the previous path part. The "." sequence indicates to skip over the next path part to the following path part. The root part of the path cannot be removed.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathCombine

Concatenates two strings that represent properly formed paths into one path, as well as any relative path pieces.

```
LPTSTR PathCombine(
  LPTSTR lpszDest,
  LPCTSTR lpszDir,
  LPCTSTR lpszFile
);
```

Parameters

lpszDest

[out] Pointer to a buffer with the NULL-terminated string to hold the combined path string. The size of this buffer should be set to MAX_PATH to ensure that it is large enough to hold the returned string.

lpszDir

[in]Address of the string that represents the directory path.

lpszFile

[in]Address of the string that represents the file path.

Return Values

Returns a pointer to a string with the concatenated path if successful, or NULL otherwise.

Remarks

The directory path should be in the form of A:,B:, .., Z:. The file path should be in a correct form that represents the file part of the path. The file path must not be null, and if it ends with a backslash, the backslash will be maintained.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathCommonPrefix

Compares two paths to determine if they share a common prefix. A prefix is one of these types: “C:\\”, “.”, “.”, “.\\”.

```
int PathCommonPrefix(
  LPCTSTR pszFile1,
  LPCTSTR pszFile2,
  LPTSTR pszPath
);
```

Parameters

pszFile1
[in] Address of the first path name.

pszFile2
[in] Address of the second path name.

pszPath
[out] Address of a buffer that receives the common prefix. If there is no common prefix, it will be set to NULL.

Return Values

Returns the number of characters copied to the output buffer (not including the NULL terminator) if successful, or zero otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathCompactPath

Truncates a file path to fit within a given pixel width by replacing path components with ellipses.

```
BOOL PathCompactPath(
  HDC hDC,
  LPTSTR lpszPath,
  UINT dx
);
```

Parameters

hDC

Handle to the device context used for font metrics.

lpszPath

Address of the string to be modified.

dx

Width, in pixels, that the string will be forced to fit within.

Return Values

Returns TRUE if the path was successfully compacted to the specified width. Returns FALSE on failure, or if the base portion of the path would not fit the specified width.

Remarks

This function uses the font currently selected in *hDC* to calculate the width of the text. This function will not compact the path beyond the base file name preceded by ellipses.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathCompactPathEx

Truncates a path to fit within a certain number of characters by replacing path components with ellipses.

```
BOOL PathCompactPathEx(
  LPTSTR  pszOut,
  LPCTSTR pszSrc,
  UINT    cchMax,
  DWORD   dwFlags
);
```

Parameters

pszOut

[out] Address of the string that has been altered.

pszSrc
[in] Address of the string to be altered.

cchMax
[in] Maximum number of characters to be contained in the new string, including the terminating NULL character. For example, if *cchMax* = 8, the resulting string can contain a maximum of 7 characters plus the NULL terminator.

dwFlags
Reserved.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

The ‘/’ separator will be used instead of ‘\’ if the original string used it. If *pszSource* points to a file name that is too long, rather than a path, the file name will be truncated to *cchMax* characters, including the ellipsis and the terminating NULL character. For example, if the input file name is “My Filename” and *cchMax* is 10, **PathCompactPathEx** will return “My Fil..”.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathCreateFromUrl

Takes a file URL and converts it to a DOS path.

```
HRESULT PathCreateFromUrl(
  LPCTSTR pszUrl,
  LPTSTR pszPath,
  LPDWORD pcchPath,
  DWORD dwReserved
);
```

Parameters

pszUrl

Pointer to the string with the URL.

pszPath

Value used to return the DOS path. The size of this buffer should be set to MAX_PATH to ensure that it is large enough to hold the returned string.

pcchPath

Length of *pszPath*.

dwReserved

Reserved. Set this parameter to NULL.

Return Values

Returns S_OK if successful, or a standard OLE error value otherwise.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathFileExists

Determines if a file exists.

```
BOOL PathFileExists(
  LPCTSTR pszPath
);
```

Parameters

pszPath

Address of the file to verify.

Return Values

Returns TRUE if the file exists, or FALSE otherwise.

Remarks

This function tests the validity of the file and path. It works only on the local file system or on a remote drive that has been mounted to a drive letter. It will return FALSE for remote file paths that begin with the UNC names \\server or \\server\share. It will also return FALSE if a mounted remote drive is out of service.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathFindExtension

Searches a path for an extension.

```
LPTSTR PathFindExtension(
  LPCTSTR pPath
);
```

Parameters

pPath
Address of the path that contains the extension for which to search.

Return Values

Returns the address of the "." preceding the extension within *pPath* if an extension is found, or the address of the trailing NULL character otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathFindFileName

Searches a path for a file name.

```
LPTSTR PathFindFileName(
  LPCTSTR pPath
);
```

Parameters

pPath

Address of the file name for which to search.

Return Values

Returns a pointer to the address of the string if successful, or a pointer to the beginning of the path otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathFindNextComponent

Parses a path for the next path component.

```
LPTSTR PathFindNextComponent(
  LPCTSTR pszPath
);
```

Parameters

pszPath

[in] Pointer to a NULL-terminated string with the path. Paths are delimited by backslashes or by the NULL at the end of the path.

Return Values

Returns a pointer to a NULL-terminated string with the next path component if successful, or NULL otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathFindOnPath

Searches for a file.

```
BOOL PathFindOnPath(
  LPTSTR pszFile,
  LPCTSTR *ppszOtherDirs
);
```

Parameters

pszFile

File name for which to search. If the search is successful, this parameter is used to return the fully qualified path name.

ppszOtherDirs

Optional null-terminated array of directories to be searched first.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

PathFindOnPath searches for the file specified by *pszFile*. If no directories are specified in *ppszOtherDirs*, it attempts to find the file by searching standard directories such as System32 and the directories specified in the PATH environment variable. To expedite the process or enable **PathFindOnPath** to search a wider range of directories, use the *ppszOtherDirs* parameter to specify one or more directories to be searched first. If more than one file has the name specified by *pszFile*, **PathFindOnPath** returns the first instance it finds.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathFindSuffixArray

Determines if a given file name has one of a list of suffixes.

```
LPCTSTR PathFindSuffixArray(
  LPCTSTR pszPath,
  LPCTSTR *apszSuffix,
  int iArraySize
);
```

Parameters

pszPath
[in] File name to be tested. A full path can also be used.

apszSuffix
[in] Array of suffixes to be tested for.

iArraySize
[in] Number of elements in *apszSuffix*.

Return Values

Returns a pointer to a string with the matching suffix if successful, or NULL if *pszPath* does not end with one of the specified suffixes.

Remarks

This function does a case-sensitive comparison. The suffix must match exactly.

Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathGetArgs

Finds the command line arguments within a given path.

```
LPTSTR PathGetArgs(
  LPCTSTR pszPath
);
```

Parameters

pszPath
  Address of the path to be searched.

Return Values

Returns the address of the beginning of the command line arguments if successful, or NULL otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathGetCharType

Determines the type of character with respect to a path.

```
UINT PathGetCharType(
  TUCHAR ch
);
```

Parameters

ch
  Character for which to determine the type.

Return Values

Returns one or more of the following values that define the type of character:

GCT_INVALID	The character is not valid in a path.
GCT_LFNCHAR	The character is valid in a long file name.
GCT_SEPARATOR	The character is a path separator.
GCT_SHORTCHAR	The character is valid in a short (8.3) file name.
GCT_WILD	The character is a wildcard character.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathGetDriveNumber

Searches a path for a drive letter within the range of 'A' to 'Z' and returns the corresponding drive number.

```
int PathGetDriveNumber(
  LPCTSTR lpsz
);
```

Parameters

lpsz
Address of a string that contains the path to be searched.

Return Values

Returns 0 through 25 (corresponding to 'A' through 'Z') if the path has a drive letter, or –1 otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.

Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathIsContentType

Determines if a file's registered content type matches the specified content type. This function obtains the content type for the specified file type and compares that string with the *pszContentType*. The comparison is not case sensitive.

```
BOOL PathIsContentType(
  LPCTSTR pszPath,
  LPCTSTR pszContentType
);
```

Parameters

pszPath

Address of a character buffer that contains the file whose content type will be compared.

pszContentType

Address of a character buffer that contains the content type string to which the file's registered content type will be compared.

Return Values

Returns nonzero if the file's registered content type matches *pszContentType*, or zero otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathIsDirectory

Verifies that a path is a valid directory.

```
BOOL PathIsDirectory(
  LPCTSTR pszPath
);
```

Parameters

pszPath
Address of the path to verify.

Return Values

Returns TRUE if the path is a valid directory, or FALSE otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathIsDirectoryEmpty

Determines whether or not a specified path is an empty directory.

```
BOOL PathIsDirectoryEmpty(
  LPCTSTR pszPath
);
```

Parameters

pszPath
[in] NULL-terminated string with the path to be tested.

Return Values

Returns TRUE if *pszPath* is an empty directory. Returns FALSE if *pszPath* is not a directory, or if it contains at least one file other than "." or ".".

Remarks

"C:\" is considered a directory.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

+ See Also

PathIsDirectory

PathIsFileSpec

Searches a path for any path delimiting characters (for example, ‘:’ or ‘\’). If there are no path delimiting characters present, the path is considered to be a File Spec path.

```
BOOL PathIsFileSpec(
  LPCTSTR lpszPath
);
```

Parameters

lpszPath
Address of the path to be searched.

Return Values

Returns TRUE if there are no path delimiting characters within the path, or FALSE if there are path delimiting characters.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathIsHTMLFile

Determines if a file is an HTML file. The determination is made based on the content type that is registered for the file’s extension.

```
BOOL PathIsHTMLFile(
  LPCTSTR pszFile
);
```

Parameters

pszFile

Address of a character buffer that contains the path and name of the file.

Return Values

Returns nonzero if the file is an HTML file, or zero otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathIsLFNFileSpec

Determines whether or not a file name is in long format.

```
BOOL PathIsLFNFileSpec(
  LPCTSTR pszName
);
```

Parameters

pszName

[in] NULL-terminated string with the file name to be tested.

Return Values

Returns TRUE if *pszName* exceeds the number of characters allowed by the 8.3 format, or FALSE otherwise.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

+ See Also

PathIsFileSpec

PathIsNetworkPath

Determines whether a path string represents a network resource.

```
BOOL PathIsNetworkPath(
  LPCTSTR pszPath
);
```

Parameters

pszPath
[in] NULL-terminated string that contains the path.

Return Values

Returns TRUE if the string represents a network resource, or FALSE otherwise.

Remarks

PathIsNetwork interprets two types of paths as network paths:

- Paths that begin with two backslash (\) characters are interpreted as UNC paths.
- Paths that begin with a letter followed by a colon (:) are interpreted as a mounted network drive.

The path is not checked to see if it refers to an actual network server.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathIsPrefix

Searches a path to determine if it contains a valid prefix of the type passed by *pszPrefix*. A prefix is one of these types: “C:\\”, “.”, “.”, “.\\”.

```
BOOL PathIsPrefix(
  IN LPCTSTR pszPrefix,
  IN LPCTSTR pszPath
);
```

Parameters

pszPrefix

Address of the prefix for which to search.

pszPath

Address of the path to be searched.

Return Values

Returns TRUE if the compared path is the full prefix for the path, or FALSE otherwise.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathIsRelative

Searches a path and determines if it is relative.

```
BOOL PathIsRelative(
  LPCTSTR lpszPath
);
```

Parameters

lpszPath

Address of the path to search.

Return Values

Returns TRUE if the path is relative, or FALSE if it is absolute.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathIsRoot

Parses a path to determine if it is a directory root.

```
BOOL PathIsRoot(
  LPCTSTR pPath
);
```

Parameters

pPath
Address of the path to be validated.

Return Values

Returns TRUE if the specified path is a root, or FALSE otherwise.

Remarks

Returns TRUE for paths such as “\”, “*X*:\”, “\\*server**share*”, or “\*server*\”, or for paths that begin with those strings. Paths such as “.\path2” will return FALSE.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathIsSameRoot

Compares two paths to determine if they have a common root component.

```
BOOL PathIsSameRoot(
  LPCTSTR pszPath1,
  LPCTSTR pszPath2
);
```

Parameters

pszPath1
Address of the first path to be compared.

pszPath2
Address of the second path to be compared.

Return Values

Returns TRUE if both strings have the same root component, or FALSE otherwise.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathIsSystemFolder

Determines if an existing folder contains the attributes that make it a system folder. Alternately indicates if certain attributes qualify a folder to be a system folder.

```
BOOL PathIsSystemFolder(
  LPCTSTR pszPath,
  DWORD dwAttrb
);
```

Parameters

pszPath
[in] Address of a character buffer that contains the name of an existing folder. The attributes for this folder will be retrieved and compared with those that define a system folder. If this folder contains the attributes to make it a system folder, the function

returns nonzero. If this value is NULL, this function determines if the attributes passed in *dwAttrb* qualify it to be a system folder.

dwAttrb

[in] Contains the file attributes to be compared. If *pszPath* is not NULL, this value is ignored. If *pszPath* is NULL, the attributes passed in this value are compared with those that qualify a folder as a system folder. If the attributes are sufficient to make this a system folder, this function returns nonzero. These attributes are the attributes that are returned from **GetFileAttributes**.

Return Values

Returns nonzero if the *pszPath* or *dwAttrb* represent a system folder, or zero otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with
Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathIsUNC

Determines if the string is a valid UNC (universal naming convention) for a server and share path.

```
BOOL PathIsUNC(
  LPCTSTR pszPath
);
```

Parameters

pszPath

Address of the path to validate.

Return Values

Returns TRUE if the string is a valid UNC path, or FALSE otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathIsUNCServer

Determines if a string is a valid UNC (universal naming convention) for a server path only.

```
BOOL PathIsUNCServer(
  LPCTSTR pszPath
);
```

Parameters

pszPath
Address of the path to validate.

Return Values

Returns TRUE if the string is a valid UNC path for a server only (no share name), or FALSE otherwise.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathIsUNCServerShare

Determines if a string is a valid universal naming convention (UNC) share path, \\server\share.

```
BOOL PathIsUNCServerShare(
  LPCTSTR pszPath
);
```

Parameters

pszPath

Pointer to a string with the path to be validated.

Return Values

Returns TRUE if the string is in the form \\*server**share*, or FALSE otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathIsURL

Tests a given string to determine if it conforms to a valid URL format.

```
BOOL PathIsURL(
  LPCTSTR pszPath
);
```

Parameters

pszPath

[in] Address of the URL path to validate.

Return Values

Returns TRUE if *pszPath* has a valid URL format, or FALSE otherwise.

Remarks

This function does not verify that the path points to an existing site—only that it has a valid URL format.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathMakePretty

Converts a path to all lowercase characters to give the path a consistent appearance.

```
BOOL PathMakePretty(
  LPTSTR lpPath
);
```

Parameters

lpPath
Address of the path to be converted.

Return Values

Returns TRUE if the path has been converted, or FALSE otherwise.

Remarks

This function only operates on paths that are entirely uppercase. For example: C:\WINDOWS will be converted to c:\windows, but c:\Windows will not be changed.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathMakeSystemFolder

Gives an existing folder the proper attributes to become a system folder.

```
BOOL PathMakeSystemFolder(
  LPTSTR pszPath
);
```

Parameters

pszPath

Address of a character buffer that contains the name of an existing folder that will be made into a system folder.

Return Values

Returns nonzero if successful, or zero otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathMatchSpec

Searches a string using a DOS wild card match type.

```
BOOL PathMatchSpec(
  LPCTSTR pszFileParam,
  LPCTSTR pszSpec
);
```

Parameters

pszFileParam

Pointer to the string to be searched.

pszSpec

Pointer to a NULL-terminated string with the file type for which to search. For example, to test whether or not *pszFileParam* is a DOC file, *pszSpec* should be set to "*.doc".

Return Values

Returns TRUE if the string matches, or FALSE otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathParseIconLocation

Parses a file location string containing a file location and icon index, and returns separate values.

```
int PathParseIconLocation(
  LPTSTR pszIconFile
);
```

Parameters

pszIconFile

[in/out] Pointer to a file location string. It should be in the form "*pathname,iconindex*". When the function returns, *pszIconFile* will point to the file's pathname.

Return Values

Returns the valid icon index value.

Remarks

This function is useful for taking a DefaultIcon value retrieved from the registry by SHGetValue, and separating the icon index from the pathname.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathQuoteSpaces

Searches a path for spaces. If spaces are found, the entire path is enclosed in quotation marks.

```
void PathQuoteSpaces(
  LPTSTR lpsz
);
```

Parameters

lpsz

[in/out] Pointer to a buffer with a string containing the path to search. The size of this buffer should be set to MAX_PATH to ensure that it is large enough to hold the returned string.

Return Values

No return value.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathRelativePathTo

Creates a relative path from one file or folder to another.

```
BOOL PathRelativePathTo(
  LPTSTR pszPath,
  LPCTSTR pszFrom,
  DWORD dwAttrFrom,
  LPCTSTR pszTo,
  DWORD dwAttrTo
);
```

Parameters

pszPath

Pointer to a string that receives the relative path. This buffer is assumed to be at least MAX_PATH characters in size.

pszFrom

Pointer to a string that contains the path that defines the start of the relative path.

dwAttrFrom

File attributes of *pszFrom*. If this value contains FILE_ATTRIBUTE_DIRECTORY, *pszFrom* is assumed to be directory; otherwise, *pszFrom* is assumed to be a file.

pszTo

Pointer to a string that contains the path that defines the endpoint of the relative path.

dwAttrTo

File attributes of *pszTo*. If this value contains FILE_ATTRIBUTE_DIRECTORY, *pszTo* is assumed to be directory; otherwise, *pszTo* is assumed to be a file.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

This function takes a pair of paths and generates a relative path from one to the other. The paths do not have to be fully-qualified, but they must have a common prefix or the function will fail and return FALSE.

For example, let the starting point, *pszFrom*, be “c:\FolderA\FolderB\FolderC” and the ending point, *pszTo*, be “c:\FolderA\FolderD\FolderE”. **PathRelativePathTo** will return the relative path from *pszFrom* to *pszTo* as: “.\.\FolderD\FolderE”. You will get the same result if you set *pszFrom* to “\FolderA\FolderB\FolderC” and *pszTo* to “\FolderA\FolderD\FolderE”. On the other hand, “c:\FolderA\FolderB” and “a:\FolderA\FolderD do not share a common prefix, and the function will fail. Note that “\\” is not considered a prefix and is ignored. If you set *pszFrom* to “\\FolderA\FolderB”, and *pszTo* to “\\FolderC\FolderD”, the function will fail.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathRemoveArgs

Removes any arguments from a given path.

```
void PathRemoveArgs(
  LPTSTR pszPath
);
```

Parameters

pszPath
Address of the path from which to remove arguments.

Return Values

No return value.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathRemoveBackslash

Removes the trailing backslash from a given path.

```
LPTSTR PathRemoveBackslash(
  LPTSTR lpszPath
);
```

Parameters

lpszPath
Address of the string from which to remove the backslash.

Return Values

Returns the address of the NULL that replaced the backslash, or the address of the last character if it's not a backslash.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathRemoveBlanks

Removes all leading and trailing spaces from a string.

```
void PathRemoveBlanks(
  LPTSTR lpszString
);
```

Parameters

lpszString
Address of the string from which to strip all leading and trailing spaces.

Return Values

No return value.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathRemoveExtension

Removes the file extension from a path, if there is one.

```
void PathRemoveExtension(
  LPTSTR pszPath
);
```

Parameters

pszPath

Address of the path from which to remove the extension.

Return Values

No return value.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathRemoveFileSpec

Removes the trailing file name and backslash from a path, if it has them.

```
BOOL PathRemoveFileSpec(
  LPTSTR pszPath
);
```

Parameters

pszPath

Address of the path from which to remove the file name.

Return Values

Returns nonzero if something was removed, or zero otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathRenameExtension

Replaces the extension of a file name with a new extension. If the file name does not contain an extension, the extension will be attached to the end of the string.

```
BOOL PathRenameExtension(
  LPTSTR pszPath,
  LPCTSTR pszExt
);
```

Parameters

pszPath

Address of the path for which to replace the extension. This buffer must be at least MAX_PATH characters in size.

pszExt

Address of a character buffer that contains a '.' followed by the new extension.

Return Values

Returns nonzero if successful, or zero if the new path and extension would exceed MAX_PATH characters.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathSearchAndQualify

Determines if a given path is correctly formatted and fully qualified.

```
BOOL PathSearchAndQualify(
  LPCTSTR pcszPath,
  LPTSTR pszFullyQualifiedPath,
  UINT cchFullyQualifiedPath
);
```

Parameters

pcszPath
Address of the path to search.

pszFullyQualifiedPath
Address of the path to be referenced. The size of this buffer should be set to MAX_PATH to ensure that it is large enough to hold the returned string.

cchFullyQualifiedPath
Width of the path, in characters, to be qualified.

Return Values

Returns TRUE if the path is qualified, or FALSE otherwise.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathSetDlgItemPath

Sets the text of a child control in a window or dialog box, using **PathCompactPath** to make sure the path fits in the control.

```
bool PathSetDlgItemPath(
  HWND hDlg,
  int id,
  LPCSTR pszPath,
);
```

Parameters

hDlg
Handle to the dialog box or window.

id
Identifier of the control.

pszPath
Address of a string that contains the path to set in the control.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathSkipRoot

Parses a path, ignoring the drive letter or UNC server/share path parts.

```
LPTSTR PathSkipRoot(
  LPCTSTR pszPath
);
```

Parameters

pszPath
Address of the path to parse.

Return Values

Returns the address of the beginning of the subpath that follows the root (drive letter or UNC server/share).

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathStripPath

Removes the path portion of a fully qualified path and file.

```
void PathStripPath(
  LPTSTR pszPath
);
```

Parameters

pszPath

Address of a string that contains the path and file name that will have the path portion removed.

Return Values

No return value.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathStripToRoot

Removes all parts of the path except for the root information.

```
BOOL PathStripToRoot(
  LPTSTR szRoot
);
```

Parameters

szRoot

Address of the path to be converted.

Return Values

Returns TRUE if a valid drive letter was found in the path, or FALSE otherwise.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathUndecorate

Removes the decoration from a path string.

```
VOID PathUndecorate(
  LPTSTR pszPath
);
```

Parameters

pszPath

[in] NULL-terminated string that contains the path. When the function returns, *pszPath* points to the undecorated string.

Example

The following table illustrates how strings are modified by **PathUndecorate**:

Initial string	Undecorated string
C:\Path\File[5].txt	C:\Path\File.txt
C:\Path\File[12]	C:\Path\File
C:\Path\File.txt	C:\Path\File.txt
C:\Path\[3].txt	C:\Path\[3].txt

Return Values

None.

Remarks

A decoration consists of a pair of square brackets with one or more digits in between, inserted immediately after the base name and before the file name extension.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathUnExpandEnvStrings

Takes a fully qualified path, and replaces several folder names with their associated environment string.

```
BOOL PathUnExpandEnvStrings(
  LPCTSTR pszPath,
  LPTSTR pszBuf,
  UINT cchBuf
);
```

Parameters

pszPath

[in] Path to be unexpanded.

pszBuf

[out] Buffer to receive the unexpanded string. The size of this buffer should be set to MAX_PATH to ensure that it is large enough to hold the returned string.

cchBuf

[in] Number of characters in the buffer.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

The following folders' paths will be replaced by their equivalent environment string:

Folder	Environment string
The current user's profile folder	%USERPROFILE%
The All Users profile folder	%ALLUSERSPROFILE%
The Program Files folder	%ProgramFiles%
The system root folder	%SystemRoot%
The system drive letter	%SystemDrive%

Note %USERPROFILE% is relative to the user making the call. This function does not work if the user is being impersonated from a service. For further discussion of access control issues, see *Access Control.*

The environment variables listed in the above table might not all be set on any particular system. If an environment variable is not set, it will not be unexpanded. In particular, none of these variables are set for the default environment of Windows 95 or Windows 98. The %ProgramFiles% variable is new for Windows 2000, and will typically not be set on Windows NT 4.0 systems.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4 with Internet Explorer 5 or later installed).
Windows 95/98: Requires Windows 98 or Windows 95 with Internet Explorer 5 or later.
Windows CE: Unsupported.

Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

+ See Also

DoEnvironmentSubst

PathUnmakeSystemFolder

Removes the attributes from a folder that make it a system folder. This folder must actually exist in the file system.

```
BOOL PathUnmakeSystemFolder(
  LPTSTR pszPath
);
```

Parameters

pszPath
Address of a character buffer that contains the name of an existing folder that will have the system folder attributes removed.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

PathUnquoteSpaces

Removes quotes from the beginning and end of a path.

```
void PathUnquoteSpaces(
  LPTSTR lpsz
);
```

Parameters

lpsz
Address of the path from which to remove the quotes.

Return Values

No return value.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

UrlApplyScheme

Takes a URL string, determines a scheme for it, and returns a string with an appropriate prefix.

```
HRESULT UrlApplyScheme(
  LPCTSTR pszIn,
  LPTSTR pszOut,
  LPDWORD pcchOut,
  DWORD dwFlags
);
```

Parameters

pszIn
URL string.

pszOut
URL specified by *pszIn*, converted to the standard *prefix://URL_string* format.

pcchOut
Length of *pszOut*.

dwFlags
Flags that specify how to determine the scheme. The following flags can be combined.

URL_APPLY_DEFAULT
Apply the default scheme if **UrlApplyScheme** can't determine one. The default prefix is stored in the registry but is typically "http".

URL_APPLY_GUESSSCHEME
Attempt to determine the scheme by examining *pszIn*.

URL_APPLY_GUESSFILE
Attempt to determine a file URL from *pszIn*.

URL_APPLY_FORCEAPPLY
Force **UrlApplyScheme** to determine a scheme for pszIn.

Return Values

S_OK
A scheme was determined; *pszOut* contains the URL string with the scheme's prefix.

S_FALSE
There were no errors, but no prefix was prepended.

Errors
A standard OLE error value is returned.

Remarks

Almost any combination of two or more characters followed by a colon will be parsed as a scheme. Valid characters include some common punctuation marks, such as ".". If your input string fits this description, **UrlApplyScheme** may treat it as valid and not apply a prefix. To ensure that an appropriate prefix is prepended, set the URL_APPLY_FORCEAPPLY flag. You should also set the URL_APPLY_DEFAULT flag, since that is what will normally be applied.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

UrlCanonicalize

Takes a URL string and converts it into canonical form.

```
HRESULT UrlCanonicalize(
  LPCTSTR pszUrl,
  LPTSTR pszCanonicalized,
  LPDWORD pcchCanonicalized,
  DWORD dwFlags
);
```

Parameters

pszUrl
[in] Pointer to a URL string. If it does not refer to a file, it must include a valid scheme such as “http://”.

pszCanonicalized
[out] A pointer to a NULL-terminated string used to return the converted URL.

pcchCanonicalized
[out] The number of characters in *pszCanonicalized*.

dwFlags
[in] Flags that specify how the URL will be converted to canonical form. The following flags can be combined:

Flag	Description
URL_DONT_SIMPLIFY	Do not convert “.” and “..”.
URL_DONT_ESCAPE_EXTRA_INFO	Don’t convert the ‘#’ or ‘?’ character, or any characters following them in the string.
URL_ESCAPE_SPACES_ONLY	Replace only spaces with escape sequences. This flag cannot be combined with URL_ESCAPE_UNSAFE.
URL_ESCAPE_UNSAFE	Replace unsafe values with their escape sequences.

Return Values

Returns S_OK if successful, or a standard OLE error value otherwise.

Remarks

This function will do such tasks as replacing unsafe characters with their escape sequences and collapsing sequences like ".\..".

Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

UrlCombine

Takes a relative URL and its base and returns a URL in canonical form.

```
HRESULT UrlCombine(
  LPCTSTR pszBase,
  LPCTSTR pszRelative,
  LPTSTR pszCombined,
  LPDWORD pcchCombined,
  DWORD dwFlags
);
```

Parameters

pszBase
[in] Pointer to a string with the base URL.

pszRelative
[in] Pointer to a string with the relative URL.

pszCombined
[out] Value used to return the combined URL.

pcchCombined
[out] Length of *pszCombined*.

dwFlags
[in] Flags that specify how the URL will be converted to canonical form. The following flags can be combined:

URL_ESCAPE_UNSAFE	Replace unsafe values with their escape sequences.
URL_DONT_SIMPLIFY	Do not convert "." and "..".
URL_ESCAPE_SPACES_ONLY	Replace only spaces with escape sequences.

Return Values

Returns S_OK if successful, or a standard OLE error value otherwise.

Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

See Also

UrlCanonicalize

UrlCompare

Compares two URL strings.

```
int UrlCompare(
  LPCTSTR pszURL1,
  LPCTSTR pszURL2,
  BOOL fIgnoreSlash
);
```

Parameters

pszURL1
[in] NULL-terminated string with the first URL.

pszURL2
[in] NULL-terminated string with the second URL.

fIgnoreSlash
[in] Value that is set to TRUE to have **UrlCompare** ignore a trailing '/' character on both URLs.

Return Values

Returns zero if the two strings are equal, apart from a trailing '\' character if *fIgnoreSlash* is set to TRUE. Returns a negative integer if the string pointed to by *pszURL1* is less than the string pointed to by *pszURL2*. Otherwise, it returns a positive integer.

Remarks

The comparison is case sensitive.

Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

See Also

StrCmp

UrlCreateFromPath

Takes a DOS path and converts it to a canonicalized URL.

```
HRESULT UrlCreateFromPath(
  LPCTSTR pszPath,
  LPTSTR pszUrl,
  LPDWORD pcchUrl,
  DWORD dwReserved
);
```

Parameters

pszPath
  Pointer to the string with the DOS path.

pszUrl
  Value used to return the URL.

pcchPath
  Length of *pszUrl*.

dwReserved
  Reserved. Set this parameter to NULL.

Return Values

Returns S_FALSE if *pszPath* is already in URL format. In this case, *pszPath* will simply be copied to *pszUrl*. Otherwise, it returns S_OK if successful or a standard OLE error value if not.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

UrlEscape

Converts unsafe characters, such as spaces, into their corresponding escape sequences.

```
HRESULT UrlEscape(
  LPCTSTR pszURL,
  LPTSTR pszEscaped,
  LPDWORD pcchEscaped,
  DWORD dwFlags
);
```

Parameters

pszURL
[in] Pointer to a NULL-terminated string with the URL.

pszEscaped
[out] Pointer to a NULL-terminated string containing the string pointed to by *pszURL*, with unsafe characters converted to their escape sequences.

pcchEscaped
[in/out] Number of characters in the buffer pointed to by *pszEscaped*. On entry, the value *pcchEscaped* points to is set to the size of the buffer. When the function returns, the value *pcchEscaped* points to is set to the number of characters written to that buffer, not counting the terminating NULL character. If an E_POINTER error code is returned, the buffer was too small, and the value *pcchEscaped* points to is set to the required number of characters in the buffer. If any other errors are returned, the value that *pcchEscaped* points to is undefined.

dwFlags

[in] Flags that control which characters are escaped. It can be a combination of the following flags.

Flag	Description
URL_DONT_ESCAPE_EXTRA_INFO	Don't convert the # or ? character, or any characters following them in the string.
URL_ESCAPE_SPACES_ONLY	Only escape space characters. This flag cannot be combined with URL_ESCAPE_PERCENT or URL_ESCAPE_SEGMENT_ONLY.
URL_ESCAPE_PERCENT	Escape the % character. By default, this character is not escaped.
URL_ESCAPE_SEGMENT_ONLY	Escape the sections following the server component, but not the extra information sections following a # or ? character.

Return Values

Returns an OLE success code if successful. The value pointed to by *pcchEscaped* will be set to the number of characters written to the output buffer, excluding the terminating NULL. If the buffer was too small, E_POINTER is returned, and the value pointed to by *pcchEscaped* will be set to the required buffer size. Otherwise, an OLE error value is returned.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

UrlEscapeSpaces

A macro that converts space characters into their corresponding escape sequence.

```
HRESULT UrlEscapeSpaces(
  LPCTSTR pszURL,
  LPTSTR pszEscaped,
  LPDWORD pcchEscaped
);
```

Parameters

pszURL

[in] Pointer to a URL string. If it does not refer to a file, it must include a valid scheme such as “http://”.

pszEscaped

[out] Pointer to a NULL-terminated string containing the string pointed to by *pszURL*, with space characters converted to their escape sequence.

pcchEscaped

[out] Number of characters in *pszEscaped*.

Return Values

Returns S_OK if successful, or a standard OLE error value otherwise.

Remarks

UrlEscapeSpaces is equivalent to:

```
UrlCanonicalize(pszUrl, pszEscaped, pcchEscaped, URL_ESCAPE_SPACES_ONLY
    |URL_DONT_ESCAPE_EXTRA_INFO )
```

Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

See Also

UrlCanonicalize

UrlGetLocation

Retrieves the location from a URL.

```
LPCTSTR UrlGetLocation(
  LPCTSTR pszURL,
);
```

Parameters

pszURL

[in] NULL-terminated string that contains the location.

Return Values

Returns a pointer to a NULL-terminated string with the location, or NULL otherwise.

Remarks

The location is the segment of the URL starting with a ? or # character.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

UrlGetPart

Takes a URL string and returns a specified part.

```
HRESULT UrlGetPart(
  LPCTSTR pszIn,
  LPTSTR pszOut,
  LPDWORD pcchOut,
  DWORD dwPart,
  DWORD dwFlags
);
```

Parameters

pszIn

[in] NULL-terminated string that contains the URL.

pszOut

[out] NULL-terminated string with the specified part.

pcchOut

[in/out] Pointer to a value with the number of characters in the *pszOut* buffer. When the function returns, its value will be the number of characters actually written to the buffer.

dwPart

[in] Flags that specify which part of the URL to retrieve. It can have one of the following values:

Flag	Description
URL_PART_HOSTNAME	The host name
URL_PART_PASSWORD	The password
URL_PART_PORT	The port number
URL_PART_SCHEME	The URL scheme
URL_PART_USERNAME	The username

dwFlags

[in] Flag that can be set to keep the URL scheme, in addition to the part that is specified by *dwPart*.

Flag	Description
URL_PARTFLAG_KEEPSCHEME	Keep the URL scheme.

Return Values

Returns an OLE success code if successful. The value pointed to by *pcchEscaped* will be set to the number of characters written to the output buffer, excluding the terminating NULL. If the buffer was too small, E_POINTER is returned, and the value pointed to by *pcchEscaped* will be set to the required buffer size. Otherwise, an OLE error value is returned.

Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

UrlHash

Hashes a URL string.

```
HRESULT UrlHash(
  LPCTSTR pszURL,
```

(continued)

(continued)

```
  LPBYTE pbHash,
  DWORD cbHash
);
```

Parameters

pszURL
[in] Pointer to a NULL-terminated string with the URL.

pbHash
[out] Buffer to receive the hashed array.

cbHash
[in] Number of elements in *pbHash.* It should be no larger than 256.

Return Values

Returns S_OK if successful, or a standard OLE error value otherwise.

Remarks

For example, to hash a URL into a single byte, set *cbHash* = sizeof(BYTE) and *pbHash* = (LPBYTE)&bHashedValue, where bHashedValue is a one-byte buffer. To hash a URL into a DWORD, set *cbHash* = sizeof(DWORD) and *pbHash* = (LPBYTE)&dwHashedValue, where dwHashedValue is a DWORD buffer.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

+ See Also

HashData

UrlIs

Tests whether or not a URL is a specified type.

```
BOOL UrlIs(
  LPCTSTR pszUrl,
  URLIS UrlIs
);
```

Parameters

pszUrl

Pointer to a string containing the URL.

UrlIs

Type of URL to be tested for. *UrlIs* can take one of the following values:

URLIS_URL
A valid URL

URLIS_OPAQUE
An opaque URL

URLIS_NOHISTORY
A "No History" URL

URLIS_FILEURL
A file URL

URLIS_APPLIABLE
Attempt to determine a valid scheme for the URL.

Return Values

For the first four URL types, **UrlIs** returns TRUE if the URL is the specified type, or FALSE if not. With the URLIS_APPLIABLE type, **UrlIs** will attempt to determine the URL scheme. If it is able to determine a scheme, it will return TRUE, or FALSE if not.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

UrlIsFileUrl

Tests a URL to determine if it is a file URL.

```
BOOL UrlIsFileUrl(
  LPCTSTR pszUrl
);
```

Parameters

pszUrl
Pointer to a NULL-terminated string containing the URL.

Return Values

Returns a non-zero value if the URL is a file URL, or zero otherwise.

Remarks

A file URL has the form "File://*xxx*". **UrlIsFileUrl** is actually one of the following macros, depending on whether ANSI or Unicode is selected.

```
#define  UrlIsFileUrlA(pszURL) UrlIsA(pszURL, URLIS_FILEURL)
#define  UrlIsFileUrlW(pszURL) UrlIsW(pszURL, URLIS_FILEURL)
```

Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

See Also

UrlIs

UrlIsNoHistory

Returns whether or not a URL is a No History URL.

```
BOOL UrlIsNoHistory(
  LPCTSTR pszURL,
);
```

Parameters

pszURL
[in] NULL-terminated string with the URL.

Return Values

Returns a non-zero value if the URL is a No History URL, or zero otherwise.

Remarks

A No History URL is a URL that browsers typically do not include in their navigation history. This function is equivalent to:

```
UrlIs(pszURL, URLIS_NOHISTORY)
```

Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

See Also

UrlIs

UrlIsOpaque

Returns whether a URL is opaque.

```
BOOL UrlIsOpaque(
  LPCTSTR pszURL,
);
```

Parameters

pszURL
  [in] NULL-terminated string with the URL.

Return Values

Returns a non-zero value if the URL is opaque, or zero otherwise.

Remarks

An opaque URL is one that is nonhierarchical and cannot be parsed. For example, mailto:xyz@somecompany.com is an opaque URL. This function is equivalent to:

```
UrlIsA(pszURL, URLIS_OPAQUE)
```

Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

See Also

UrlIs

UrlUnEscape

Converts escape sequences back into ordinary characters.

```
HRESULT UrlUnEscape(
  LPTSTR pszURL,
  LPTSTR pszUnEscaped,
  LPDWORD pcchUnEscaped,
  DWORD dwFlags
);
```

Parameters

pszURL

[in/out] Pointer to a NULL-terminated string with the URL. If *dwFlags* is set to URL_UNESCAPE_INPLACE, the converted string is returned through this parameter.

pszUnEscaped

[out] Pointer to a NULL-terminated string containing the unescaped version of *pszURL*. If URL_UNESCAPE_INPLACE is set in *dwFlags*, this parameter is ignored.

pcchUnEscaped

[in/out] Number of characters in the buffer pointed to by *pszEscaped*. On entry, the value *pcchEscaped* points to is set to the size of the buffer. When the function returns, the value *pcchEscaped* points to is set to the number of characters written to that buffer, not counting the terminating NULL character. If an E_POINTER error code is returned, the buffer was too small, and the value *pcchEscaped* points to is set to the required number of characters in the buffer. If any other errors are returned, the value that *pcchEscaped* points to is undefined.

dwFlags

[in] Flags that control which characters are unescaped. It can be a combination of the following flags.

Flag	Description
URL_DONT_UNESCAPE_EXTRA_INFO	Don't convert the # or ? character, or any characters following them in the string.
URL_UNESCAPE_INPLACE	Use *pszURL* to return the converted string instead of *pszUnEscaped.*

Return Values

Returns an OLE success code if successful. If the URL_UNESCAPE_INPLACE flag is not set, the value pointed to by *pcchUnEscaped* will be set to the number of characters in the output buffer pointed to by *pszEscaped.* Returns E_POINTER if the URL_UNESCAPE_INPLACE flag is not set and the output buffer is too small. The *pcchUnEscaped* parameter will be set to the required buffer size. Otherwise, returns an OLE error value.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

UrlUnEscapeInPlace

A macro that converts escape sequences back into ordinary characters and overwrites the original string.

```
HRESULT UrlUnEscapeInPlace(
  LPTSTR pszURL,
  DWORD dwFlags
);
```

Parameters

pszURL

[in/out] Pointer to a NULL-terminated string that contains the URL. The converted string is returned through this parameter.

dwFlags

[in] Flags that control which characters are unescaped.

URL_DONT_UNESCAPE_EXTRA_INFO	Don't convert the # or ? character, or any characters following them in the string.

Return Values

Returns S_OK if successful, or a standard OLE error value otherwise.

Remarks

UrlUnEscapeInPlace is equivalent to:

```
UrlUnescape(pszUrl, NULL, NULL, dwFlags | URL_UNESCAPE_INPLACE)
```

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

+ See Also

UrlUnEscape

Registry Data Types

The following data types can be used to specify the type of a registry value:

REG_BINARY

Binary data in any form.

REG_DWORD

32-bit number.

REG_QWORD

64-bit number.

REG_DWORD_LITTLE_ENDIAN

32-bit number in little-endian format. This is equivalent to REG_DWORD.

In little-endian format, a multibyte value is stored in memory from the lowest byte (the "little end") to the highest byte. For example, the value 0x12345678 is stored as (0x78 0x56 0x34 0x12) in little-endian format.

Windows NT and Windows 95 are designed to run on little-endian computer architectures. A user may connect to computers that have big-endian architectures, such as some UNIX systems.

REG_QWORD_LITTLE_ENDIAN

A 64-bit number in little-endian format. This is equivalent to REG_QWORD.

REG_DWORD_BIG_ENDIAN

32-bit number in big-endian format.

In big-endian format, a multibyte value is stored in memory from the highest byte (the "big end") to the lowest byte. For example, the value 0x12345678 is stored as (0x12 0x34 0x56 0x78) in big-endian format.

REG_EXPAND_SZ

Null-terminated string that contains unexpanded references to environment variables (for example, "%PATH%"). It will be a UNICODE or ANSI string, depending on whether you use the UNICODE or ANSI functions.

REG_LINK

Unicode symbolic link.

REG_MULTI_SZ

Array of null-terminated strings that are terminated by two null characters.

REG_NONE

No defined value type.

REG_RESOURCE_LIST

Device-driver resource list.

REG_SZ

Null-terminated string. It will be a Unicode or ANSI string, depending on whether you use the Unicode or ANSI functions.

REGSAM

Data type used for specifying the security access attributes in the registry. A REGSAM value can be one or more of the following values:

KEY_ALL_ACCESS

Combination of KEY_QUERY_VALUE, KEY_ENUMERATE_SUB_KEYS, KEY_NOTIFY, KEY_CREATE_SUB_KEY, KEY_CREATE_LINK, and KEY_SET_VALUE access.

KEY_CREATE_LINK

Permission to create a symbolic link.

KEY_CREATE_SUB_KEY

Permission to create subkeys.

KEY_ENUMERATE_SUB_KEYS

Permission to enumerate subkeys.

KEY_EXECUTE
Permission for read access.

KEY_NOTIFY
Permission for change notification.

KEY_QUERY_VALUE
Permission to query subkey data.

KEY_READ
Combination of KEY_QUERY_VALUE, KEY_ENUMERATE_SUB_KEYS, and KEY_NOTIFY access.

KEY_SET_VALUE
Permission to set subkey data.

KEY_WRITE
Combination of KEY_SET and KEY_CREATE_SUB_KEY access.

Registry Functions

AssocCreate

Returns a pointer to an **IQueryAssociations** interface.

```
HRESULT AssocCreate(
  CLSID clsid,
  REFIID riid,
  LPVOID *pqa
);
```

Parameters

clsid
[in] CLSID of the object that exposes the interface. This parameter must be set to CLSID_QueryAssociations.

riid
[in] REFIID of the interface. This parameter must be set to IID_IQueryAssociations.

pqa
[out] Pointer to the interface.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

AssocQueryKey

Searches for and retrieves a file association-related key from the registry.

```
HRESULT AssocQueryKey(
  ASSOCF flags,
  ASSOCKEY key,
  LPCTSTR pszAssoc,
  LPCTSTR pszExtra,
  HKEY *phkeyOut
);
```

Parameters

flags

[in] Flags that can be used to control the search. It can be any combination of **ASSOCF** values, except that only one ASSOCF_INIT value can be included.

key

[in] **ASSOCKEY** value that specifies the type of key that is to be returned.

pszAssoc

[in] Pointer to a NULL-terminated string that is used to determine the root key. Four types of strings can be used.

String type	Description
File name extension	A file name extension, such as .txt.
CLSID	A CLSID GUID in the standard "{GUID}" format.
ProgID	An application's ProgID, such as Word.Document.8.
Executable name	The name of an application's .exe file. The ASSOCF_OPEN_BYEXENAME flag must be set in *flags*.

pszExtra

[in] Pointer to an optional NULL-terminated string with additional information about the location of the string. It is normally set to a shell verb such as **open**. Set this parameter to NULL if it is not used.

phkeyOut
[out] Pointer to the key's HKEY value.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

This function is a wrapper for the **IQueryAssociations** interface. It is intended to simplify the process of using the interface. For further discussion of how the file association functions work, see **IQueryAssociations**.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

AssocQueryString

Searches for and retrieves a file association-related string from the registry.

```
HRESULT AssocQueryString(
  ASSOCF flags,
  ASSOCSTR str,
  LPCTSTR pszAssoc,
  LPCTSTR pszExtra,
  LPTSTR pszOut,
  DWORD *pcchOut
);
```

Parameters

flags
[in] Flags that can be used to control the search. It can be any combination of **ASSOCF** values, except that only one ASSOCF_INIT value can be included.

str
[in] **ASSOCSTR** value that specifies the type of string that is to be returned.

pszAssoc

[in] Pointer to a NULL-terminated string that is used to determine the root key. Four types of strings can be used.

String type	Description
File name extension	A file name extension, such as .txt.
CLSID	A CLSID GUID in the standard "{GUID}" format.
ProgID	An application's ProgID, such as Word.Document.8.
Executable name	The name of an application's .exe file. The ASSOCF_OPEN_BYEXENAME flag must be set in *flags*.

pszExtra

[in] Optional NULL-terminated string with additional information about the location of the string. It is normally set to a shell verb such as **open**. Set this parameter to NULL if it is not used.

pszOut

[out] NULL-terminated string used to return the requested string.

pcchOut

[in/out] Pointer to a value that is set to the number of characters in the *pszOut* buffer. When the function returns, it will be set to the number of characters actually placed in the buffer. If the ASSOCF_NOTRUNCATE flag is set in *flags* and the buffer specified in *pcchOut* is too small, the function returns E_POINTER and the value is set to the required size of the buffer.

Return Values

Returns S_OK if successful, E_POINTER if the buffer is too small, or an OLE error value otherwise.

Remarks

This function is a wrapper for the **IQueryAssociations** interface. It is intended to simplify the process of using this interface. For further discussion of how the file association functions work, see **IQueryAssociations**.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

AssocQueryStringByKey

Searches for and retrieves a file association-related string from the registry starting from a specified key.

```
HRESULT AssocQueryStringByKey(
  ASSOCF flags,
  ASSOCSTR str,
  HKEY hkAssoc,
  LPCTSTR pszExtra,
  LPTSTR pszOut,
  DWORD *pcchOut
);
```

Parameters

flags

[in] Flags that can be used to control the search. It can be any combination of **ASSOCF** values, except that only one ASSOCF_INIT value can be included.

str

[in] **ASSOCSTR** value that specifies the type of string that is to be returned.

hkAssoc

[in] HKEY value of the key that will be used as a root key. The search will look only below this key.

pszExtra

[in] Pointer to an optional NULL-terminated string with additional information about the location of the string. It is normally set to a shell verb such as **open**. Set this parameter to NULL if it is not used.

pszOut

[out] Pointer to a NULL-terminated string used to return the requested string.

pcchOut

[in/out] Pointer to a value that is set to the number of characters in the *pszOut* buffer. When the function returns, it will be set to the number of characters actually placed in the buffer. If the ASSOCF_NOTRUNCATE flag is set in *flags* and the buffer specified in *pcchOut* is too small, the function returns E_POINTER and the value is set to the required size of the buffer.

Return Values

Returns S_OK if successful, E_POINTER if the buffer is too small, or an OLE error value otherwise.

Remarks

This function is a wrapper for the **IQueryAssociations** interface. It is intended to simplify the process of using this interface. For further discussion of how the file association functions work, see *IQueryAssociations*.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHCopyKey

Recursively copies the subkeys and values of the source subkey to the destination key.

```
DWORD SHCopyKey(
  HKEY hkeySrc,
  LPCTSTR szSrcSubKey,
  HKEY hkeyDest,
  DWORD fReserved
);
```

Parameters

hkeySrc
Handle to the source key (for example, HKEY_CURRENT_USER).

szSrcSubKey
Subkey whose subkeys and values are to be copied.

hkeyDest
Destination key.

fReserved
Reserved. Pass NULL.

Return Values

Returns ERROR_SUCCESS if successful, or one of the nonzero error codes defined in Winerror.h otherwise. Use **FormatMessage** with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a generic description of the error.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHDeleteEmptyKey

Deletes an empty key.

```
DWORD SHDeleteEmptyKey(
  HKEY hkey,
  LPCTSTR pszSubKey
);
```

Parameters

hkey

Handle to the currently open key, or any of the following predefined values:

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_CONFIG
- HKEY_CURRENT_USER
- HKEY_DYN_DATA (Windows 95 only)
- HKEY_LOCAL_MACHINE
- HKEY_PERFORMANCE_DATA (Windows NT only)
- HKEY_USERS

pszSubKey

Address of a null-terminated string specifying the name of the key to delete.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a generic description of the error.

Remarks

Note SHDeleteEmptyKey will not delete a key if it contains any subkeys or values. Use **SHDeleteKey** instead.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHDeleteKey

Deletes a subkey and all its descendants. The function will remove the key and all of the key's values from the registry.

```
DWORD SHDeleteKey(
  HKEY hkey,
  LPCTSTR pszSubKey
);
```

Parameters

hkey

Handle to the currently open key, or any of the following predefined values:

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_CONFIG
- HKEY_CURRENT_USER
- HKEY_DYN_DATA (Windows 95 only)
- HKEY_LOCAL_MACHINE
- HKEY_PERFORMANCE_DATA (Windows NT only)
- HKEY_USERS

pszSubKey

Address of a null-terminated string specifying the name of the key to delete.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a generic description of the error.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHDeleteValue

Deletes a named value from the specified registry key.

```
DWORD SHDeleteValue(
  HKEY hkey,
  LPCTSTR pszSubKey,
  LPCTSTR pszValue
);
```

Parameters

hkey

Handle to the currently open key, or any of the following predefined values:

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_CONFIG
- HKEY_CURRENT_USER
- HKEY_DYN_DATA (Windows 95 only)
- HKEY_LOCAL_MACHINE
- HKEY_PERFORMANCE_DATA (Windows NT only)
- HKEY_USERS

pszSubKey

Address of a null-terminated string specifying the name of the subkey for which to change the value.

pszValue

Address of the value to be deleted.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a generic description of the error.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHEnumKeyEx

Enumerates the subkeys of the specified open registry key.

```
DWORD SHEnumKeyEx(
  HKEY hkey,
  DWORD dwIndex,
  LPTSTR pszName,
  LPDWORD pcchName
);
```

Parameters

hkey

Handle to the currently open key, or any of the following predefined values:

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_CONFIG
- HKEY_CURRENT_USER
- HKEY_DYN_DATA (Windows 95 only)
- HKEY_LOCAL_MACHINE
- HKEY_PERFORMANCE_DATA (Windows NT only)
- HKEY_USERS

dwIndex

Index of the subkey to retrieve. This parameter should be zero for the first call and incremented for subsequent calls.

pszName

Address of a character buffer that receives the enumerated key name.

pcchName

Address of a DWORD that, on entry, contains the size of the buffer at *pszName*. On exit, this contains the number of characters that were copied to *pszName*.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a textual description of the error.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHEnumValue

Enumerates the values of the specified open registry key.

```
DWORD SHEnumValue(
  HKEY hkey,
  DWORD dwIndex,
  LPTSTR pszValueName,
  LPDWORD pcchValueName,
  LPDWORD pdwType,
  LPVOID pvData,
  LPDWORD pcbData
);
```

Parameters

hkey

Handle to the currently open key, or any of the following predefined values:

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_CONFIG
- HKEY_CURRENT_USER
- HKEY_DYN_DATA (Windows 95 only)
- HKEY_LOCAL_MACHINE
- HKEY_PERFORMANCE_DATA (Windows NT only)
- HKEY_USERS

dwIndex

Index of the value to retrieve. This parameter should be zero for the first call and incremented for subsequent calls.

pszValueName

Address of a character buffer that receives the enumerated value name. The size of this buffer is specified in *pcchVlaueName*.

pcchValueName
Address of a DWORD that, on entry, contains the size of the buffer at *pszValueName*. On exit, this contains the number of characters that were copied to *pszValueName*.

pdwType
Address of a DWORD that receives the data type of the value. These are the same values as those described under the *lpType* parameter of **RegEnumValue**.

pvData
Address of a buffer that receives the data for the value entry. The size of this buffer is specified in *pcbData*. This parameter can be NULL if the data is not required.

pcbData
Address of a DWORD that, on entry, contains the size of the buffer at *pvData*. On exit, this contains the number of bytes that were copied to *pvData*.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a textual description of the error.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHGetValue

Retrieves a registry value.

```
DWORD SHGetValue(
  HKEY hkey,
  LPCTSTR pszSubKey,
  LPCTSTR pszValue,
  LPDWORD pdwType,
  LPVOID pvData,
  LPDWORD pcbData
);
```

Parameters

hkey

Handle to the currently open key, or any of the following predefined values:

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_CONFIG
- HKEY_CURRENT_USER
- HKEY_DYN_DATA (Windows 95 only)
- HKEY_LOCAL_MACHINE
- HKEY_PERFORMANCE_DATA (Windows NT only)
- HKEY_USERS

pszSubKey

Address of a null-terminated string that specifies the name of the subkey from which to get the value.

pszValue

Address of the value.

pdwType

Type of value. For more information, see *Registry Data Types*.

pvData

Address of the destination data buffer.

pcbData

Size of the destination data buffer.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a generic description of the error.

Remarks

Note If your application must set/get a series of values in the same key, it is better to open the key once and set/get the values with the regular Win32 registry functions rather than use this function repeatedly.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.

Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHQueryInfoKey

Retrieves information about a specified registry key.

```
DWORD SHQueryInfoKey(
  HKEY hkey,
  LPDWORD pcSubKeys,
  LPDWORD pcchMaxSubKeyLen,
  LPDWORD pcValues,
  LPDWORD pcchMaxValueNameLen
);
```

Parameters

hkey

Handle to the currently open key, or any of the following predefined values:

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_CONFIG
- HKEY_CURRENT_USER
- HKEY_DYN_DATA (Windows 95 only)
- HKEY_LOCAL_MACHINE
- HKEY_PERFORMANCE_DATA (Windows NT only)
- HKEY_USERS

pcSubKeys

Address of a DWORD that receives the number of subkeys under the specified key.

pcchMaxSubKeyLen

Address of a DWORD that receives the number of characters in the name of the subkey with the largest name.

pcValues

Address of a DWORD that receives the number of values under the specified key.

pcchMaxValueNameLen

Address of a DWORD that receives the number of characters in the name of the value with the largest name.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a textual description of the error.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHQueryValueEx

Opens a registry key and queries it for a specific value.

```
DWORD SHQueryValueEx(
  HKEY hkey,
  LPCTSTR pszValue,
  LPDWORD pdwReserved,
  LPDWORD pdwType,
  LPVOID pvData,
  LPDWORD pcbData
);
```

Parameters

hkey

Handle to the currently open key, or any of the following predefined values:

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_CONFIG
- HKEY_CURRENT_USER
- HKEY_DYN_DATA (Windows 95 only)
- HKEY_LOCAL_MACHINE
- HKEY_PERFORMANCE_DATA (Windows NT only)
- HKEY_USERS

pszValue

Address of the null-terminated string that contains the name of the value to be queried.

pdwReserved

Reserved; must be null.

pdwType

Address of the variable that receives the key's value type. For more information, see *Registry Data Types*.

pvData

Address of the buffer that receives the value's data. This parameter can be NULL if the data is not required.

pcbData

Address of the variable that specifies the size, in bytes, of the buffer pointed to by the *pvData* parameter. When the function returns, this variable contains the size of the data copied to *pvData*.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a generic description of the error.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHRegCloseUSKey

Closes a handle to a user-specific registry key.

```
DWORD SHRegCloseUSKey(
  HUSKEY hUSKey,
);
```

Parameters

hUSKey

Handle to a user-specific key that is currently open.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. Use **FormatMessage** with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a generic description of the error.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHRegCreateUSKey

Creates or opens a user-specific registry key.

```
LONG SHRegCreateUSKey(
  LPCTSTR pszPath,
  REGSAM samDesired,
  HUSKEY hRelativeUSKey,
  PHUSKEY phNewUSKey,
  DWORD dwFlags
);
```

Parameters

pszPath
[in] Address of NULL-terminated string that contains the registry path of the key to be created or opened.

samDesired
[in] Desired security access. For more information on security access, see REGSAM.

hRelativeUSKey
[in] Key to be used as a base for relative paths. If *pszPath* is a relative path, the key it specifies will be relative to *hRelativeUSKey*. If *pszPath* is an absolute path, set *hRelativeUSKey* to NULL. The key will then be created under HKLM or HKCU, depending the value of *dwFlags*.

phNewUSKey
[out] Address of an HUSKEY that receive the handle to the new key.

dwFlags
[in] Base key under which the key should be opened. This can be one or more of the following values:

SHREGSET_HKCU	Create/open the key under HKEY_CURRENT_USER. Only creates a key if it is empty.
SHREGSET_FORCE_HKCU	Create/open the key under HKEY_CURRENT_USER, even if not empty.

SHREGSET_HKLM	Create/open the key under HKEY_LOCAL_MACHINE. Only creates a key if it is empty.
SHREGSET_FORCE_HKLM	Create/open the key under HKEY_LOCAL_MACHINE, even if not empty.
SHREGSET_DEFAULT	Create/open the key under both HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a generic description of the error.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHRegDeleteEmptyUSKey

Deletes an empty user-specific registry key.

```
LONG SHRegDeleteEmptyUSKey
 (HUSKEY hUSKey,
 LPCSTR pszValue,
 SHREGDEL_FLAGS delRegFlags
);
```

Parameters

hUSKey
Handle to the currently open user-specific key.

pszValue
Address of the null-terminated string that specifies the empty user-defined registry key to be deleted.

delRegFlags
One of the **SHREGDEL_FLAGS** that specifies from which base key the key will be deleted.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a generic description of the error.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

See Also

SHRegDeleteUSValue

SHRegDeleteUSValue

Deletes a user-specific registry value.

```
LONG SHRegDeleteUSValue(
  HUSKEY hUSKey,
  LPCTSTR pszValue,
  SHREGDEL_FLAGS delRegFlags
);
```

Parameters

hUSKey
Handle to the currently open user-specific key.

pszValue
Address of the null-terminated string that names the value to remove.

delRegFlags
One of the **SHREGDEL_FLAGS** that specifies from which base key the value will be deleted.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a generic description of the error.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

See Also

SHRegDeleteEmptyUSKey

SHRegDuplicateHKey

Duplicates a registry key's HKEY handle.

```
HKEY SHRegDuplicateHKey(
  HKEY hkey
);
```

Parameters

hkey
[in] HKEY handle to be duplicated.

Return Values

Returns a duplicate of the handle specified in *hkey*.

Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.

Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHRegEnumUSKey

Enumerates the subkeys of a user-specific key.

```
DWORD SHRegEnumUSKey(
  HUSKEY hUSKey,
  DWORD dwIndex,
  LPSTR pszName,
  LPDWORD pcchName,
  SHREGENUM_FLAGS enumRegFlags
);
```

Parameters

hUSKey
Handle to the currently open user-specific key.

dwIndex
Index of the subkey to retrieve. This parameter should be zero for the first call and incremented for subsequent calls.

pszName
Address of a character buffer that receives the enumerated key name.

pcchName
Address of a DWORD that, on entry, contains the size of the buffer at *pszName*. On exit, this contains the number of characters that were copied to *pszName*.

enumRegFlags
One of the **SHREGENUM_FLAGS** that specifies the base key in which the enumeration should take place.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a textual description of the error.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.

Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHRegEnumUSValue

Enumerates the values of the specified user-specific registry key.

```
DWORD SHRegEnumUSValue(
  HUSKEY hUSkey,
  DWORD dwIndex,
  LPTSTR pszValueName,
  LPDWORD pcchValueNameLen,
  LPDWORD pdwType,
  LPVOID pvData,
  LPDWORD pcbData,
  SHREGENUM_FLAGS enumRegFlags
);
```

Parameters

hUSKey
Handle to the currently open user-specific key.

dwIndex
Index of the value to retrieve. This parameter should be zero for the first call and incremented for subsequent calls.

pszValueName
Address of a character buffer that receives the enumerated value name. The size of this buffer is specified in *pcchValueName*.

pcchValueNameLen
Address of a DWORD that, on entry, contains the size of the buffer at *pszValueName*. On exit, this contains the number of characters that were copied to *pszValueName*.

pdwType
Address of a DWORD that receives the data type of the value. These are the same values as those described under the *lpType* parameter of **RegEnumValue**.

pvData
Address of a buffer that receives the data for the value entry. The size of this buffer is specified in *pcbData*. This parameter can be NULL if the data is not required.

pcbData
Address of a DWORD that, on entry, contains the size of the buffer at *pvData*. On exit, this contains the number of bytes that were copied to *pvData*.

enumRegFlags
One of the **SHREGENUM_FLAGS** that specifies the base key in which the enumeration should take place.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a textual description of the error.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHRegGetBoolUSValue

Gets a user-specific Boolean value from the registry.

```
BOOL SHRegGetBoolUSValue(
  LPCTSTR pszSubKey,
  LPCTSTR pszValue,
  BOOL fIgnoreHKCU,
  BOOL fDefault
);
```

Parameters

pszSubKey

[in] Pointer to a null-terminated string with the name of the subkey relative to HKLM and HKCU. For example: "Software\MyCompany\MyProduct".

pszValue

[in] Pointer to a null-terminated string that specifies the name of the value.

fIgnoreHKCU

[in] Variable that specifies which key to look under. When set to TRUE, **SHRegGetUSValue** ignores HKCU and returns a value from HKLM.

fDefault

[in] Value that will be returned if there is no registry value.

Return Values

Returns either the value from the registry, or *fDefault* if none is found.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHRegGetPath

Gets a file path from the registry, expanding environment variables as needed.

```
DWORD SHRegGetPath(
  HKEY hkey,
  LPCTSTR pszSubkey,
  LPCTSTR pszValue,
  LPTSTR pszPath,
  DWORD dwFlags
);
```

Parameters

hkey
[in] Handle to a key that is currently open, or a registry root key.

pszSubkey
[in] Pointer to a NULL-terminated string that contains the name of the subkey.

pszValue
[in] Pointer to a NULL-terminated string that contains the name of the value that holds the unexpanded path string.

pszPath
[out] Buffer to hold the expanded path. The size of this buffer should be set to MAX_PATH to ensure that it is large enough to hold the returned string.

dwFlags
Reserved for future use.

Return Values

Returns ERROR_SUCCESS if successful, or a Windows error code otherwise.

Remarks

The data type of the specified registry value must be either REG_EXPAND_SZ or REG_SZ. If it has the REG_EXPAND_SZ type, any environment variables in the registry string will be expanded with **ExpandEnvironmentStrings**. If it has the REG_SZ data type, environment variables will not be expanded and the string pointed to by *pszPath* will be identical to the string in the registry.

For Windows 2000, the following environment strings will be replaced by their equivalent path.

Environment string	Folder
%USERPROFILE%	The current user's profile folder
%ALLUSERSPROFILE%	The All Users profile folder
%ProgramFiles%	The Program Files folder
%SystemRoot%	The system root folder
%SystemDrive%	The system drive letter

Note %USERPROFILE% is relative to the user making the call. This function does not work if the user is being impersonated from a service.

The environment variables listed in the above table might not all be set on any particular system. If an environment variable is not set, it will not be unexpanded. In particular, none of these variables are set for the default environment of Windows 95 or Windows 98. The %ProgramFiles% variable is new for Windows 2000, and will typically not be set on Windows NT 4.0 systems.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHRegGetUSValue

Retrieves a user-specific registry value.

```
LONG SHRegGetUSValue(
  LPCTSTR pszSubKey,
    LPCTSTR pszValue,
    LPDWORD pdwType,
    LPVOID pvData,
    LPDWORD pcbData,
    BOOL fIgnoreHKCU,
    LPVOID pvDefaultData,
    DWORD dwDefaultDataSize
);
```

Parameters

pszSubKey

[in] Pointer to a null-terminated string with the name of the subkey relative to HKLM and HKCU. For example: "Software\MyCompany\MyProduct".

pszValue

[in] Pointer to a null-terminated string with the name of the value.

pdwType

[out] Pointer to a variable that receives the key's value type. For more information, see *Registry Data Types*.

pvData

[out] Pointer to a buffer that will receive the value's data.

pcbData

[in/out] Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by *pvData*. When **SHRegGetUSValue** returns, *pcbData* contains the size of the data copied to *pvData*.

fIgnoreHKCU

[in] Variable that specifies which key to look under. When set to TRUE, **SHRegGetUSValue** ignores HKCU and returns a value from HKLM.

pvDefaultData

[out] Pointer to a buffer that will receive the value's default data.

dwDefaultDataSize

[in] Length, in bytes, of buffer pointed to by *pvDefaultData*.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a generic description of the error.

Remarks

This function opens the key each time it is used. If your code involves getting a series of values from the same key, it is more efficient to open the key once with **SHRegOpenUSKey** and then use **SHRegQueryUSValue** to retrieve the data.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHRegOpenUSKey

Opens a user-specific registry key.

```
LONG SHRegOpenUSKey(
  LPCTSTR pszPath,
  REGSAM samDesired,
  HUSKEY hRelativeUSKey,
  PHUSKEY phNewUSKey,
  BOOL fIgnoreHKCU
);
```

Parameters

pszPath
[in] Pointer to a null-terminated string with the name of the subkey.

samDesired
[in] Desired security access. For more information on security access, see REGSAM.

hRelativeUSKey
[in] Key to be used as a base for relative paths. If *pszPath* is a relative path, the key it specifies will be relative to *hRelativeUSKey*. If *pszPath* is an absolute path, set *hRelativeUSKey* to NULL.

phNewUSKey
[out] Pointer to the handle of the opened key.

fIgnoreHKCU
[in] Variable that specifies which key to look under. When set to TRUE, **SHRegGetUSValue** ignores HKCU and returns a value from HKLM.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a generic description of the error.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHRegQueryInfoUSKey

Retrieves information about a specified user-specific registry key.

```
DWORD SHRegQueryInfoUSKey(
  HUSKEY hUSKey,
  LPDWORD pcSubKeys,
  LPDWORD pcchMaxSubKeyLen,
  LPDWORD pcValues,
  LPDWORD pcchMaxValueNameLen
);
```

Parameters

hUSKey
Handle to the user-specific key that is currently open.

pcSubKeys
Address of a DWORD that receives the number of subkeys under the specified key.

pcchMaxSubKeyLen
Address of a DWORD that receives the number of characters in the largest subkey name.

pcValues
Address of a DWORD that receives the number of values under the specified key.

pcchMaxValueNameLen
Address of a DWORD that receives the number of characters in the largest value name.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a textual description of the error.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHRegQueryUSValue

Retrieves the type and data for a specified name associated with an open USKEY.

```
LONG SHRegQueryUSValue(
  HUSKEY hUSKey,
  LPCTSTR pszValue,
  LPDWORD pdwType,
  LPVOID pvData,
  LPDWORD pcbData,
  BOOL fIgnoreHKCU,
  LPVOID pvDefaultData,
  DWORD dwDefaultDataSize
);
```

Parameters

hUSKey

[in] Handle to the currently open USKEY, or one of the following predefined values:

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_CONFIG
- HKEY_CURRENT_USER
- HKEY_DYN_DATA (Windows 95 only)
- HKEY_LOCAL_MACHINE
- HKEY_PERFORMANCE_DATA (Windows NT only)
- HKEY_USERS

pszValue

[out] Address of the null-terminated string that contains the name of the value to be queried.

pdwType

[out] Address of the variable that receives the key's value type. For more information, see *Registry Data Types*.

pvData

[out] Address of the buffer that receives the value's data. This parameter can be NULL if the data is not required.

pcbData

[in/out] Address of the variable that specifies the size, in bytes, of the buffer pointed to by the *pvData* parameter. When the function returns, this variable contains the size of the data copied to *pvData*.

fIgnoreHKCU

[in] Variable that specifies which key to look under. When set to TRUE, **SHRegQueryUSValue** ignores HKCU and returns a value from HKLM.

pvDefaultData

[out] Address of the default data.

dwDefaultDataSize

[in] Length, in bytes, of the default data.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a generic description of the error.

Remarks

If you only need to read a single value, **SHRegGetUSValue** will both open the key and return the value. To use **SHRegQueryUSValue**, you must first open the key with **SHRegOpenUSKey**. However, once the key is opened, you can use **SHRegQueryUSValue** as many times as necessary. If you need to get more than one value from the same key, using multiple calls to **SHRegQueryUSValue** is usually more efficient than **SHRegGetUSValue**, as the key is only opened once.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHRegSetPath

Takes a file path, replaces folder names with environment strings, and places the resulting string in the registry.

```
DWORD SHRegSetPath(
  HKEY hkey,
  LPCTSTR pszSubkey,
  LPCTSTR pszValue,
  LPCTSTR pszPath,
  DWORD dwFlags
);
```

Parameters

hkey
  [in] Handle to a key that is currently open, or a registry root key.

pszSubkey
  [in] Pointer to a NULL-terminated string containing the name of an existing subkey. If the subkey does not exist, **SHRegSetPath** will fail.

pszValue
  [in] Pointer to a NULL-terminated string with the name of the value to hold the path string.

pszPath
  [in] Pointer to a NULL-terminated string with a fully qualified file path.

dwFlags
  Reserved for future use.

Return Values

Returns ERROR_SUCCESS if successful, or a Windows error code otherwise.

Remarks

For Windows 2000, **RegSetPath** uses **PathUnExpandEnvStrings** to convert folder names to their corresponding environment string. If any environment variables were substituted, the registry value will be set with the REG_EXPAND_SZ data type. Otherwise, it will be set with the REG_SZ data type.

The following folders' paths will be replaced by their equivalent environment string.

Folder	Environment string
The current user's profile folder	%USERPROFILE%
The All Users profile folder	%ALLUSERSPROFILE%
The Program Files folder	%ProgramFiles%
The system root folder	%SystemRoot%
The system drive letter	%SystemDrive%

Note %USERPROFILE% is relative to the user making the call. This function does not work if the user is being impersonated from a service.

The environment variables listed in the above table might not all be set on any particular system. If an environment variable is not set, it will not be unexpanded. In particular, none of these variables are set for the default environment of Windows 95 or Windows 98. The %ProgramFiles% variable is new for Windows 2000, and will typically not be set on Windows NT 4.0 systems.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHRegSetUSValue

Sets a user-specific registry value.

```
LONG SHRegSetUSValue(
  LPCTSTR pszSubKey,
  LPCTSTR pszValue,
  DWORD dwType,
  LPVOID pvData,
  DWORD cbData,
  DWORD dwFlags
);
```

Parameters

pszSubKey
  [in] Pointer to a null-terminated string with the name of the subkey.

pszValue
  [in] Pointer to a null-terminated string that specifies the name of the value.

dwType
  [in] Type of data to be stored. This parameter must be the REG_SZ type. For more information, see *Registry Data Types.*

pvData

[in] Address of a null-terminated string that contains the value to be set for the specified key.

cbData

[in] Length, in bytes, of the string pointed to by the *pvData* parameter, not including the terminating null character.

dwFlags

[in] Flags indicating where the data should be written.

SHREGSET_HKCU	Write to HKCU if empty.
SHREGSET_FORCE_HKCU	Write to HKCU.
SHREGSET_HKLM	Write to HKLM if empty.
SHREGSET_FORCE_HKLM	Write to HKLM.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a generic description of the error.

Remarks

This function opens the key each time it is used. If your code involves setting a series of values in the same key, it is more efficient to open the key once with **SHRegOpenUSKey** and then use **SHRegWriteUSValue** to write the data.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHRegWriteUSValue

Writes a user-specific registry value.

```
LONG SHRegWriteUSValue(
  HUSKEY hUSKey,
  LPCTSTR pszValue,
  DWORD dwType,
  LPVOID pvData,
  DWORD cbData,
  DWORD dwFlags
);
```

Parameters

hUSKey

[in] Handle to the currently open USKEY, or any of the following predefined values:

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_CONFIG
- HKEY_CURRENT_USER
- HKEY_DYN_DATA (Windows 95 only)
- HKEY_LOCAL_MACHINE
- HKEY_PERFORMANCE_DATA (Windows NT only)
- HKEY_USERS

pszValue

[in] Address of the null-terminated string that specifies the name of the value.

dwType

[in] Type of data to be stored. This parameter must be the REG_SZ type. For more information, see *Registry Data Types*.

pvData

[in] Address of a null-terminated string that contains the value to be set for the specified key.

cbData

[in] Length, in bytes, of the string pointed to by the *pvData* parameter, not including the terminating null character.

dwFlags

[in] Flags indicating where the data should be written.

SHREGSET_HKCU	Write to HKCU if empty.
SHREGSET_FORCE_HKCU	Write to HKCU.
SHREGSET_HKLM	Write to HKLM if empty.
SHREGSET_FORCE_HKLM	Write to HKLM.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a generic description of the error.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHSetValue

Sets the value of a registry key.

```
DWORD SHSetValue(
  HKEY hkey,
  LPCTSTR pszSubKey,
  LPCTSTR pszValue,
  DWORD dwType,
  LPCVOID pvData,
  DWORD cbData
);
```

Parameters

hkey

Handle to the currently open key, or any of the following predefined values:

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_CONFIG
- HKEY_CURRENT_USER
- HKEY_DYN_DATA (Windows 95 only)
- HKEY_LOCAL_MACHINE
- HKEY_PERFORMANCE_DATA (Windows NT only)
- HKEY_USERS

pszSubKey

Address of a null-terminated string that specifies the name of the subkey with which a value is associated. This can be NULL or a pointer to an empty string. In this case, the value will be added to the key identified by the *hKey* parameter.

pszValue

Address of a null-terminated string that specifies the value.

dwType

Type of data to be stored. This parameter must be the REG_SZ type. For more information, see *Registry Data Types*.

pvData

Address of a null-terminated string that contains the value to set for the specified key.

cbData

Length, in bytes, of the string pointed to by the *pvData* parameter, not including the terminating null character.

Return Values

Returns ERROR_SUCCESS if successful, or a nonzero error code defined in Winerror.h otherwise. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a generic description of the error.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHREGDEL_FLAGS

Provides a set of values that indicate from which base key an item will be deleted.

```
typedef enum
{
    SHREGDEL_DEFAULT = 0x00000000,
    SHREGDEL_HKCU    = 0x00000001,
    SHREGDEL_HKLM    = 0x00000010,
    SHREGDEL_BOTH    = 0x00000011,
} SHREGDEL_FLAGS;
```

Members

SHREGDEL_DEFAULT

Deletes from HKEY_CURRENT_USER, or, if the specified item is not found under HKEY_CURRENT_USER, deletes from HKEY_LOCAL_MACHINE.

SHREGDEL_HKCU

Enumerates from HKEY_CURRENT_USER only.

SHREGDEL_HKLM

Enumerates under HKEY_LOCAL_MACHINE only.

SHREGDEL_BOTH

Deletes from both HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlwapi.h.

SHREGENUM_FLAGS

Provides a set of values that indicate the base key that will be used for an enumeration.

```
typedef enum{
    SHREGENUM_DEFAULT = 0x00000000,
    SHREGENUM_HKCU    = 0x00000001,
    SHREGENUM_HKLM    = 0x00000010,
    SHREGENUM_BOTH    = 0x00000011,
} SHREGENUM_FLAGS;
```

Members

SHREGENUM_DEFAULT
Enumerates under HKEY_CURRENT_USER, or, if the specified item is not found in HKEY_CURRENT_USER, enumerates under HKEY_LOCAL_MACHINE.

SHREGENUM_HKCU
Enumerates under HKEY_CURRENT_USER only.

SHREGENUM_HKLM
Enumerates under HKEY_LOCAL_MACHINE only.

SHREGENUM_BOTH
Not used.

Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shlwapi.h.

Color Palette Functions

ColorAdjustLuma

Changes the luminance of a red-green-blue (RGB) value. Hue and saturation are not affected.

```
COLORREF ColorAdjustLuma(
  COLORREF clrRGB,
  int n,
  BOOL fScale
);
```

Parameters

clrRGB

Initial RGB value.

n

Luminance in units of 0.1 percent of the total range. For example, a value of *n* = 50 corresponds to 5 percent of the maximum luminance.

fScale

If *fScale* is set to TRUE, *n* specifies how much to increment or decrement the current luminance. If *fScale* is set to FALSE, *n* specifies the absolute luminance.

Return Values

Returns the modified RGB value.

Remarks

If *fScale* is set to TRUE, *n* can range from -1000 to +1000.

If *fScale* is set to FALSE, *n* can range from 0 to 1000. Available luminance values range from 0 to a maximum. If the requested value is negative or exceeds the maximum, the luminance will be set to either zero or the maximum value, respectively.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

ColorHLSToRGB

Converts colors from hue-luminance-saturation (HLS) to red-green-blue (RGB) format.

```
COLORREF ColorHLSToRGB(
  WORD wHue,
  WORD wLuminance,
  WORD wSaturation
);
```

Parameters

wHue
HLS hue value.

wLuminance
HLS luminance value.

wSaturation
HLS saturation value.

Return Values

Returns the RGB value.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

ColorRGBToHLS

Converts colors from red-green-blue (RGB) to hue-luminance-saturation (HLS) format.

```
VOID ColorRGBToHLS(
  COLORREF clrRGB,
  WORD *pwHue,
  WORD *pwLuminance,
  WORD *pwSaturation
);
```

Parameters

clrRGB
[in] RGB color.

pwHue
[out] HLS hue value.

pwLuminance
[out] HLS luminance value.

pwSaturation
[out] HLS saturation value.

Return Values

No return value.

Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHCreateShellPalette

Creates a halftone palette for the specified device context.

```
HPALETTE SHCreateShellPalette(
  HDC hdc
);
```

Parameters

hdc
Device context.

Return Values

Returns the palette if successful, or zero otherwise.

Remarks

This function behaves the same as **CreateHalftonePalette**, but it is designed to accommodate the differences between Windows 95, Windows NT 4.0, and Windows 2000. The palette that is returned depends on the device context in the following way:

- If *hdc* is set to NULL, a full palette is returned.
- If the device context is palettized, a full palette is returned.
- If the device context is not palettized, a default palette (VGA colors) is returned.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

Miscellaneous

DllInstall

Handles installation and setup for a DLL.

```
STDAPI DllInstall(
  BOOL bInstall,
  LPCWSTR pszCmdLine
);
```

Parameters

bInstall
Value that is set to TRUE if the DLL is being installed, or FALSE if it is being uninstalled.

pszCmdLine
String passed in by **regsvr32** that indicates which setup procedure to use.

Return Values

Returns NOERROR if successful, or an OLE-defined error value otherwise.

Remarks

This function may be implemented and exported by name by a DLL for use during application installation or setup. It is invoked by **regsvr32** to allow the DLL to perform tasks such adding information to the registry.

DllInstall is used only for application installation and setup. It should not be called by an application. It is similar in purpose to **DllRegisterServer** or **DllUnregisterServer**. Unlike these functions, **DllInstall** takes an input string which can be used to specify a variety of different actions. This allows a DLL to be installed in more than one way, based on any criteria that is appropriate.

To use **DllInstall** with **regsvr32**, add a "/i" flag followed by a colon (:) and a string. The string will be passed to the **DllInstall** as the *pszCmdLine* parameter. If you omit the colon and string, *pszCmdLine* will be set to NULL. The following example would be used to install a DLL:

```
regsvr32 /i:"Install_1" dllname.dll.
```

DllInstall is invoked with *bInstall* set to TRUE and *pszCmdLine* set to "Install_1". To uninstall a DLL, use:

```
regsvr32 /u /i:"Install_1" dllname.dll.
```

With both of the above examples, **DllRegisterServer** or **DllUnregisterServer** will also be called. To call **DllInstall** only, add a "/n" flag:

```
regsvr32 /n /i:"Install_1" dllname.dll.
```

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98.
Windows CE: Unsupported.
Header: Declared in shlwapi.h.

HashData

Hashes an array of data.

```
HRESULT HashData(
  LPBYTE pbData,
  DWORD cbData,
  LPBYTE pbHash,
  DWORD cbHash
);
```

Parameters

pbData
[in] Pointer to the data array.

cbData
[in] Number of elements in *pbData*.

pbHash
[out] Value used to return the hashed array.

cbHash
[in] Number of elements in *pbHash*. It should be no larger than 256.

Return Values

Returns S_OK if successful, or a standard OLE error value otherwise.

Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHAutoComplete

Instructs system edit controls to use AutoComplete to help complete URLs or file system paths.

```
HRESULT SHAutoComplete(
  HWND hwndEdit,
  DWORD dwFlags
);
```

Parameters

hwndEdit

[in] Window handle of a system edit control. Typically, this parameter is the handle of an edit control or the edit control embedded in a comboboxex control.

dwFlags

[in] Flags to control the operation of **SHAutoComplete**. The first four are used to override the Internet Explorer registry settings. The user can change these settings manually by launching the Internet Options property sheet from the Tools menu and clicking the Advanced tab. The last five can be used to specify which files or URLs will be available for autoappend or autosuggest operations.

Flag	Description
SHACF_AUTOAPPEND_FORCE_OFF	Ignore the registry default and force the autoappend feature off.
SHACF_AUTOAPPEND_FORCE_ON	Ignore the registry value and force the autoappend feature on. The completed string will be displayed in the edit box with the added characters highlighted.

Flag	Description
SHACF_AUTOSUGGEST_FORCE_OFF	Ignore the registry default and force the autosuggest feature off.
SHACF_AUTOSUGGEST_FORCE_ON	Ignore the registry value and force the autosuggest feature on. A selection of possible completed strings will be displayed as a dropdown list, below the edit box.
SHACF_DEFAULT	SHACF_FILESYSTEM | SHACF_URLALL.
SHACF_FILESYSTEM	Include the file system as well as virtual folders such as Desktop or Control Panel.
SHACF_FILESYS_ONLY	Include only the file system. Do not include virtual folders such as Desktop or Control Panel.
SHACF_URLALL	SHACF_URLHISTORY | SHACF_URLMRU.
SHACF_URLHISTORY	Include the URLs in the user's History list.
SHACF_URLMRU	Include the URLs in the user's Recently Used list.

Return Values

Returns S_OK if successful, or a standard OLE error value otherwise.

Remarks

SHAutoComplete will work on any system edit control, including the edit and controls that contain edit controls such as comboboxex controls. To get a handle to the edit control that is embedded in a comboboxex control send it a **CBEM_GETEDITCONTROL**.

An application must have invoked either **CoInitialize** or **OleInitialize** prior to calling this function. **CoUninitialize** or **OleUninitialize** cannot be called until the edit box has finished processing the **WM_DESTROY** message for *hwndEdit*.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).

Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHCreateStreamOnFile

Takes a file name, opens the file, and returns an **IStream** interface that can be used to read from and write to the file.

```
HRESULT SHCreateStreamOnFile(
  LPCTSTR pszFile,
  DWORD grfMode,
  IStream **ppstm
);
```

Parameters

pszFile
[in] Pointer to a NULL-terminated string with the file name.

grfMode
[in] **STGM** value that is used to specify access modes and how the object that exposes the **IStream** interface is created and deleted.

ppstm
[out] Pointer to an **IStream** interface for the file.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHCreateThread

Creates a thread.

```
BOOL SHCreateThread(
  LPTHREAD_START_ROUTINE pfnThreadProc,
  VOID *pData,
  DWORD dwFlags,
  LPTHREAD_START_ROUTINE pfnCallback
);
```

Parameters

pfnThreadProc

Pointer to an application-defined function of the **LPTHREAD_START_ROUTINE** type. If a new thread was successfully created, this function will be called in the context of that thread. **SHCreateThread** does not wait for this function to complete before returning to its caller. The return value of this function will be the exit code of the thread.

pData

Pointer to an application-defined data structure containing initialization data. It is passed to the function pointed to by *pfnThreadProc* and, optionally, *pfnCallback*.

dwFlags

Flags that control the behavior of the function. This parameter can be a combination of the following flags.

Flag	Description
CTF_COINIT	Initialize COM for the created thread before calling either the optional function pointed to by *pfnCallback* or the function pointed to by *pfnThreadProc*. This flag is useful when COM needs to be initialized for a thread. COM will automatically be uninitialized as well.
CTF_INSIST	If the attempt to create the thread with **CreateThread** fails, setting this flag will cause the function pointed to by *pfnThreadProc* to be called synchronously from the calling thread. This flag cannot be used if *pfnCallback* has a non-NULL value.
CTF_PROCESS_REF	Hold a reference to the Windows Explorer process for the duration of the call to the function pointed to by *pfnThreadProc*. This flag is useful for shell extension handlers, which might need to keep the Windows Explorer process from closing prematurely. Examples of where this action would be useful include tasks such as doing work on a background thread or copying files. For further information, see **SHGetInstanceExplorer**.
CTF_THREAD_REF	Hold a reference to the creating thread for the duration of the call to the function pointed to by *pfnThreadProc*. This reference must have been set with **SHSetThreadRef**.

pfnCallback

Pointer to an optional application-defined function of the **LPTHREAD_START_ROUTINE** type. This function is called in the context of the created thread before the function pointed to by *pfnThreadProc* is called. It will also receive *pData* as its argument. **SHCreateThread** will wait for the function pointed to by *pfnCallback* to return before returning to its caller. The return value of the function pointed to by *pfnCallback* is ignored.

Return Values

Returns TRUE if the thread is successfully created, or FALSE otherwise.

Remarks

The function pointed to by *pfnThreadProc* and *pfnCallback* must take the form:

```
DWORD WINAPI ThreadProc(LPVOID pData)
{
...
}
```

The function name is arbitrary. The *pData* parameter points to an application-defined data structure with initialization information.

Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

See Also

CreateThread, **CreateProcess**

SHGetThreadRef

Retrieves the per-thread object reference set by **SHSetThreadRef**.

```
HRESULT SHGetThreadRef(
  IUnknown **ppunk
);
```

Parameters

ppunk

Address of a pointer to an **IUnknown** interface. If successful, this parameter will hold the object's **IUnknown** pointer on return. Your application is responsible for freeing the pointer when it is finished.

Return Values

Returns S_OK if the object reference exists, or E_NOINTERFACE otherwise.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

+ See Also

SHCreateThread

SHOpenRegStream

Opens a registry value and supplies an **IStream** interface that can be used to read from or write to the value.

```
struct IStream* SHOpenRegStream(
  HKEY hkey,
  LPCTSTR pszSubkey,
  LPCTSTR pszValue,
  DWORD grfMode
);
```

Parameters

hkey

Handle to the key that is currently open.

pszSubkey

Address of a null-terminated string that specifies the name of the subkey.

pszValue

Address of the value to be accessed.

grfMode

Type of access for the stream. This can be one of the following values:

STGM_READ	Open the stream for reading.
STGM_WRITE	Open the stream for writing.
STGM_READWRITE	Open the stream for reading and writing.

Return Values

Returns the address of an **IStream** interface if successful, or NULL otherwise.

Remarks

The calling application is responsible for calling this interface's **Release** method when the **IStream** object is no longer needed.

! Requirements

Version 4.71 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 4.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHOpenRegStream2

Opens a registry value and returns an **IStream** interface that can be used to read from or write to the value. It supersedes **SHOpenRegStream**.

```
IStream* SHOpenRegStream2(
  HKEY hkey,
  LPCSTR pszSubkey,
  LPCSTR pszValue,
  DWORD grfMode
);
```

Parameters

hkey

[in] Handle to a key that is currently open, or a registry root key.

pszSubkey

[in] Pointer to a NULL-terminated string that contains the name of the subkey.

pszValue

[in] Pointer to a NULL-terminated string with the name of the value that holds the unexpanded path string.

grfMode

[in] Type of access for the stream. It can have one of the following values.

Value	Description
STGM_READ	Open the stream for reading.
STGM_WRITE	Open the stream for writing.
STGM_READWRITE	Open the stream for reading and writing.

Return Values

Returns the address of an **IStream** interface if successful, or NULL otherwise.

Remarks

The calling application is responsible for calling this interface's **Release** method when the **IStream** object is no longer needed.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

SHSetThreadRef

Stores a per-thread reference to a COM object.

```
HRESULT SHSetThreadRef(
  IUnknown *punk
);
```

Parameters

punk

Pointer to the **IUnknown** interface of the object to which you want to store a reference.

Return Values

Returns S_OK if successful, or an OLE error value otherwise.

Remarks

Use **SHGetThreadRef** to retrieve the **IUnknown** pointer.

! Requirements

Version 5.00 and later of Shlwapi.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in shlwapi.h.
Import Library: shlwapi.lib.

+ See Also

SHCreateThread

CHAPTER 12

Shell Messages and Notifications

Shell Messages and Notifications

ABM_ACTIVATE

Notifies the system that an appbar has been activated. An appbar should call this message in response to the **WM_ACTIVATE** message.

```
SHAppBarMessage(ABM_ACTIVATE, pabd);
```

Parameters

pabd

Address of an **APPBARDATA** structure that identifies the appbar to activate. You must specify the **cbSize** and **hWnd** members when sending this message; all other members are ignored.

Return Values

Always returns TRUE.

Remarks

This message is ignored if the **hWnd** member of the structure pointed to by *pabd* identifies an autohide appbar. The system automatically sets the z-order for autohide appbars.

! Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.

ABM_GETAUTOHIDEBAR

Retrieves the handle to the autohide appbar associated with an edge of the screen.

```
hwndAutoHide = (HWND) SHAppBarMessage(ABM_GETAUTOHIDEBAR, pabd);
```

Parameters

pabd

Address of an **APPBARDATA** structure that specifies the screen edge. You must specify the **cbSize**, **hWnd**, and **uEdge** members when sending this message; all other members are ignored.

Return Values

Returns the handle to the autohide appbar. The return value is NULL if an error occurs or if no autohide appbar is associated with the given edge.

! Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.

ABM_GETSTATE

Retrieves the autohide and always-on-top states of the Windows taskbar.

```
fuState = (UINT) SHAppBarMessage(ABM_GETSTATE, pabd);
```

Parameters

pabd

Address of an **APPBARDATA** structure. You must specify the **cbSize** and **hWnd** members when sending this message; all other members are ignored.

Return Values

Returns zero if the taskbar is not in the autohide or always-on-top state. Otherwise, the return value is one or both of the following:

ABS_ALWAYSONTOP	The taskbar is in the always-on-top state.
ABS_AUTOHIDE	The taskbar is in the autohide state.

! Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.

ABM_GETTASKBARPOS

Retrieves the bounding rectangle of the Windows taskbar.

```
fResult = (BOOL) SHAppBarMessage(ABM_GETTASKBARPOS, pabd);
```

Parameters

pabd

Address of an **APPBARDATA** structure whose **rc** member receives the bounding rectangle, in screen coordinates, of the taskbar. You must specify the **cbSize** and **hWnd** when sending this message; all other members are ignored.

Return Values

Returns TRUE if successful, or FALSE otherwise.

! Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.

ABM_NEW

Registers a new appbar and specifies the message identifier that the system should use to send it notification messages. An appbar should send this message before sending any other appbar messages.

```
fRegistered = (BOOL) SHAppBarMessage(ABM_NEW, pabd);
```

Parameters

pabd

Address of an **APPBARDATA** structure that contains the new appbar's window handle and message identifier. You must specify the **cbSize**, **hWnd**, and **uCallbackMessage** members when sending this message; all other members are ignored.

Return Values

Returns TRUE if successful, or FALSE if an error occurs or if the appbar is already registered.

! Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.

Windows CE: Unsupported.
Header: Declared in shellapi.h.

ABM_QUERYPOS

Requests a size and screen position for an appbar. When the request is made, the message proposes a screen edge and a bounding rectangle for the appbar. The system adjusts the bounding rectangle so that the appbar does not interfere with the Windows taskbar or any other appbars.

```
SHAppBarMessage(ABM_QUERYPOS, pabd);
```

Parameters

pabd

Address of an **APPBARDATA** structure. The **uEdge** member specifies a screen edge, and the **rc** member contains the proposed bounding rectangle. When the **SHAppBarMessage** function returns, **rc** contains the approved bounding rectangle. You must specify the **cbSize**, **hWnd**, **uEdge**, and **rc** members when sending this message; all other members are ignored.

Return Values

Always returns TRUE.

Remarks

An appbar should send this message before sending the **ABM_SETPOS** message.

! Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.

ABM_REMOVE

Unregisters an appbar by removing it from the system's internal list. The system no longer sends notification messages to the appbar or prevents other applications from using the screen area occupied by the appbar.

```
SHAppBarMessage(ABM_REMOVE, pabd);
```

Parameters

pabd

Address of an **APPBARDATA** structure that contains the handle to the appbar to unregister. You must specify the **cbSize** and **hWnd** members when sending this message; all other members are ignored.

Return Values

Always returns TRUE.

Remarks

This message causes the system to send the **ABN_POSCHANGED** notification message to all appbars.

! Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.

ABM_SETAUTOHIDEBAR

Registers or unregisters an autohide appbar for an edge of the screen.

```
fSuccess = (BOOL) SHAppBarMessage(ABM_SETAUTOHIDEBAR, pabd);
```

Parameters

pabd

Address of an **APPBARDATA** structure. The **uEdge** member specifies the screen edge. The **lParam** parameter is set to TRUE to register the appbar or FALSE to unregister it. You must specify the **cbSize**, **hWnd**, **uEdge**, and **lParam** members when sending this message; all other members are ignored.

Return Values

Returns TRUE if successful, or FALSE if an error occurs or if an autohide appbar is already registered for the given edge.

Remarks

The system allows only one autohide appbar for each edge of the screen. This is determined when the member **uEdge** of the **APPBARDATA** structure is set.

! Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.

ABM_SETPOS

Sets the size and screen position of an appbar. The message specifies a screen edge and the bounding rectangle for the appbar. The system may adjust the bounding rectangle so that the appbar does not interfere with the Windows taskbar or any other appbars.

```
SHAppBarMessage(ABM_SETPOS, pabd);
```

Parameters

pabd

Address of an **APPBARDATA** structure. The **uEdge** member specifies a screen edge, and the **rc** member contains the bounding rectangle. When the **SHAppBarMessage** function returns, **rc** contains the approved bounding rectangle. You must specify the **cbSize**, **hWnd**, **uEdge**, and **rc** members when sending this message; all other members are ignored.

Return Values

Always returns TRUE.

Remarks

This message causes the system to send the **ABN_POSCHANGED** notification message to all appbars.

! Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.

ABM_WINDOWPOSCHANGED

Notifies the system when an appbar's position has changed. An appbar should call this message in response to the **WM_WINDOWPOSCHANGED** message.

```
SHAppBarMessage(ABM_WINDOWPOSCHANGED, pabd);
```

Parameters

pabd

Address of an **APPBARDATA** structure that identifies the appbar to activate. You must specify the **cbSize** and **hWnd** members when sending this message; all other members are ignored.

Return Values

Always returns TRUE.

Remarks

This message is ignored if the **hWnd** member of the structure pointed to by *pabd* identifies an autohide appbar.

! Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.

ABN_FULLSCREENAPP

Notifies an appbar when a full-screen application is opening or closing. This notification is sent in the form of an application-defined message that is set by the **ABM_NEW** message.

```
ABN_FULLSCREENAPP
    fOpen = (BOOL) lParam;
```

Parameters

fOpen

Flag specifying whether a full-screen application is opening or closing. This parameter is TRUE if the application is opening or FALSE if it is closing.

Return Values

No return value.

Remarks

When a full-screen application is opening, an appbar must drop to the bottom of the z-order. When it is closing, the appbar should restore its z-order position.

! Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.

ABN_POSCHANGED

Notifies an appbar when an event has occurred that may affect the appbar's size and position. Events include changes in the taskbar's size, position, and visibility state, as well as the addition, removal, or resizing of another appbar on the same side of the screen.

```
ABN_POSCHANGED
```

Return Values

No return value.

Remarks

An appbar should respond to this notification message by sending the **ABM_QUERYPOS** and **ABM_SETPOS** messages. If its position has changed, the appbar should call the **MoveWindow** function to move itself to the new position.

! Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.

ABN_STATECHANGE

Notifies an appbar that the taskbar's autohide or always-on-top state has changed—that is, the user has selected or cleared the "Always on top" or "Auto hide" check box on the taskbar's property sheet.

```
ABN_STATECHANGE
```

Return Values

No return value.

Remarks

An appbar can use this notification message to set its state to conform to that of the taskbar, if desired.

! Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.

ABN_WINDOWARRANGE

Notifies an appbar that the user has selected the Cascade, Tile Horizontally, or Tile Vertically command from the taskbar's context menu.

```
ABN_WINDOWARRANGE
    fBeginning = (BOOL) lParam;
```

Parameters

fBeginning

Flag specifying whether the cascade or tile operation is beginning. This parameter is TRUE if the operation is beginning and the windows have not yet been moved. It is FALSE if the operation has completed.

Return Value

No return value.

Remarks

The system sends this notification message twice—first with *lParam* set to TRUE and then with *lParam* set to FALSE. The first notification is sent before the windows are cascaded or tiled, and the second is sent after the cascade or tile operation has occurred.

! Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in shellapi.h.

CPL_DBLCLK

Sent to the CPlApplet function of a Control Panel application when the user double-clicks the icon of a dialog box supported by the application.

```
CPL_DBLCLK
    uAppNum = (UINT) lParam1;
    lData = (LONG) lParam2;
```

Parameters

uAppNum

Dialog box number. This number must be in the range zero through one less than the value returned in response to the **CPL_GETCOUNT** message (CPL_GETCOUNT –1).

lData

Value that the Control Panel application loaded into the **lData** member of the **CPLINFO** or **NEWCPLINFO** structure for the dialog box. The application loads the **lData** member in response to the **CPL_INQUIRE** or **CPL_NEWINQUIRE** message.

Return Values

If the **CPlApplet** function processes this message successfully, the return value is zero; otherwise, it is nonzero.

Remarks

In response to this message, a Control Panel application must display the corresponding dialog box.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in cpl.h.

See Also

CPL_SELECT

CPL_EXIT

Sent once to the **CPlApplet** function of a Control Panel application before the DLL containing the Control Panel application is released.

```
CPL_EXIT
```

Return Values

If the **CPlApplet** function processes this message successfully, it should return zero.

Remarks

This message is sent after the last **CPL_STOP** message is sent.

In response to this message, a Control Panel application must free any memory that it has allocated and perform global-level cleanup.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in cpl.h.

See Also

FreeLibrary

CPL_GETCOUNT

Sent to the **CPlApplet** function of a Control Panel application to retrieve the number of dialog boxes supported by the application.

```
CPL_GETCOUNT
```

Return Values

The **CPlApplet** function returns the number of dialog boxes that the Control Panel application supports.

Remarks

This message is sent immediately after the **CPL_INIT** message.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in cpl.h.

CPL_INIT

Sent to the **CPlApplet** function of a Control Panel application to prompt it to perform global initialization, especially memory allocation.

```
CPL_INIT
```

Return Values

If initialization succeeds, the **CPlApplet** function should return nonzero. Otherwise, it should return zero.

If **CPlApplet** returns zero, the controlling application ends communication and releases the DLL containing the Control Panel application.

Remarks

Because this is the only way a Control Panel application can signal an error condition, the application should allocate memory in response to this message.

This message is sent immediately after the DLL containing the application is loaded.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in cpl.h.

See Also

FreeLibrary

CPL_INQUIRE

Sent to the CPlApplet function of a Control Panel application to request information about a dialog box that the application supports.

```
CPL_INQUIRE
    uAppNum = (UINT) lParam1;
    lpcpli = (LPCPLINFO) lParam2;
```

Parameters

uAppNum

Dialog box number. This number must be in the range zero through one less than the value returned in response to the **CPL_GETCOUNT** message (CPL_GETCOUNT –1).

lpcpli

Address of a **CPLINFO** structure. The application must fill this structure with resource identifiers for the icon, short name, description, and any user-defined value associated with the dialog box.

Return Values

If the **CPlApplet** function processes this message successfully, it should return zero.

Remarks

The Control Panel sends the **CPL_INQUIRE** message once for each dialog box supported by your application. The Control Panel also sends a **CPL_NEWINQUIRE** message for each dialog box. These messages are sent immediately after the CPL_GETCOUNT message. However, the system does not guarantee the order in which the CPL_INQUIRE and CPL_NEWINQUIRE messages are sent.

You can perform initialization for the dialog box when you receive CPL_INQUIRE. If you must allocate memory, do so in response to the **CPL_INIT** message.

On Windows 95 and Windows NT version 4.0, the system caches the information returned in the **CPLINFO** structure used by CPL_INQUIRE. This provides significantly better performance because the system only needs to load your application the first time the Control Panel starts up. On the other hand, the CPL_NEWINQUIRE message returns information in a form that the system cannot cache. For this reason, most **CPlApplet** functions should process CPL_INQUIRE and ignore CPL_NEWINQUIRE.

The only applications that should use CPL_NEWINQUIRE are those that need to change their icon or display strings based on the state of the computer. In this case, your CPL_INQUIRE handler should specify the CPL_DYNAMIC_RES value for the **idIcon**, **idName**, or **idInfo** members of the **CPLINFO** structure, rather than specifying a valid resource identifier. This causes the Control Panel to send the CPL_NEWINQUIRE message each time it needs the icon and display strings, allowing you to specify information based on the current state of the computer. Of course, this is significantly slower than using cached information.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in cpl.h.

CPL_NEWINQUIRE

Sent to the **CPlApplet** function of a Control Panel application to request information about a dialog box that the application supports.

```
CPL_NEWINQUIRE
    uAppNum = (UINT) lParam1;
    lpncpli = (LPNEWCPLINFO) lParam2;
```

Parameters

uAppNum

Dialog box number. This number must be in the range zero through one less than the value returned in response to the **CPL_GETCOUNT** message (CPL_GETCOUNT −1).

lpncpli

Address of a **NEWCPLINFO** structure. The Control Panel application should fill this structure with information about the dialog box.

Return Values

If the **CPlApplet** function processes this message successfully, it should return zero.

Remarks

For better performance, most applications should ignore CPL_NEWINQUIRE and process the **CPL_INQUIRE** message instead.

The Control Panel sends the CPL_NEWINQUIRE message once for each dialog box supported by your application. The Control Panel also sends a CPL_INQUIRE message for each dialog box. These messages are sent immediately after the CPL_GETCOUNT message. However, the system does not guarantee the order in which the CPL_INQUIRE and CPL_NEWINQUIRE messages are sent.

You can perform initialization for the dialog box when you receive CPL_INQUIRE. If you must allocate memory, do so in response to the **CPL_INIT** message.

The CPL_NEWINQUIRE message was introduced in Windows version 3.1 as a replacement for CPL_INQUIRE. However, CPL_INQUIRE is the preferred message for Windows 95 and Windows NT version 4.0. This is because CPL_NEWINQUIRE returns information in a form that the system cannot cache. Consequently, applications that process CPL_NEWINQUIRE must be loaded each time the Control Panel needs the information, resulting in a significant reduction in performance.

The only applications that should use CPL_NEWINQUIRE are those that need to change their icon or display strings based on the state of the computer. In this case, your CPL_INQUIRE handler should specify the CPL_DYNAMIC_RES value for the **idIcon**, **idName**, or **idInfo** members of the **CPLINFO** structure, rather than specifying a valid resource identifier. This causes the Control Panel to send the CPL_NEWINQUIRE message each time it needs the icon and display strings, allowing you to specify information based on the current state of the computer. Of course, this is significantly slower than using cached information.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in cpl.h.

CPL_STARTWPARMS

Sent to notify **CPlApplet** that the user has chosen the icon associated with a given dialog box. **CPlApplet** should display the corresponding dialog box and carry out any user-specified tasks.

```
CPL_STARTWPARMS
    uAppNum = (UINT) lParam1;
    lpszExtra = (LPCTSTR) lParam2;
```

Parameters

uAppNum

Dialog box number. This number must be in the range zero through one less than the value returned in response to the **CPL_GETCOUNT** message (CPL_GETCOUNT –1).

lpszExtra

String with additional directions for execution.

Return Values

Returns TRUE if the message was handled, or FALSE otherwise.

Remarks

CPL_STARTWPARMS is similar to **CPL_DBLCLK** but allows you to pass a string to **CPlApplet** instead of an integer. You can use this string as a flexible way to provide detailed directions for execution.

Requirements

Version 5.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows 2000.
Windows CE: Unsupported.
Header: Declared in cpl.h.

CPL_STOP

Sent to the **CPlApplet** function of a Control Panel application when the controlling application of the Control Panel closes. The controlling application sends the message once for each dialog box that the application supports.

```
CPL_STOP
    uAppNum = (UINT) lParam1;
    lData = (LONG) lParam2;
```

Parameters

uAppNum
: Dialog box number.

lData
: Value that the Control Panel application loaded into the **lData** member of the **CPLINFO** or **NEWCPLINFO** structure for the dialog box. The application loads the **lData** member in response to the **CPL_INQUIRE** or **CPL_NEWINQUIRE** message.

Return Values

If the **CPlApplet** function processes this message successfully, it should return zero.

Remarks

In response to this message, a Control Panel application must perform cleanup for the given dialog box.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in cpl.h.

+ See Also

CPL_EXIT, **CPL_GETCOUNT**

FM_GETDRIVEINFO

Sent by a File Manager extension to retrieve drive information from the active File Manager window.

```
FM_GETDRIVEINFO
    wParam = 0;
    lParam = (LPARAM) (LPFMS_GETDRIVEINFO) lpfmsgdi;
```

Parameters

lpfmsgdi

Address of an **FMS_GETDRIVEINFO** structure that receives drive information.

Return Values

Returns nonzero.

Remarks

If 0xFFFFFFFF is returned in the **dwTotalSpace** or **dwFreeSpace** member of the **FMS_GETDRIVEINFO** structure, the extension library must compute the value or values.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

+ See Also

FMExtensionProc

FM_GETFILESEL

Sent by a File Manager extension to retrieve information about a selected file from the active File Manager window (either the directory window or the Search Results window).

```
FM_GETFILESEL
    wParam = (WPARAM) index;
    lParam = (LPARAM) (LPFMS_GETFILESEL) lpfmsgfs;
```

Parameters

index

Zero-based index of the selected file to retrieve.

lpfmsgfs

Address of an **FMS_GETFILESEL** structure that receives information about the selection.

Return Value

Returns the zero-based index of the selected file that was retrieved.

Remarks

An extension can use the **FM_GETSELCOUNT** message to get the count of selected files.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

See Also

FMExtensionProc, FM_GETFILESELLFN, FM_GETSELCOUNTLFN

FM_GETFILESELLFN

Sent by a File Manager extension to retrieve information about a selected file from the active File Manager window (either the directory window or the Search Results window). The selected file can have a long file name.

```
FM_GETFILESELLFN
    wParam = (WPARAM) index;
    lParam = (LPARAM) (LPFMS_GETFILESEL) lpfmsgfs;
```

Parameters

index
Zero-based index of the selected file to retrieve.

lpfmsgfs
Address of an **FMS_GETFILESEL** structure that receives information about the selection.

Return Values

Returns the zero-based index of the selected file that was retrieved.

Remarks

Only extensions that support long file names (for example, network-aware extensions) should use this message.

An extension can use the **FM_GETSELCOUNTLFN** message to get the count of selected files.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

See Also

FMExtensionProc, **FM_GETFILESEL**, **FM_GETSELCOUNT**

FM_GETFOCUS

Sent by a File Manager extension to retrieve the type of File Manager window that has the input focus.

```
FM_GETFOCUS
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the type of File Manager window that has the input focus. It can be one of the following values:

FMFOCUS_DIR	Directory portion of a directory window.
FMFOCUS_TREE	Tree portion of a directory window.
FMFOCUS_DRIVES	Drive bar of a directory window.
FMFOCUS_SEARCH	Search Results window.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

FM_GETSELCOUNT

Sent by a File Manager extension to retrieve a count of the selected files in the active File Manager window (either the directory window or the Search Results window).

```
FM_GETSELCOUNT
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the number of selected files.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

+ See Also

FM_GETFILESEL, **FM_GETFILESELLFN**, **FM_GETSELCOUNTLFN**

FM_GETSELCOUNTLFN

Sent by a File Manager extension to retrieve the number of selected files in the active File Manager window (either the directory window or the Search Results window). The count includes files that have long file names.

```
FM_GETSELCOUNTLFN
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the number of selected files.

Remarks

Only extensions that support long file names (for example, network-aware extensions) should use this message.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

+ See Also

FM_GETFILESEL, **FM_GETFILESELLFN**, **FM_GETSELCOUNT**

FM_REFRESH_WINDOWS

Sent by a File Manager extension to cause File Manager to repaint either its active window or all of its windows.

```
FM_REFRESH_WINDOWS
    wParam = (WPARAM) (BOOL) fRepaint;
    lParam = 0;
```

Parameters

fRepaint

Value that indicates whether File Manager repaints its active window or all of its windows. If this parameter is TRUE, File Manager repaints all of its windows. Otherwise, File Manager repaints only its active window.

Return Values

No return value.

Remarks

File system changes caused by an extension are automatically detected by File Manager. An extension should use this message only in situations where drive connections are made or canceled.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

See Also

FMExtensionProc

FM_RELOAD_EXTENSIONS

Sent by a File Manager extension (or another application) to cause File Manager to reload all extension DLLs listed in the [AddOns] section of the Winfile.ini file.

```
FM_RELOAD_EXTENSIONS
    wParam = 0;
    lParam = 0;
```

Return Values

No return value.

Remarks

Other applications can use the **PostMessage** function to send this message to File Manager. To obtain the appropriate File Manager window handle, an application can specify "WFS_Frame" as the *lpszClassName* parameter in a call to the **FindWindow** function.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

See Also

FMExtensionProc

FMEVENT_HELPMENUITEM

Sent to a File Manager extension DLL procedure when the user presses F1 on a menu or toolbar command item. The extension should call **WinHelp**, with that function's *hwnd* parameter set to the value of the extension's *hwnd* parameter.

```
FMEVENT_HELPMENUITEM
    uItem = (UINT) lParam;
```

Parameters

uItem

Value that identifies the menu or toolbar command item for which Help is sought. The extension procedure uses this value to determine how best to call **WinHelp.**

Return Values

An extension DLL procedure should return zero if it processes this message.

Requirements

Windows NT/2000: Requires Windows NT 3.5 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

See Also

FMExtensionProc, **FMEVENT_HELPSTRING**

FMEVENT_HELPSTRING

Sent to a File Manager extension DLL procedure when File Manager wants a Help string for a menu or toolbar command item.

```
FMEVENT_HELPSTRING
    lpfmshs = (LPFMS_HELPSTRING) lParam;
```

Parameters

lpfmshs

Address of an **FMS_HELPSTRING** structure that communicates command item Help string data.

The **FMS_HELPSTRING** structure identifies the command item for which a Help string is wanted, along with a handle to its menu. An application then writes the appropriate Help string to the **FMS_HELPSTRING** structure's **szHelp** member.

Return Values

An extension DLL procedure should return zero if it processes this message.

Requirements

Windows NT/2000: Requires Windows NT 3.5 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

See Also

FMExtensionProc, **FMEVENT_HELPMENUITEM**

FMEVENT_INITMENU

Sent to an extension DLL when the user selects the menu for the extension from the File Manager menu bar. The extension can use this notification to initialize menu items.

```
FMEVENT_INITMENU
    hmenu = (HMENU) lParam;
```

Parameters

hmenu

Handle to the File Manager menu bar.

Return Values

An extension DLL should return zero if it processes this message.

Remarks

An extension DLL receives this message only when the user selects the top-level menu. If the extension contains submenus, it must initialize them at the same time it initializes the top-level menu.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

See Also

FMExtensionProc

FMEVENT_LOAD

Sent to an extension DLL when File Manager is loading the DLL.

```
FMEVENT_LOAD
    lpfmsld = (LPFMS_LOAD) lParam;
```

Parameters

lpfmsld

Address of an **FMS_LOAD** structure that specifies the menu item delta value.

Return Values

An extension DLL must return TRUE to continue loading the DLL. If the DLL returns FALSE, File Manager calls the **FreeLibrary** function and ends any communication with the extension DLL.

Remarks

An application should fill the **dwSize**, **szMenuName**, and **hMenu** members in the **FMS_LOAD** structure. It should also save the value of the **wMenuDelta** member and use it to identify menu items when modifying the menu.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

See Also

FMExtensionProc

FMEVENT_SELCHANGE

Sent to an extension DLL when the user selects a file name in the File Manager directory window or Search Results window.

```
FMEVENT_SELCHANGE
```

Return Values

An extension DLL should return zero if it processes this message.

Remarks

Changes in the tree portion of the directory window do not produce this message.

Because the user can change the selection many times, the extension DLL must return promptly after processing this message to avoid slowing the selection process for the user.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

+ See Also

FMExtensionProc

FMEVENT_TOOLBARLOAD

Sent to an extension DLL when File Manager is loading its toolbar. This message allows an extension DLL to add a button to the File Manager toolbar.

```
FMEVENT_TOOLBARLOAD
    lpfmstbl = (LPFMS_TOOLBARLOAD) lParam;
```

Parameters

lpfmstbl

Address of an **FMS_TOOLBARLOAD** structure. If the extension DLL adds a button to the toolbar in File Manager, the DLL should fill the structure with information about the button.

Return Values

An extension DLL must return TRUE to add the button to the toolbar. If the DLL returns FALSE, File Manager does not add the button.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

+ See Also

FMExtensionProc

FMEVENT_UNLOAD

Sent to an extension DLL when File Manager is unloading the DLL.

```
FMEVENT_UNLOAD
```

Return Values

An extension DLL should return zero if it processes this message.

Remarks

The *hwnd* and *hMenu* values passed with the **FMEVENT_LOAD** and **FMEVENT_INITMENU** messages may not be valid at the time this message is sent.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

+ See Also

FMExtensionProc

FMEVENT_USER_REFRESH

Sent to an extension DLL when the user chooses the Refresh command from the View menu in File Manager. The extension can use this notification to update its menu.

```
FMEVENT_USER_REFRESH
```

Return Values

An extension DLL should return zero if it processes this message.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows CE: Unsupported.
Header: Declared in wfext.h.

See Also

FMExtensionProc

WM_CPL_LAUNCH

Sent by an application to the Windows Control Panel to request that a Control Panel application be started.

```
WM_CPL_LAUNCH
    hwnd = (HWND) wParam;
    lpszAppName = (LPSTR) lParam;
```

Parameters

hwnd

Handle to the window that is sending the request. The **WM_CPL_LAUNCHED** message is returned to this window.

lpszAppName

Address of a string containing the name of the Control Panel application to open.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in cpl.h.

WM_CPL_LAUNCHED

Sent when a Control Panel application that was started by the **WM_CPL_LAUNCH** message has closed. The WM_CPL_LAUNCHED message is sent to the window identified by the *wParam* parameter of the WM_CPL_LAUNCH message that started the application.

```
WM_CPL_LAUNCHED
    fAppStarted = (BOOL) wParam;
```

Parameters

fAppStarted

Value that indicates whether the application was started. If the application was started, this parameter is TRUE; otherwise, it is FALSE.

Return Values

The return value is ignored.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in cpl.h.

WM_DROPFILES

Sent when the user drops a file on the window of an application that has registered itself as a recipient of dropped files.

```
WM_DROPFILES
    hDrop = (HDROP) wParam;
```

Parameters

hDrop

Handle to an internal structure describing the dropped files. Pass this handle **DragFinish**, **DragQueryFile**, or **DragQueryPoint** to retrieve information about the dropped files.

Return Values

An application should return zero if it processes this message.

Remarks

The HDROP handle is declared in Shellapi.h. You must include this header in your build to use WM_DROPFILES. See *Transferring Shell Data Using Drag-Drop or the Clipboard* for further discussion of how to use drag-drop to transfer shell data.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in winuser.h.

See Also

DragAcceptFiles

WM_HELP

Indicates that the user pressed the F1 key. If a menu is active when F1 is pressed, WM_HELP is sent to the window associated with the menu; otherwise, WM_HELP is sent to the window that has the keyboard focus. If no window has the keyboard focus, WM_HELP is sent to the currently active window.

```
WM_HELP
    lphi = (LPHELPINFO) lParam;
```

Parameters

lphi

Address of a **HELPINFO** structure that contains information about the menu item, control, dialog box, or window for which Help is requested.

Return Values

Returns TRUE.

Remarks

The **DefWindowProc** function passes WM_HELP to the parent window of a child window or to the owner of a top-level window.

Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winuser.h.

WM_TCARD

Sent to an application that has initiated a training card with Windows Help. The message informs the application when the user clicks an authorable button. An application initiates a training card by specifying the HELP_TCARD command in a call to the **WinHelp** function.

```
WM_TCARD
    idAction = wParam;
    dwActionData = lParam;
```

Parameters

idAction

Value that indicates the action the user has taken. This can be one of these values:

IDABORT	The user clicked an authorable Abort button.
HELP_TCARD_DATA	The user clicked an authorable button. The *lParam* parameter contains a long integer specified by the Help author.
HELP_TCARD_NEXT	The user clicked an authorable Next button.
HELP_TCARD_OTHER_CALLER	Another application has requested training cards.
IDCANCEL	The user clicked an authorable Cancel button.
IDCLOSE	The user closed the training card.
IDHELP	The user clicked an authorable Windows Help button.
IDIGNORE	The user clicked an authorable Ignore button.
IDNO	The user clicked an authorable No button.
IDOK	The user clicked an authorable OK button.
IDRETRY	The user clicked an authorable Retry button.
IDYES	The user clicked an authorable Yes button.

dwActionData

If *idAction* specifies HELP_TCARD_DATA, this parameter is a long integer specified by the Help author. Otherwise, this parameter is zero.

Return Values

The return value is ignored; use zero.

Requirements

Version 4.00 and later of Shell32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in winuser.h.

APPENDIX A

Index A: Elements Grouped by Technology

The indexes in Part 3 make finding information in the Win32 Library volumes as easy as possible. Rather than cluttering the Table of Contents with information about individual programmatic elements (and thereby making the TOC uselessly jumbled), I've created these indexes and put them in the back of each volume. With these indexes, you'll be able to locate the programmatic element you're interested in—and see where it fits within the volumes and their technologies—quickly and easily.

Also, to keep you informed and up-to-date about Microsoft technologies, I've created a live Web-based document that maps Microsoft technologies to the locations where you can get more information about them. This link gets you to the live index of technologies: *www.iseminger.com/winprs/technologies*

As always, send me feedback if you can think of ways to improve this section. I can't guarantee a reply, but I'll read it, and if others can benefit, I'll incorporate the idea into future volumes.

Color Palette Functions 707
ColorAdjustLuma
ColorHLSToRGB
ColorRGBToHLS
SHCreateShellPalette

Miscellaneous 710
DllInstall
HashData
SHAutoComplete
SHCreateStreamOnFile
SHCreateThread
SHGetThreadRef
SHOpenRegStream
SHOpenRegStream2
SHSetThreadRef

Path Functions 610
PathAddBackslash
PathAddExtension
PathAppend
PathBuildRoot
PathCanonicalize
PathCombine
PathCommonPrefix
PathCompactPath
PathCompactPathEx
PathCreateFromUrl
PathFileExists
PathFindExtension
PathFindFileName
PathFindNextComponent
PathFindOnPath
PathFindSuffixArray
PathGetArgs
PathGetCharType
PathGetDriveNumber
PathIsContentType
PathIsDirectory
PathIsDirectoryEmpty
PathIsFileSpec
PathIsHTMLFile
PathIsLFNFileSpec
PathIsNetworkPath
PathIsPrefix
PathIsRelative
PathIsRoot
PathIsSameRoot
PathIsSystemFolder
PathIsUNC
PathIsUNCServer
PathIsUNCServerShare
PathIsURL
PathMakePretty
PathMakeSystemFolder
PathMatchSpec
PathParseIconLocation

Path Functions *(continued)*

- PathQuoteSpaces
- PathRelativePathTo
- PathRemoveArgs
- PathRemoveBackslash
- PathRemoveBlanks
- PathRemoveExtension
- PathRemoveFileSpec
- PathRenameExtension
- PathSearchAndQualify
- PathSetDlgItemPath
- PathSkipRoot
- PathStripPath
- PathStripToRoot
- PathUndecorate
- PathUnExpandEnvStrings
- PathUnmakeSystemFolder
- PathUnquoteSpaces
- UrlApplyScheme
- UrlCanonicalize
- UrlCombine
- UrlCompare
- UrlCreateFromPath
- UrlEscape
- UrlEscapeSpaces
- UrlGetLocation
- UrlGetPart
- UrlHash
- UrlIs
- UrlIsFileUrl
- UrlIsNoHistory
- UrlIsOpaque
- UrlUnEscape
- UrlUnEscapeInPlace

Registry Data Types 668

- REGSAM

Registry Functions 670

- AssocCreate
- AssocQueryKey
- AssocQueryString
- AssocQueryStringByKey
- SHCopyKey
- SHDeleteEmptyKey
- SHDeleteKey
- SHDeleteValue
- SHEnumKeyEx
- SHEnumValue
- SHGetValue
- SHQueryInfoKey
- SHQueryValueEx
- SHRegCloseUSKey
- SHRegCreateUSKey
- SHRegDeleteEmptyUSKey
- SHRegDeleteUSValue
- SHRegDuplicateHKey
- SHRegEnumUSKey
- SHRegEnumUSValue
- SHRegGetBoolUSValue
- SHRegGetPath
- SHRegGetUSValue
- SHRegOpenUSKey
- SHRegQueryInfoUSKey
- SHRegQueryUSValue
- SHRegSetPath
- SHRegSetUSValue
- SHRegWriteUSValue
- SHSetValue
- SHREGDEL_FLAGS
- SHREGENUM_FLAGS

Shell Callback Functions 481

- BrowseCallbackProc
- FMExtensionProc
- UndeleteFile

Shell Enumerations 561

- ASSOCDATA
- ASSOCF
- ASSOCKEY
- ASSOCSTR
- FOLDERFLAGS
- FOLDERVIEWMODE
- IURL_SETURL_FLAGS
- IURL_SETURL_INVOKECOMMAND_FLAGS
- SHCONTF
- SHGNO
- TRANSLATEURL_IN_FLAGS
- URLASSOCIATIONDIALOG_IN_FLAGS

Shell Functions 409

- CPlApplet
- DefScreenSaverProc
- DllGetVersion
- DLLGETVERSIONPROC
- DoEnvironmentSubst
- DragAcceptFiles
- DragFinish
- DragQueryFile
- DragQueryPoint
- FindEnvironmentString
- FindExecutable
- GetMenuContextHelpId
- GetWindowContextHelpId
- InetIsOffline
- MIMEAssociationDialog
- RegisterDialogClasses
- ScreenSaverConfigureDialog
- ScreenSaverProc
- SetMenuContextHelpId
- SetWindowContextHelpId

- SHAddToRecentDocs
- SHAppBarMessage
- SHBindToParent
- SHBrowseForFolder
- SHChangeNotify
- SHCreateDirectoryEx
- SHCreateProcessAsUser
- Shell_NotifyIcon
- ShellAbout
- ShellExecute
- ShellExecuteEx
- SHEmptyRecycleBin
- SHFileOperation
- SHFreeNameMappings
- SHGetDataFromIDList
- SHGetDesktopFolder
- SHGetDiskFreeSpace
- SHGetFileInfo
- SHGetFolderLocation
- SHGetFolderPath
- SHGetIconOverlayIndex
- SHGetInstanceExplorer
- SHGetMalloc
- SHGetNewLinkInfo
- SHGetPathFromIDList
- SHGetSettings
- SHGetSpecialFolderLocation
- SHGetSpecialFolderPath
- SHInvokePrinterCommand
- SHLoadInProc
- SHQueryRecycleBin
- SoftwareUpdateMessageBox
- TranslateURL
- URLAssociationDialog
- WinHelp

Shell Interfaces 139

- IACList 139
  - IACList::Expand
- IACList2 142
  - IACList2::GetOptions
  - IACList2::SetOptions
- IActiveDesktop Interface 144
  - IActiveDesktop::AddDesktopItem Method
  - IActiveDesktop::AddDesktopItemWithUI Method
  - IActiveDesktop::AddUrl Method
  - IActiveDesktop::ApplyChanges
  - IActiveDesktop::GenerateDesktopItemHtml
  - IActiveDesktop::GetDesktopItem
  - IActiveDesktop::GetDesktopItemByID
  - IActiveDesktop::GetDesktopItemBySource
  - IActiveDesktop::GetPattern
  - IActiveDesktop::GetDesktopItemCount
  - IActiveDesktop::GetDesktopItemOptions
  - IActiveDesktop::GetWallpaper
  - IActiveDesktop::GetWallpaperOptions
  - IActiveDesktop::ModifyDesktopItem
  - IActiveDesktop::RemoveDesktopItem
  - IActiveDesktop::SetDesktopItemOptions
  - IActiveDesktop::SetPattern
  - IActiveDesktop::SetWallpaper
  - IActiveDesktop::SetWallpaperOptions
- IAsyncOperation 160
  - IASyncOperation::EndOperation
  - IASyncOperation::GetAsyncMode
  - IASyncOperation::InOperation
  - IASyncOperation::SetAsyncMode
  - IASyncOperation::StartOperation
- IAutoComplete 165
  - IAutoComplete::Enable
  - IAutoComplete::Init
- IAutoComplete2 169
  - IAutoComplete2::GetOptions
  - IAutoComplete2::SetOptions
- IColumnProvider 172
  - IColumnProvider::GetColumnInfo
  - IColumnProvider::GetItemData
  - IColumnProvider::Initialize
- ICommDlgBrowser 176
  - ICommDlgBrowser::IncludeObject
  - ICommDlgBrowser::OnDefaultCommand
  - ICommDlgBrowser::OnStateChange
- ICommDlgBrowser2 179
  - ICommDlgBrowser2::GetDefaultMenuText
  - ICommDlgBrowser2::GetViewFlags
  - ICommDlgBrowser2::Notify
- IContextMenu 182
  - IContextMenu::GetCommandString
  - IContextMenu::InvokeCommand
  - IContextMenu::QueryContextMenu
- IContextMenu2 188
  - IContextMenu2::HandleMenuMsg
- IContextMenu3 190
  - IContextMenu3::HandleMenuMsg2
- ICopyHook 192
  - ICopyHook::CopyCallback
- ICurrentWorkingDirectory 194
  - ICurrentWorkingDirectory::GetDirectory
  - ICurrentWorkingDirectory::SetDirectory
- IDeskBand 196
  - IDeskBand::GetBandInfo
- IDockingWindow 198
  - IDockingWindow::CloseDW
  - IDockingWindow::ResizeBorderDW
  - IDockingWindow::ShowDW
- IDockingWindowFrame 201
  - IDockingWindowFrame::AddToolbar
  - IDockingWindowFrame::FindToolbar
  - IDockingWindowFrame::RemoveToolbar
- IDragSourceHelper 205

- Shell Interfaces *(continued)*
  - IDropTargetHelper *(continued)*
    - IDragSourceHelper::InitializeFromBitmap
    - IDragSourceHelper::InitializeFromWindow
  - IDropTargetHelper 208
    - IDropTargetHelper::DragEnter
    - IDropTargetHelper::DragLeave
    - IDropTargetHelper::DragOver
    - IDropTargetHelper::Drop
    - IDropTargetHelper::Show
  - IDockingWindowSite 213
    - IDockingWindowSite::GetBorderDW
    - IDockingWindowSite::RequestBorderSpaceDW
    - IDockingWindowSite::SetBorderSpaceDW
  - IEmptyVolumeCache 216
    - IEmptyVolumeCache::Deactivate
    - IEmptyVolumeCache::GetSpaceUsed
    - IEmptyVolumeCache::Initialize
    - IEmptyVolumeCache::Purge
    - IEmptyVolumeCache::ShowProperties
  - IEmptyVolumeCache2 223
    - IEmptyVolumeCache2::InitializeEx
  - IEmptyVolumeCacheCallBack 226
    - IEmptyVolumeCacheCallback::PurgeProgress
    - IEmptyVolumeCacheCallback::ScanProgress
  - IEnumExtraSearch 229
    - IEnumExtraSearch::Clone
    - IEnumExtraSearch::Next
    - IEnumExtraSearch::Reset
    - IEnumExtraSearch::Skip
  - IEnumIDList 232
    - IEnumIDList::Clone
    - IEnumIDList::Next
    - IEnumIDList::Reset
    - IEnumIDList::Skip
  - IExtractIcon 236
    - IExtractIcon::Extract
    - IExtractIcon::GetIconLocation
  - IExtractImage 240
    - IExtractImage::Extract
    - IExtractImage::GetLocation
  - IExtractImage2 243
    - IExtractImage2::GetDateStamp
  - IFileViewer 245
    - IFileViewer::PrintTo
    - IFileViewer::Show
    - IFileViewer::ShowInitialize
  - IFileViewerSite 248
    - IFileViewerSite::GetPinnedWindow
    - IFileViewerSite::SetPinnedWindow
  - IInputObject 249
    - IInputObject::HasFocusIO
    - IInputObject::TranslateAcceleratorIO
    - IInputObject::UIActivateIO
  - IInputObjectSite 252
    - IInputObjectSite::OnFocusChangeIS
  - INewShortcutHook 253
    - INewShortcutHook::GetExtension
    - INewShortcutHook::GetFolder
    - INewShortcutHook::GetName
    - INewShortcutHook::GetReferent
    - INewShortcutHook::SetFolder
    - INewShortcutHook::SetReferent
  - INotifyReplica 258
    - INotifyReplica::YouAreAReplica
  - IObjMgr 260
    - IObjMgr::Append
    - IObjMgr::Remove
  - IPersistFolder 261
    - IPersistFolder::Initialize
  - IPersistFolder2 263
    - IPersistFolder2::GetCurFolder
  - IPersistFolder3 264
    - IPersistFileSystemFolder::GetFolderTargetInfo
    - IPersistFileSystemFolder::InitializeEx
  - IProgressDialog 267
    - IProgressDialog::HasUserCancelled
    - IProgressDialog::SetAnimation
    - IProgressDialog::SetCancelMsg
    - IProgressDialog::SetLine
    - IProgressDialog::SetProgress
    - IProgressDialog::SetProgress64
    - IProgressDialog::SetTitle
    - IProgressDialog::StartProgressDialog
    - IProgressDialog::StopProgressDialog
    - IProgressDialog::Timer
  - IQueryAssociations 277
    - IQueryAssociations::GetData
    - IQueryAssociations::GetEnum
    - IQueryAssociations::GetKey
    - IQueryAssociations::GetString
    - IQueryAssociations::Init
  - IQueryInfo 284
    - IQueryInfo::GetInfoFlags
    - IQueryInfo::GetInfoTip
  - IReconcilableObject 286
    - IReconcilableObject::GetProgressFeedbackMaxEstimate
    - IReconcilableObject::Reconcile
  - IReconcileInitiator 291
    - IReconcileInitiator::SetAbortCallback
    - IReconcileInitiator::SetProgressFeedback
  - IRemoteComputer 294
    - IRemoteComputer::Initialize
  - IResolveShellLink 295
    - IResolveShellLink::ResolveShellLink
  - IRunnableTask 297
    - IRunnableTask::IsRunning
    - IRunnableTask::Kill
    - IRunnableTask::Resume

  IRunnableTask::Run
  IRunnableTask::Suspend
IShellBrowser 301
  IShellBrowser::BrowseObject
  IShellBrowser::EnableModelessSB
  IShellBrowser::GetControlWindow
  IShellBrowser::GetViewStateStream
  IShellBrowser::InsertMenusSB
  IShellBrowser::OnViewWindowActive
  IShellBrowser::QueryActiveShellView
  IShellBrowser::RemoveMenusSB
  IShellBrowser::SendControlMsg
  IShellBrowser::SetMenuSB
  IShellBrowser::SetStatusTextSB
  IShellBrowser::SetToolbarItems
  IShellBrowser::TranslateAcceleratorSB
IShellChangeNotify 316
  IShellChangeNotify::OnChange
IShellDetails 319
  IShellDetails::ColumnClick
  IShellDetails::GetDetailsOf
IShellExecuteHook 322
  IShellExecuteHook::Execute
IShellExtInit 324
  IShellExtInit::Initialize
IShellFolder 326
  IShellFolder::BindToObject
  IShellFolder::BindToStorage
  IShellFolder::CompareIDs
  IShellFolder::CreateViewObject
  IShellFolder::EnumObjects
  IShellFolder::GetAttributesOf
  IShellFolder::GetDisplayNameOf
  IShellFolder::GetUIObjectOf
  IShellFolder::ParseDisplayName
  IShellFolder::SetNameOf
IShellFolder2 343
  IShellFolder2::EnumSearches
  IShellFolder2::GetDefaultColumn
  IShellFolder2::GetDefaultColumnState
  IShellFolder2::GetDefaultSearchGUID
  IShellFolder2::GetDetailsEx
  IShellFolder2::GetDetailsOf
  IShellFolder2::MapNameToSCID
IShellIcon 350
  IShellIcon::GetIconOf
IShellIconOverlay 352
  IShellIconOverlay::GetOverlayIconIndex
  IShellIconOverlay::GetOverlayIndex
IShellIconOverlayIdentifier 355
  IShellIconOverlayIdentifier::GetOverlayInfo
  IShellIconOverlayIdentifier::GetPriority
  IShellIconOverlayIdentifier::IsMemberOf
IShellLink 359
  IShellLink::GetArguments
  IShellLink::GetDescription
  IShellLink::GetHotkey
  IShellLink::GetIconLocation
  IShellLink::GetIDList
  IShellLink::GetPath
  IShellLink::GetShowCmd
  IShellLink::GetWorkingDirectory
  IShellLink::Resolve
  IShellLink::SetArguments
  IShellLink::SetDescription
  IShellLink::SetHotkey
  IShellLink::SetIconLocation
  IShellLink::SetIDList
  IShellLink::SetPath
  IShellLink::SetRelativePath
  IShellLink::SetShowCmd
  IShellLink::SetWorkingDirectory
IShellLinkDataList 375
  IShellLinkDataList::AddDataBlock
  IShellLinkDataList::CopyDataBlock
  IShellLinkDataList::GetFlags
  IShellLinkDataList::RemoveDataBlock
  IShellLinkDataList::SetFlags
IShellPropSheetExt 381
  IShellPropSheetExt::AddPages
  IShellPropSheetExt::ReplacePage
IShellView 383
  IShellView::AddPropertySheetPages
  IShellView::CreateViewWindow
  IShellView::DestroyViewWindow
  IShellView::EnableModeless
  IShellView::EnableModelessSV
  IShellView::GetCurrentInfo
  IShellView::GetItemObject
  IShellView::Refresh
  IShellView::SaveViewState
  IShellView::SelectItem
  IShellView::TranslateAccelerator
  IShellView::UIActivate
IShellView2 395
  IShellView2::CreateViewWindow2
  IShellView2::GetView
  IShellView2::HandleRename
  IShellView2::SelectAndPositionItem
ITaskbarList 399
  ITaskbarList::ActivateTab
  ITaskbarList::AddTab
  ITaskbarList::DeleteTab
  ITaskbarList::HrInit
  ITaskbarList::SetActiveAlt
IUniformResourceLocator 403
  IUniformResourceLocator::GetURL
  IUniformResourceLocator::InvokeCommand
  IUniformResourceLocator::SetURL

IURLSearchHook *(continued)*
- IURLSearchHook::Translate

Shell Macros 571
- MAKEDLLVERULL
- SOANGLETENTHS
- SOPALETTEINDEX
- SOPALETTERGB
- SORGB
- SOSETRATIO

Shell Messages and Notifications 721
- ABM_ACTIVATE
- ABM_GETAUTOHIDEBAR
- ABM_GETSTATE
- ABM_GETTASKBARPOS
- ABM_NEW
- ABM_QUERYPOS
- ABM_REMOVE
- ABM_SETAUTOHIDEBAR
- ABM_SETPOS
- ABM_WINDOWPOSCHANGED
- ABN_FULLSCREENAPP
- ABN_POSCHANGED
- ABN_STATECHANGE
- ABN_WINDOWARRANGE
- CPL_DBLCLK
- CPL_EXIT
- CPL_GETCOUNT
- CPL_INIT
- CPL_INQUIRE
- CPL_NEWINQUIRE
- CPL_STARTWPARMS
- CPL_STOP
- FM_GETDRIVEINFO
- FM_GETFILESEL
- FM_GETFILESELLFN
- FM_GETFOCUS
- FM_GETSELCOUNT
- FM_GETSELCOUNTLFN
- FM_REFRESH_WINDOWS
- FM_RELOAD_EXTENSIONS
- FMEVENT_HELPMENUITEM
- FMEVENT_HELPSTRING
- FMEVENT_INITMENU
- FMEVENT_LOAD
- FMEVENT_SELCHANGE
- FMEVENT_TOOLBARLOAD
- FMEVENT_UNLOAD
- FMEVENT_USER_REFRESH
- WM_CPL_LAUNCH
- WM_CPL_LAUNCHED
- WM_DROPFILES
- WM_HELP
- WM_TCARD

Shell Structures 485
- APPBARDATA
- BROWSEINFO
- CIDA
- CMINVOKECOMMANDINFO
- CMINVOKECOMMANDINFOEX
- COMPONENT
- COMPONENTSOPT
- COMPPOS
- COMPSTATEINFO
- CPLINFO
- DATABLOCK_HEADER
- DESKBANDINFO
- DLLVERSIONINFO
- DLLVERSIONINFO2
- DROPFILES
- EXP_DARWIN_LINK
- EXP_SPECIAL_FOLDER
- EXP_SZ_LINK
- EXT_BUTTON
- EXTRASEARCH
- FILEDESCRIPTOR
- FILEGROUPDESCRIPTOR
- FMS_GETDRIVEINFO
- FMS_GETFILESEL
- FMS_HELPSTRING
- FMS_LOAD
- FMS_TOOLBARLOAD
- FOLDERSETTINGS
- FVSHOWINFO
- HELPINFO
- HELPWININFO
- IE4COMPONENT
- ITEMIDLIST
- MULTIKEYHELP
- NEWCPLINFO
- NOTIFYICONDATA
- NRESARRAY
- NT_CONSOLE_PROPS
- NT_FE_CONSOLE_PROPS
- PERSIST_FOLDER_TARGET_INFO
- SHCOLUMNDATA
- SHCOLUMNID
- SHCOLUMNINFO
- SHCOLUMNINIT
- SHCREATEPROCESSINFOW
- SHDESCRIPTIONID
- SHDRAGIMAGE
- SHELLEXECUTEINFO
- SHELLFLAGSTATE
- SHFILEINFO
- SHFILEOPSTRUCT
- SHITEMID
- SHNAMEMAPPING
- SHQUERYRBINFO

STRRET
SV2CVW2_PARAMS
URLINVOKECOMMANDINFO
WALLPAPEROPT

String Functions 575

ChrCmpI
IntlStrEqN
IntlStrEqNI
IntlStrEqWorker
MLLoadLibrary
SHStrDup
StrCat
StrCatBuff
StrChr
StrChrI
StrCmp
StrCmpI
StrCmpN
StrCmpNI
StrCpy
StrCpyN
StrCSpn
StrCSpnI
StrDup
StrFormatByteSize
StrFormatByteSize64A
StrFormatKBSize
StrFromTimeInterval
StrIsIntlEqual
StrNCat
StrPBrk
StrRChr
StrRChrI
StrRetToBuf
StrRetToStr
StrRStrI
StrSpn
StrStr
StrStrI
StrToInt
StrToIntEx
StrTrim
wnsprintf
wvnsprintf

APPENDIX B

Index B: Volume 1, Elements Listed Alphabetically

A

AbnormalTermination 750
AddAtom 346
AddUsersToEncryptedFile 655
AllocateUserPhysicalPages 261
AreFileApisANSI 481
AssignProcessToJobObject 74
AttachThreadInput 75

B

Beep 767
BindIoCompletionCallback 77
BY_HANDLE_FILE_INFORMATION 606

C

CallMsgFilter 420
CallNextHookEx 421
CallWndProc 422
CallWndRetProc 424
CancelIo 482
CBT_CREATEWND 456
CBTACTIVATESTRUCT 456
CBTProc 425
ChangeClipboardChain 363
CHARSETINFO 810
CloseClipboard 364
CloseHandle 404
CommandLineToArgvW 78
ConvertThreadToFiber 79
COPYDATASTRUCT 343
CopyFile 483
CopyFileEx 485
CopyMemory 263
CopyProgressRoutine 486
CountClipboardFormats 364
CreateDirectory 488
CreateDirectoryEx 489
CreateFiber 80
CreateFile 491
CreateHardLink 656
CreateIoCompletionPort 502
CreateJobObject 81
CreateProcess 82
CreateProcessAsUser 92
CreateProcessWithLogonW 100
CreateRemoteThread 107
CreateThread 110
CWPRETSTRUCT 457
CWPSTRUCT 458

D

DEBUGHOOKINFO 459
DebugProc 429
DecryptFile 658
DefineDosDevice 504
DeleteAtom 347
DeleteFiber 112
DeleteFile 506
DeleteVolumeMountPoint 659
DisableThreadLibraryCalls 217
DISKQUOTA_USER_INFORMATION 731
DllMain 219
DuplicateHandle 406

E

EFS_CERTIFICATE_BLOB 732
EFS_HASH_BLOB 733
EmptyClipboard 365
EncryptFile 660
ENCRYPTION_CERTIFICATE 733
ENCRYPTION_CERTIFICATE_HASH 734
ENCRYPTION_CERTIFICATE_HASH_LIST 735
ENCRYPTION_CERTIFICATE_LIST 735
EncryptionDisable 661
EnumClipboardFormats 366
EVENTMSG 460
EXCEPTION_POINTERS 759
EXCEPTION_RECORD 759
ExitProcess 113
ExitThread 115

F

FatalAppExit 768
FiberProc 116
FILE_NOTIFY_INFORMATION 609
FileEncryptionStatus 662
FileIOCompletionRoutine 507
FillMemory 264
FindAtom 348
FindClose 509
FindCloseChangeNotification 510
FINDEX_INFO_LEVELS 617
FINDEX_SEARCH_OPS 618
FindFirstChangeNotification 511
FindFirstFile 513
FindFirstFileEx 514
FindFirstVolume 663
FindFirstVolumeMountPoint 665
FindNextChangeNotification 517
FindNextFile 518
FindNextVolume 666
FindNextVolumeMountPoint 667
FindVolumeClose 668
FindVolumeMountPointClose 669
FlashWindow 769
FlashWindowEx 770
FLASHWINFO 783
FlushFileBuffers 519
FONTSIGNATURE 810
ForegroundIdleProc 432
FormatMessage 771
FreeEncryptionCertificateHashList 670
FreeEnvironmentStrings 117
FreeLibrary 222
FreeLibraryAndExitThread 223
FreeUserPhysicalPages 265

G

GET_FILEEX_INFO_LEVELS 619
GetAtomName 349
GetBinaryType 521
GetClipboardData 367
GetClipboardFormatName 368
GetClipboardOwner 369
GetClipboardSequenceNumber 370
GetClipboardViewer 371
GetCommandLine 117
GetCompressedFileSize 670
GetCurrentDirectory 522
GetCurrentFiber 207
GetCurrentProcess 118
GetCurrentProcessId 119
GetCurrentThread 120
GetCurrentThreadId 121
GetDiskFreeSpace 523
GetDiskFreeSpaceEx 525
GetDriveType 526
GetEnvironmentStrings 122
GetEnvironmentVariable 123
GetExceptionCode 751
GetExceptionInformation 753
GetExitCodeProcess 124
GetExitCodeThread 125
GetFiberData 207
GetFileAttributes 527
GetFileAttributesEx 530
GetFileInformationByHandle 531
GetFileSize 532
GetFileSizeEx 533
GetFileType 534
GetFullPathName 535
GetGuiResources 126
GetHandleInformation 413
GetLastError 776
GetLogicalDrives 536
GetLogicalDriveStrings 537
GetLongPathName 538
GetModuleFileName 224
GetModuleHandle 225
GetMsgProc 433
GetOpenClipboardWindow 371
GetPriorityClass 127
GetPriorityClipboardFormat 372
GetProcAddress 226
GetProcessAffinityMask 128
GetProcessHeap 266
GetProcessHeaps 267
GetProcessIoCounters 130
GetProcessPriorityBoost 130
GetProcessShutdownParameters 131
GetProcessTimes 132
GetProcessVersion 134
GetProcessWorkingSetSize 135
GetQueuedCompletionStatus 539
GetShortPathName 541
GetStartupInfo 136
GetTempFileName 543
GetTempPath 545
GetTextCharset 795
GetTextCharsetInfo 796
GetThreadPriority 137
GetThreadPriorityBoost 138
GetThreadTimes 139
GetVolumeInformation 672
GetVolumeNameForVolumeMountPoint 675
GetVolumePathName 676
GetWriteWatch 268
GlobalAddAtom 350
GlobalDeleteAtom 352

- GlobalFindAtom 353
- GlobalGetAtomName 354
- GlobalMemoryStatus 269

H

- HeapAlloc 271
- HeapCompact 273
- HeapCreate 275
- HeapDestroy 277
- HeapFree 278
- HeapLock 280
- HeapReAlloc 281
- HeapSize 284
- HeapUnlock 286
- HeapValidate 287
- HeapWalk 289

I

- IDiskQuotaControl 683
- AddUserName 684
    - AddUserSid 686
    - CreateEnumUsers 688
    - CreateUserBatch 690
    - DeleteUser 691
    - FindUserName 692
    - FindUserSid 693
    - GetDefaultQuotaLimit 694
    - GetDefaultQuotaLimitText 695
    - GetDefaultQuotaThreshold 696
    - GetDefaultQuotaThresholdText 697
    - GetQuotaLogFlags 698
    - GetQuotaState 699
    - GiveUserNameResolutionPriority 700
    - Initialize 701
    - InvalidateSidNameCache 702
    - SetDefaultQuotaLimit 703
    - SetDefaultQuotaThreshold 704
    - SetQuotaLogFlags 705
    - SetQuotaState 706
    - ShutdownNameResolution 707
- IDiskQuotaEvents 708
    - OnUserNameChanged 708
- IDiskQuotaUser 709
    - GetAccountStatus 710
    - GetID 711
    - GetName 712
    - GetQuotaInformation 713
    - GetQuotaLimit 714
    - GetQuotaLimitText 715
    - GetQuotaThreshold 716
    - GetQuotaThresholdText 716
    - GetQuotaUsed 717
    - GetQuotaUsedText 718
    - GetSid 719
    - GetSidLength 720
    - Invalidate 721
    - SetQuotaLimit 721
    - SetQuotaThreshold 722
- IDiskQuotaUserBatch 723
    - Add 724
    - Remove 725
    - RemoveAll 726
    - FlushToDisk 726
- IEnumDiskQuotaUsers 727
    - Clone 728
    - Next 729
    - Reset 730
    - Skip 730
- InitAtomTable 355
- Int32x32To64 546
- Int64ShllMod32 547
- Int64ShraMod32 548
- Int64ShrlMod32 549
- IO_COUNTERS 184
- IsBadCodePtr 290
- IsBadReadPtr 291
- IsBadStringPtr 293
- IsBadWritePtr 294
- IsClipboardFormatAvailable 373
- IsDBCSLeadByte 798
- IsDBCSLeadByteEx 799
- IsReparseTagHighLatency 736
- IsReparseTagMicrosoft 737
- IsReparseTagNameSurrogate 738
- IsTextUnicode 800

J

- JOBOBJECT_ASSOCIATE_COMPLETION_PORT 85
- JOBOBJECT_BASIC_ACCOUNTING_INFORMATION 188
- JOBOBJECT_BASIC_AND_IO_ACCOUNTING_INFORMATION 190
- JOBOBJECT_BASIC_LIMIT_INFORMATION 191
- JOBOBJECT_BASIC_PROCESS_ID_LIST 195
- JOBOBJECT_BASIC_UI_RESTRICTIONS 196
- JOBOBJECT_END_OF_JOB_TIME_INFORMATION 197
- JOBOBJECT_EXTENDED_LIMIT_INFORMATION 199
- JOBOBJECT_SECURITY_LIMIT_INFORMATION 200

JournalPlaybackProc 434
JournalRecordProc 437

K

KBDLLHOOKSTRUCT 460
KeyboardProc 439

L

LARGE_INTEGER 610
LoadLibrary 228
LoadLibraryEx 230
LOCALESIGNATURE 811
LockFile 550
LockFileEx 551
LowLevelKeyboardProc 441
LowLevelMouseProc 442

M

MAKEINTATOM 356
MapUserPhysicalPages 295
MapUserPhysicalPagesScatter 297
MEMORY_BASIC_INFORMATION 328
MEMORYSTATUS 331
MessageBeep 777
MessageProc 444
METAFILEPICT 378
MOUSEHOOKSTRUCT 462
MOUSEHOOKSTRUCTEX 463
MouseProc 446
MoveFile 553
MoveFileEx 554
MoveFileWithProgress 557
MoveMemory 298
MSLLHOOKSTRUCT 464
MulDiv 560
MultiByteToWideChar 802

O

OFSTRUCT 611
OpenClipboard 374
OpenJobObject 141
OpenProcess 142
OpenThread 144

P

PostQueuedCompletionStatus 561
PROCESS_HEAP_ENTRY 333
PROCESS_INFORMATION 202

Q

QueryDosDevice 562
QueryInformationJobObject 146
QueryRecoveryAgentsOnEncryptedFile 677
QueryUsersOnEncryptedFile 678
QueueUserWorkItem 148

R

RaiseException 754
ReadDirectoryChangesW 563
ReadFile 567
ReadFileEx 571
ReadFileScatter 574
RegisterClipboardFormat 375
RemoveDirectory 576
RemoveUsersFromEncryptedFile 679
ReplaceFile 577
ResetWriteWatch 299
ResumeThread 150

S

SearchPath 580
SetClipboardData 376
SetClipboardViewer 377
SetCurrentDirectory 581
SetEndOfFile 582
SetEnvironmentVariable 151
SetErrorMode 778
SetFileApisToANSI 583
SetFileApisToOEM 585
SetFileAttributes 586
SetFilePointer 588
SetFilePointerEx 591
SetHandleInformation 414
SetInformationJobObject 152
SetLastError 780
SetLastErrorEx 781
SetPriorityClass 153
SetProcessAffinityMask 155
SetProcessPriorityBoost 156
SetProcessShutdownParameters 157
SetProcessWorkingSetSize 159
SetThreadAffinityMask 161
SetThreadIdealProcessor 162
SetThreadPriority 163
SetThreadPriorityBoost 165
SetUnhandledExceptionFilter 756
SetUserFileEncryptionKey 680
SetVolumeLabel 593
SetVolumeMountPoint 681
SetWindowsHookEx 447

ShellProc 451
Sleep 166
SleepEx 167
STARTUPINFO 202
SuspendThread 169
SwitchToFiber 170
SwitchToThread 171
SysMsgProc 453

T

TerminateJobObject 172
TerminateProcess 173
TerminateThread 174
TEXT 812
ThreadProc 176
TlsAlloc 176
TlsFree 178
TlsGetValue 179
TlsSetValue 180
TranslateCharsetInfo 805

U

UInt32x32To64 594
ULARGE_INTEGER 611
UnhandledExceptionFilter 757
UnhookWindowsHookEx 455
UnlockFile 595
UnlockFileEx 596
UserHandleGrantAccess 181

V

VirtualAlloc 301
VirtualAllocEx 306
VirtualFree 311
VirtualFreeEx 313
VirtualLock 316
VirtualProtect 318
VirtualProtectEx 320
VirtualQuery 323
VirtualQueryEx 325
VirtualUnlock 326

W

WaitForInputIdle 182
WideCharToMultiByte 806
WIN32_FILE_ATTRIBUTE_DATA 612
WIN32_FIND_DATA 614
WM_ASKCBFORMATNAME 380
WM_CANCELJOURNAL 465
WM_CHANGECBCHAIN 381
WM_CLEAR 382
WM_COPY 383
WM_COPYDATA 343
WM_CUT 383
WM_DESTROYCLIPBOARD 384
WM_DRAWCLIPBOARD 385
WM_HSCROLLCLIPBOARD 386
WM_PAINTCLIPBOARD 387
WM_PASTE 388
WM_QUEUESYNC 467
WM_RENDERALLFORMATS 389
WM_RENDERFORMAT 390
WM_SIZECLIPBOARD 391
WM_VSCROLLCLIPBOARD 392
WriteFile 598
WriteFileEx 601
WriteFileGather 604

Z

ZeroMemory 327

APPENDIX B

Index B: Volume 2, Elements Listed Alphabetically

A

ACCEL 452
ActivateKeyboardLayout 467
AppendMenu 246

B

BlockInput 469
BM_CLICK 56
BM_GETCHECK 57
BM_GETIMAGE 58
BM_GETSTATE 59
BM_SETCHECK 60
BM_SETIMAGE 61
BM_SETSTATE 62
BM_SETSTYLE 63
BN_CLICKED 64
BN_DBLCLK 65
BN_DOUBLECLICKED 66
BN_KILLFOCUS 66
BN_SETFOCUS 67
BroadcastSystemMessage 614

C

CallWindowProc 682
CB_ADDSTRING 84
CB_DELETESTRING 85
CB_DIR 86
CB_FINDSTRING 88
CB_FINDSTRINGEXACT 89
CB_GETCOUNT 90
CB_GETCURSEL 91
CB_GETDROPPEDCONTROLRECT 92
CB_GETDROPPEDSTATE 93
CB_GETDROPPEDWIDTH 93
CB_GETEDITSEL 94
CB_GETEXTENDEDUI 95
CB_GETHORIZONTALEXTENT 96
CB_GETITEMDATA 97
CB_GETITEMHEIGHT 98
CB_GETLBTEXT 99
CB_GETLBTEXTLEN 100
CB_GETLOCALE 101
CB_GETTOPINDEX 102
CB_INITSTORAGE 103
CB_INSERTSTRING 104
CB_LIMITTEXT 105
CB_RESETCONTENT 106
CB_SELECTSTRING 106
CB_SETCURSEL 108
CB_SETDROPPEDWIDTH 108
CB_SETEDITSEL 109
CB_SETEXTENDEDUI 110
CB_SETHORIZONTALEXTENT 111
CB_SETITEMDATA 112
CB_SETITEMHEIGHT 113
CB_SETLOCALE 114
CB_SETTOPINDEX 115
CB_SHOWDROPDOWN 116
CBN_CLOSEUP 117
CBN_DBLCLK 118
CBN_DROPDOWN 119
CBN_EDITCHANGE 120
CBN_EDITUPDATE 120
CBN_ERRSPACE 121
CBN_KILLFOCUS 122
CBN_SELCHANGE 123
CBN_SELENDCANCEL 124
CBN_SELENDOK 125
CBN_SETFOCUS 125
CharLower 323
CharLowerBuff 324
CharNext 325
CharNextExA 326
CharPrev 327
CharPrevExA 327
CharToOem 328
CharToOemBuff 329
CharUpper 330
CharUpperBuff 331
CheckDlgButton 53
CheckMenuItem 249
CheckMenuRadioItem 250
CheckRadioButton 54
ClipCursor 200
COMBOBOXINFO 77

COMPAREITEMSTRUCT 78
CompareString 332
CopyAcceleratorTable 446
CopyCursor 201
CopyIcon 218
CreateAcceleratorTable 447
CreateCaret 192
CreateCursor 202
CreateDialog 537
CreateDialogIndirect 539
CreateDialogIndirectParam 541
CreateDialogParam 543
CreateIcon 219
CreateIconFromResource 221
CreateIconFromResourceEx 222
CreateIconIndirect 224
CreateMDIWindow 653
CreateMenu 251
CreatePopupMenu 252
CURSORINFO 216

D

DefDlgProc 545
DefFrameProc 655
DefMDIChildProc 657
DefWindowProc 684
DeleteMenu 253
DestroyAcceleratorTable 448
DestroyCaret 193
DestroyCursor 203
DestroyIcon 225
DestroyMenu 254
DialogBox 546
DialogBoxIndirect 547
DialogBoxIndirectParam 550
DialogBoxParam 552
DialogProc 553
DispatchMessage 616
DlgDirListComboBox 73
DlgDirSelectComboBoxEx 75
DLGITEMTEMPLATE 582
DLGITEMTEMPLATEEX 584
DLGTEMPLATE 586
DLGTEMPLATEEX 589
DM_GETDEFID 595
DM_REPOSITION 596
DM_SETDEFID 596
DragDetect 372
DrawIcon 225
DrawIconEx 227
DRAWITEMSTRUCT 80
DrawMenuBar 255
DuplicateIcon 229

E

EnableMenuItem 256
EnableScrollBar 134
EnableWindow 470
EndDialog 555
EndMenu 258
EnumProps 687
EnumPropsEx 688
ExtractAssociatedIcon 229
ExtractIcon 231
ExtractIconEx 232

F

FoldString 336

G

GET_APPCOMMAND_LPARAM 437
GET_DEVICE_LPARAM 438
GET_KEYSTATE_LPARAM 439
GET_KEYSTATE_WPARAM 440
GET_NCHITTEST_WPARAM 440
GET_WHEEL_DELTA_WPARAM 441
GET_XBUTTON_WPARAM 441
GetActiveWindow 472
GetAsyncKeyState 472
GetCapture 373
GetCaretBlinkTime 194
GetCaretPos 195
GetClipCursor 204
GetComboBoxInfo 76
GetCursor 205
GetCursorInfo 206
GetCursorPos 207
GetDialogBaseUnits 556
GetDlgCtrlID 557
GetDlgItem 558
GetDlgItemInt 559
GetDlgItemText 561
GetDoubleClickTime 373
GetFocus 474
GetIconInfo 233
GetInputState 617
GetKeyboardLayout 475
GetKeyboardLayoutList 476
GetKeyboardLayoutName 477
GetKeyboardState 478
GetKeyNameText 479
GetKeyState 480
GetLastInputInfo 482
GetMenu 258
GetMenuBarInfo 259

GetMenuCheckMarkDimensions 260
GetMenuDefaultItem 261
GetMenuInfo 262
GetMenuItemCount 263
GetMenuItemID 264
GetMenuItemInfo 264
GetMenuItemRect 266
GetMenuState 267
GetMenuString 269
GetMessage 618
GetMessageExtraInfo 620
GetMessagePos 621
GetMessageTime 622
GetMouseMovePointsEx 374
GetNextDlgGroupItem 562
GetNextDlgTabItem 563
GetProp 689
GetQueueStatus 622
GetScrollBarInfo 136
GetScrollInfo 137
GetScrollPos 139
GetScrollRange 140
GetStringTypeA 338
GetStringTypeEx 342
GetStringTypeW 346
GetSubMenu 270
GetSystemMenu 271

H

HARDWAREINPUT 509
HideCaret 195
HiliteMenuItem 272

I

ICONINFO 239
ICONMETRICS 240
INPUT 510
InSendMessage 624
InSendMessageEx 625
InsertMenu 273
InsertMenuItem 276
IsCharAlpha 350
IsCharAlphaNumeric 351
IsCharLower 352
IsCharUpper 352
IsDialogMessage 564
IsDlgButtonChecked 55
IsMenu 278
IsWindowEnabled 483

K

keybd_event 483
KEYBDINPUT 511
KillTimer 674

L

LASTINPUTINFO 513
LoadAccelerators 449
LoadCursor 208
LoadCursorFromFile 209
LoadIcon 235
LoadKeyboardLayout 485
LoadMenu 278
LoadMenuIndirect 279
LoadString 353
LookupIconIdFromDirectory 236
LookupIconIdFromDirectoryEx 238
lstrcat 354
lstrcmp 355
lstrcmpi 356
lstrcpy 358
lstrcpyn 359
lstrlen 360

M

MapDialogRect 566
MapVirtualKey 487
MapVirtualKeyEx 489
MDICREATESTRUCT 659
MDINEXTMENU 297
MEASUREITEMSTRUCT 82
MENUBARINFO 297
MENUEX_TEMPLATE_HEADER 298
MENUEX_TEMPLATE_ITEM 299
MENUGETOBJECTINFO 301
MENUINFO 302
MenuItemFromPoint 280
MENUITEMINFO 304
MENUITEMTEMPLATE 309
MENUITEMTEMPLATEHEADER 310
MessageBox 567
MessageBoxEx 572
MessageBoxIndirect 577
ModifyMenu 281
mouse_event 376
MOUSEINPUT 514
MOUSEMOVEPOINT 385
MSG 645
MSGBOXPARAMS 593

O

OemKeyScan 491
OemToChar 361
OemToCharBuff 361

P

PeekMessage 626
PostMessage 628
PostQuitMessage 630
PostThreadMessage 631
PropEnumProc 690
PropEnumProcEx 691

Q

QueryPerformanceCounter 675
QueryPerformanceFrequency 676

R

RegisterHotKey 492
RegisterWindowMessage 632
ReleaseCapture 379
RemoveMenu 284
RemoveProp 692
ReplyMessage 633

S

SBM_ENABLE_ARROWS 157
SBM_GETPOS 158
SBM_GETRANGE 159
SBM_GETSCROLLINFO 159
SBM_SETPOS 161
SBM_SETRANGE 162
SBM_SETRANGEREDRAW 163
SBM_SETSCROLLINFO 164
SCROLLBARINFO 154
ScrollDC 142
SCROLLINFO 155
ScrollWindow 143
ScrollWindowEx 145
SendAsyncProc 634
SendDlgItemMessage 579
SendInput 494
SendMessage 636
SendMessageCallback 637
SendMessageTimeout 639
SendNotifyMessage 640
SetActiveWindow 495
SetCapture 380
SetCaretBlinkTime 196
SetCaretPos 197
SetCursor 211
SetCursorPos 212
SetDlgItemInt 580
SetDlgItemText 581
SetDoubleClickTime 381
SetFocus 496
SetKeyboardState 497
SetMenu 285
SetMenuDefaultItem 286
SetMenuInfo 287
SetMenuItemBitmaps 288
SetMenuItemInfo 290
SetMessageExtraInfo 642
SetProp 693
SetScrollInfo 147
SetScrollPos 149
SetScrollRange 151
SetSystemCursor 213
SetTimer 677
ShowCaret 198
ShowCursor 215
ShowScrollBar 152
STM_GETICON 173
STM_GETIMAGE 174
STM_SETICON 175
STM_SETIMAGE 176
STN_CLICKED 177
STN_DBLCLK 177
STN_DISABLE 178
STN_ENABLE 179
SwapMouseButton 382

T

TimerProc 678
ToAscii 498
ToAsciiEx 499
ToUnicode 501
ToUnicodeEx 503
TPMPARAMS 310
TrackMouseEvent 383
TRACKMOUSEEVENT 385
TrackPopupMenu 291
TrackPopupMenuEx 294
TranslateAccelerator 450
TranslateMDISysAccel 658
TranslateMessage 642

U

UnloadKeyboardLayout 505
UnregisterHotKey 506

V

VkKeyScan 507
VkKeyScanEx 508

W

WaitMessage 644
WindowProc 685
WM_ACTIVATE 517
WM_APP 646
WM_APPCOMMAND 387
WM_CAPTURECHANGED 390
WM_CHANGEUISTATE 453
WM_CHAR 518
WM_COMMAND 311
WM_COMPAREITEM 126
WM_CONTEXTMENU 312
WM_CTLCOLORBTN 68
WM_CTLCOLORDLG 597
WM_CTLCOLORSCROLLBAR 165
WM_CTLCOLORSTATIC 180
WM_DEADCHAR 520
WM_DRAWITEM 127
WM_ENTERIDLE 599
WM_ENTERMENULOOP 314
WM_ERASEBKGND 241
WM_EXITMENULOOP 315
WM_GETDLGCODE 600
WM_GETFONT 50
WM_GETHOTKEY 522
WM_HOTKEY 523
WM_HSCROLL 166
WM_ICONERASEBKGND 242
WM_INITDIALOG 601
WM_INITMENU 455
WM_INITMENUPOPUP 456
WM_KEYDOWN 524
WM_KEYUP 526
WM_KILLFOCUS 527
WM_LBUTTONDBLCLK 391
WM_LBUTTONDOWN 392
WM_LBUTTONUP 394
WM_MBUTTONDBLCLK 395
WM_MBUTTONDOWN 397
WM_MBUTTONUP 398
WM_MDIACTIVATE 661
WM_MDICASCADE 662
WM_MDICREATE 663
WM_MDIDESTROY 665
WM_MDIGETACTIVE 666
WM_MDIICONARRANGE 667
WM_MDIMAXIMIZE 667
WM_MDINEXT 668
WM_MDIREFRESHMENU 669
WM_MDIRESTORE 670
WM_MDISETMENU 671
WM_MDITILE 672
WM_MEASUREITEM 128
WM_MENUCHAR 456
WM_MENUCOMMAND 316
WM_MENUDRAG 316
WM_MENUGETOBJECT 317
WM_MENURBUTTONUP 318
WM_MENUSELECT 458
WM_MOUSEACTIVATE 399
WM_MOUSEHOVER 401
WM_MOUSELEAVE 402
WM_MOUSEMOVE 403
WM_MOUSEWHEEL 404
WM_NCHITTEST 407
WM_NCLBUTTONDBLCLK 409
WM_NCLBUTTONDOWN 410
WM_NCLBUTTONUP 411
WM_NCMBUTTONDBLCLK 412
WM_NCMBUTTONDOWN 414
WM_NCMBUTTONUP 415
WM_NCMOUSEHOVER 416
WM_NCMOUSELEAVE 417
WM_NCMOUSEMOVE 418
WM_NCRBUTTONDBLCLK 419
WM_NCRBUTTONDOWN 420
WM_NCRBUTTONUP 421
WM_NCXBUTTONDBLCLK 423
WM_NCXBUTTONDOWN 424
WM_NCXBUTTONUP 426
WM_NEXTDLGCTL 602
WM_NEXTMENU 319
WM_PAINTICON 243
WM_QUERYUISTATE 459
WM_RBUTTONDBLCLK 427
WM_RBUTTONDOWN 429
WM_RBUTTONUP 430
WM_SETCURSOR 217
WM_SETFOCUS 528
WM_SETFONT 51
WM_SETHOTKEY 529
WM_SYSCHAR 460
WM_SYSCOMMAND 462
WM_SYSDEADCHAR 530
WM_SYSKEYDOWN 532
WM_SYSKEYUP 534
WM_TIMER 679
WM_UNINITMENUPOPUP 320
WM_UPDATEUISTATE 464
WM_USER 647
WM_VSCROLL 168
WM_XBUTTONDBLCLK 431
WM_XBUTTONDOWN 433
WM_XBUTTONUP 435
wsprintf 362
wvsprintf 366

APPENDIX B

Index B: Volume 3, Elements Listed Alphabetically

A

AbortPath 586
AlphaBlend 66
AngleArc 371
AnimatePalette 202
Arc 373
ArcTo 375

B

BeginPaint 512
BeginPath 587
BitBlt 69
BITMAP 116
BITMAPCOREHEADER 118
BITMAPCOREINFO 119
BITMAPFILEHEADER 121
BITMAPINFO 122
BITMAPINFOHEADER 123
BITMAPV4HEADER 128
BITMAPV5HEADER 133
BLENDFUNCTION 140

C

CancelDC 295
ChangeDisplaySettings 296
ChangeDisplaySettingsEx 299
Chord 354
ClientToScreen 252
CloseEnhMetaFile 397
CloseFigure 589
COLORADJUSTMENT 142
COLORREF 223
CombineTransform 253
CopyEnhMetaFile 398
CopyRect 619
CreateBitmap 71
CreateBitmapIndirect 73
CreateBrushIndirect 157
CreateCompatibleBitmap 74
CreateCompatibleDC 303
CreateDC 304
CreateDIBitmap 76
CreateDIBPatternBrushPt 159
CreateDIBSection 78
CreateEnhMetaFile 399
CreateHalftonePalette 203
CreateHatchBrush 160
CreateIC 306
CreatePalette 204
CreatePatternBrush 162
CreatePen 605
CreatePenIndirect 607
CreateSolidBrush 163

D

DeleteDC 307
DeleteEnhMetaFile 401
DeleteObject 308
DIBSECTION 145
DISPLAY_DEVICE 344
DPtoLP 254
DrawAnimatedRects 513
DrawCaption 514
DrawEdge 516
DrawEscape 309
DrawFocusRect 518
DrawFrameControl 519
DrawState 522
DrawStateProc 525

E

Ellipse 356
EMR 421
EMRALPHABLEND 423
EMRANGLEARC 425
EMRARC 426
EMRARCTO 426
EMRCHORD 426
EMRPIE 426
EMRBITBLT 427
EMRCREATEBRUSHINDIRECT 431
EMRCREATECOLORSPACE 432
EMRCREATEDIBPATTERNBRUSHPT 434

EMRCREATEMONOBRUSH 435
EMRCREATEPALETTE 436
EMRCREATEPEN 437
EMRELLIPSE
EMRRECTANGLE 437
EMREOF 438
EMREXCLUDECLIPRECT 439
EMRINTERSECTCLIPRECT 439
EMREXTCREATEFONTINDIRECTW 439
EMREXTCREATEPEN 440
EMREXTFLOODFILL 441
EMREXTSELECTCLIPRGN 442
EMREXTTEXTOUTA 443
EMREXTTEXTOUTW 443
EMRFILLPATH
EMRSTROKEANDFILLPATH 444
EMRSTROKEPATH 444
EMRFILLRGN 444
EMRFORMAT 445
EMRFRAMERGN 446
EMRGDICOMMENT 447
EMRGLSBOUNDEDRECORD 448
EMRGLSRECORD 449
EMRGRADIENTFILL 450
EMRINVERTRGN 451
EMRPAINTRGN 451
EMRLINETO 452
EMRMOVETOEX 452
EMRMASKBLT 452
EMRMODIFYWORLDTRANSFORM 455
EMROFFSETCLIPRGN 455
EMRPIXELFORMAT 456
EMRPLGBLT 457
EMRPOLYDRAW 459
EMRPOLYDRAW16 460
EMRPOLYLINE 461
EMRPOLYBEZIER 461
EMRPOLYGON 461
EMRPOLYBEZIERTO 461
EMRPOLYLINETO 461
EMRPOLYLINE16 462
EMRPOLYBEZIER16 462
EMRPOLYGON16 462
EMRPOLYBEZIERTO16 462
EMRPOLYLINETO16 462
EMRPOLYPOLYLINE 463
EMRPOLYPOLYGON 463
EMRPOLYPOLYLINE16 464
EMRPOLYPOLYGON16 464
EMRPOLYTEXTOUTA 464
EMRPOLYTEXTOUTW 464
EMRRESIZEPALETTE 466
EMRRESTOREDC 466
EMRROUNDRECT 467
EMRSCALEVIEWPORTEXTEX 468
EMRSCALEWINDOWEXTEX 468
EMRSELECTOBJECT 469
EMRDELETEOBJECT 469
EMRSELECTPALETTE 470
EMRSETARCDIRECTION 471
EMRSETBKCOLOR 471
EMRSETTEXTCOLOR 471
EMRSETCOLORADJUSTMENT 472
EMRSETCOLORSPACE 469
EMRSELECTCOLORSPACE 469
EMRDELETECOLORSPACE 469
EMRSETDIBITSTODEVICE 472
EMRSETICMPROFILE 474
EMRSETMAPPERFLAGS 475
EMRSETMITERLIMIT 476
EMRSETPALETTEENTRIES 476
EMRSETPIXELV 477
EMRSETVIEWPORTEXTEX 478
EMRSETWINDOWEXTEX 478
EMRSETVIEWPORTORGEX 479
EMRSETWINDOWORGEX 479
EMRSETBRUSHORGEX 479
EMRSETWORLDTRANSFORM 479
EMRSTRETCHBLT 480
EMRSTRETCHDIBITS 482
EMRTEXT 484
EMRTRANSPARENTBLT 485
EndPaint 526
EndPath 590
Enhanced Metafile Records with No Parameters 487
Enhanced Metafile Records with One Parameter 487
EnhMetaFileProc 402
ENHMETAHEADER 488
ENHMETARECORD 491
EnumDisplayDevices 310
EnumDisplaySettings 311
EnumDisplaySettingsEx 313
EnumEnhMetaFile 403
EnumObjects 316
EnumObjectsProc 317
EqualRect 619
ExcludeClipRect 177
ExcludeUpdateRgn 526
ExtCreatePen 608
ExtFloodFill 80
EXTLOGPEN 611
ExtSelectClipRgn 178

F

FillPath 591
FillRect 357
FlattenPath 592

FrameRect 358

G

GdiComment 404
GdiFlush 527
GdiGetBatchLimit 529
GdiSetBatchLimit 530
GetArcDirection 376
GetBitmapDimensionEx 82
GetBkColor 531
GetBkMode 531
GetBoundsRect 532
GetBrushOrgEx 164
GetBValue 226
GetClipBox 180
GetClipRgn 181
GetColorAdjustment 205
GetCurrentObject 318
GetCurrentPositionEx 255
GetDC 319
GetDCBrushColor 320
GetDCEx 321
GetDCOrgEx 323
GetDCPenColor 324
GetDeviceCaps 325
GetDIBColorTable 83
GetDIBits 84
GetEnhMetaFile 407
GetEnhMetaFileBits 408
GetEnhMetaFileHeader 411
GetEnhMetaFilePaletteEntries 412
GetGraphicsMode 256
GetGValue 226
GetMapMode 257
GetMetaRgn 182
GetMiterLimit 593
GetNearestColor 206
GetNearestPaletteIndex 207
GetObject 331
GetObjectType 333
GetPaletteEntries 208
GetPath 594
GetPixel 87
GetRandomRgn 183
GetROP2 533
GetRValue 227
GetStockObject 334
GetStretchBltMode 88
GetSysColorBrush 165
GetSystemPaletteEntries 209
GetSystemPaletteUse 210
GetUpdateRect 535
GetUpdateRgn 536
GetViewportExtEx 258
GetViewportOrgEx 259
GetWindowDC 537
GetWindowExtEx 260
GetWindowOrgEx 261
GetWindowRgn 539
GetWinMetaFileBits 413
GetWorldTransform 262
GRADIENT_RECT 146
GRADIENT_TRIANGLE 147
GradientFill 88
GrayString 540

H

HANDLETABLE 491
HTUI_ColorAdjustment 211

I

InflateRect 620
IntersectClipRect 184
IntersectRect 621
InvalidateRect 542
InvalidateRgn 543
InvertRect 359
IsRectEmpty 622

L

LineDDA 377
LineDDAProc 378
LineTo 379
LoadBitmap 90
LockWindowUpdate 544
LOGBRUSH 169
LOGBRUSH32 172
LOGPALETTE 224
LOGPEN 615
LPtoDP 263

M

MAKEPOINTS 631
MAKEROP4 152
MapWindowPoints 264
MaskBlt 92
ModifyWorldTransform 265
MoveToEx 381

O

OffsetClipRgn 185
OffsetRect 623

OffsetViewportOrgEx 267
OffsetWindowOrgEx 268
OutputProc 546

P

PaintDesktop 547
PAINTSTRUCT 561
PALETTEENTRY 224
PALETTEINDEX 228
PALETTERGB 229
PatBlt 166
PathToRegion 596
Pie 360
PlayEnhMetaFile 415
PlayEnhMetaFileRecord 417
PlgBlt 95
POINT 629
POINTL 492
POINTS 629
POINTSTOPOINT 631
POINTTOPOINTS 632
PolyBezier 382
PolyBezierTo 383
PolyDraw 384
Polygon 362
Polyline 386
PolylineTo 387
PolyPolygon 363
PolyPolyline 388
PtInRect 624
PtVisible 186

R

RealizePalette 213
RECT 630
Rectangle 364
RECTL 493
RectVisible 187
RedrawWindow 547
ReleaseDC 336
ResetDC 337
ResizePalette 214
RestoreDC 338
RGB 230
RGBQUAD 148
RGBTRIPLE 149
RoundRect 365

S

SaveDC 339
ScaleViewportExtEx 269
ScaleWindowExtEx 270
ScreenToClient 271
SelectClipPath 188
SelectClipRgn 189
SelectObject 340
SelectPalette 215
SetArcDirection 389
SetBitmapDimensionEx 97
SetBkColor 550
SetBkMode 551
SetBoundsRect 552
SetBrushOrgEx 168
SetColorAdjustment 216
SetDCBrushColor 342
SetDCPenColor 343
SetDIBColorTable 98
SetDIBits 100
SetDIBitsToDevice 102
SetEnhMetaFileBits 418
SetGraphicsMode 272
SetMapMode 274
SetMetaRgn 191
SetMiterLimit 597
SetPaletteEntries 217
SetPixel 105
SetPixelV 106
SetRect 625
SetRectEmpty 626
SetROP2 554
SetStretchBltMode 107
SetSystemPaletteUse 219
SetViewportExtEx 276
SetViewportOrgEx 278
SetWindowExtEx 279
SetWindowOrgEx 280
SetWindowRgn 556
SetWinMetaFileBits 419
SetWorldTransform 282
SIZE 150
StretchBlt 109
StretchDIBits 111
StrokeAndFillPath 598
StrokePath 599
SubtractRect 626

T

TransparentBlt 114
TRIVERTEX 151

U

UnionRect 628
UnrealizeObject 221

UpdateColors 222
UpdateWindow 557

V

ValidateRect 558
ValidateRgn 559
VIDEOPARAMETERS 345

W

WidenPath 600
WindowFromDC 560
WM_DEVMODECHANGE 350
WM_DISPLAYCHANGE 562
WM_NCPAINT 563
WM_PAINT 564
WM_PALETTECHANGED 231
WM_PALETTEISCHANGING 232
WM_PRINT 566
WM_PRINTCLIENT 567
WM_QUERYNEWPALETTE 233
WM_SETREDRAW 568
WM_SYNCPAINT 569
WM_SYSCOLORCHANGE 234

X

XFORM 284

APPENDIX B

Index B: Volume 4, Elements Listed Alphabetically

A

ACM_OPEN 127
ACM_PLAY 128
ACM_STOP 129
ACN_START 136
ACN_STOP 136
AddPropSheetPageProc 435
Animate_Close 130
Animate_Create 130
Animate_Open 131
Animate_OpenEx 132
Animate_Play 133
Animate_Seek 134
Animate_Stop 135

C

CBEM_DELETEITEM 145
CBEM_GETCOMBOCONTROL 146
CBEM_GETEDITCONTROL 146
CBEM_GETEXTENDEDSTYLE 147
CBEM_GETIMAGELIST 147
CBEM_GETITEM 148
CBEM_GETUNICODEFORMAT 149
CBEM_HASEDITCHANGED 149
CBEM_INSERTITEM 150
CBEM_SETEXTENDEDSTYLE 151
CBEM_SETIMAGELIST 151
CBEM_SETITEM 152
CBEM_SETUNICODEFORMAT 153
CBEN_BEGINEDIT 154
CBEN_DELETEITEM 154
CBEN_DRAGBEGIN 155
CBEN_ENDEDIT 155
CBEN_GETDISPINFO 156
CBEN_INSERTITEM 157
CCM_GETUNICODEFORMAT 86
CCM_GETVERSION 87
CCM_SETUNICODEFORMAT 88
CCM_SETVERSION 89
COLORSCHEME 104
COMBOBOXEXITEM 158
CreatePropertySheetPage 435
CreateStatusWindow 562
CreateUpDownControl 735

D

DateTime_GetMonthCal 205
DateTime_GetMonthCalColor 205
DateTime_GetMonthCalFont 207
DateTime_GetRange 207
DateTime_GetSystemtime 208
DateTime_SetFormat 209
DateTime_SetMonthCalColor 210
DateTime_SetMonthCalFont 211
DateTime_SetRange 211
DateTime_SetSystemtime 212
DestroyPropertySheetPage 436
DL_BEGINDRAG 228
DL_CANCELDRAG 229
DL_DRAGGING 230
DL_DROPPED 230
DRAGLISTINFO 231
DrawInsert 226
DrawStatusText 563
DTM_GETMCCOLOR 197
DTM_GETMCFONT 198
DTM_GETMONTHCAL 198
DTM_GETRANGE 199
DTM_GETSYSTEMTIME 200
DTM_SETFORMAT 200
DTM_SETMCCOLOR 201
DTM_SETMCFONT 202
DTM_SETRANGE 203
DTM_SETSYSTEMTIME 204
DTN_CLOSEUP 213
DTN_DATETIMECHANGE 214
DTN_DROPDOWN 215
DTN_FORMAT 216
DTN_FORMATQUERY 216
DTN_USERSTRING 217
DTN_WMKEYDOWN 218

E

ExtensionPropSheetPageProc 437

F

FIRST_IPADDRESS 325
FlatSB_EnableScrollBar............................. 236
FlatSB_GetScrollInfo.................................. 237
FlatSB_GetScrollPos.................................. 238
FlatSB_GetScrollProp 239
FlatSB_GetScrollRange 241
FlatSB_SetScrollInfo 242
FlatSB_SetScrollPos 243
FlatSB_SetScrollProp................................. 244
FlatSB_SetScrollRange.............................. 247
FlatSB_ShowScrollBar 248
FORWARD_WM_NOTIFY 92
FOURTH_IPADDRESS.............................. 326

G

GetEffectiveClientRect 81
GetMUILanguage 82

H

HANDLE_WM_NOTIFY 93
HDHITTESTINFO....................................... 301
HDITEM.. 303
HDLAYOUT 306
HDM_CLEARFILTER 258
HDM_CREATEDRAGIMAGE 259
HDM_DELETEITEM.................................. 259
HDM_EDITFILTER.................................... 260
HDM_GETBITMAPMARGIN....................... 261
HDM_GETIMAGELIST.............................. 261
HDM_GETITEM 262
HDM_GETITEMCOUNT 262
HDM_GETITEMRECT 263
HDM_GETORDERARRAY 264
HDM_GETUNICODEFORMAT................... 265
HDM_HITTEST 265
HDM_INSERTITEM 266
HDM_LAYOUT... 266
HDM_ORDERTOINDEX 267
HDM_SETBITMAPMARGIN 268
HDM_SETFILTERCHANGETIMEOUT....... 268
HDM_SETHOTDIVIDER............................ 269
HDM_SETIMAGELIST 270
HDM_SETITEM.. 271
HDM_SETORDERARRAY.......................... 271
HDM_SETUNICODEFORMAT 272
HDN_BEGINDRAG.................................... 291
HDN_BEGINTRACK 292
HDN_DIVIDERDBLCLICK 292
HDN_ENDDRAG.. 293
HDN_ENDTRACK...................................... 293
HDN_FILTERBTNCLICK............................ 294
HDN_FILTERCHANGE 295
HDN_GETDISPINFO.................................. 295
HDN_ITEMCHANGED 296
HDN_ITEMCHANGING.............................. 297
HDN_ITEMCLICK...................................... 297
HDN_ITEMDBLCLICK................................ 298
HDN_TRACK... 298
HDTEXTFILTER Structure 306
Header_ClearFilter 274
Header_CreateDragImage 275
Header_DeleteItem..................................... 275
Header_EditFilter.. 276
Header_GetBitmapMargin 277
Header_GetImageList................................. 278
Header_GetItem 278
Header_GetItemCount................................ 279
Header_GetItemRect.................................. 280
Header_GetOrderArray 281
Header_GetUnicodeFormat........................ 282
Header_InsertItem 282
Header_Layout 283
Header_OrderToIndex................................ 284
Header_SetBitmapMargin 285
Header_SetFilterChangeTimeout............... 286
Header_SetHotDivider................................ 286
Header_SetImageList 287
Header_SetItem.. 288
Header_SetOrderArray............................... 289
Header_SetUnicodeFormat......................... 290
HKM_GETHOTKEY.................................... 315
HKM_SETHOTKEY 316
HKM_SETRULES 317

I

INDEXTOSTATEIMAGEMASK 94
InitCommonControls 83
InitCommonControlsEx................................. 83
INITCOMMONCONTROLSEX 104
InitializeFlatSB 235
InitMUILanguage 84
IPM_CLEARADDRESS 320
IPM_GETADDRESS................................... 321
IPM_ISBLANK 322
IPM_SETADDRESS 322
IPM_SETFOCUS.. 323
IPM_SETRANGE.. 323
IPN_FIELDCHANGED................................ 324

L

LBItemFromPt... 227

M

MakeDragList 228
MAKEIPADDRESS 326
MAKEIPRANGE 327
MCHITTESTINFO 382
MCM_GETCOLOR 339
MCM_GETCURSEL 340
MCM_GETFIRSTDAYOFWEEK 341
MCM_GETMAXSELCOUNT 342
MCM_GETMAXTODAYWIDTH 342
MCM_GETMINREQRECT 343
MCM_GETMONTHDELTA 344
MCM_GETMONTHRANGE 345
MCM_GETRANGE 346
MCM_GETSELRANGE 347
MCM_GETTODAY 347
MCM_GETUNICODEFORMAT 348
MCM_HITTEST 349
MCM_SETCOLOR 351
MCM_SETCURSEL 352
MCM_SETDAYSTATE 353
MCM_SETFIRSTDAYOFWEEK 354
MCM_SETMAXSELCOUNT 354
MCM_SETMONTHDELTA 355
MCM_SETRANGE 356
MCM_SETSELRANGE 357
MCM_SETTODAY 358
MCM_SETUNICODEFORMAT 358
MCN_GETDAYSTATE 379
MCN_SELCHANGE 380
MCN_SELECT 380
MenuHelp 564
MonthCal_GetColor 359
MonthCal_GetCurSel 360
MonthCal_GetFirstDayOfWeek 361
MonthCal_GetMaxSelCount 362
MonthCal_GetMaxTodayWidth 363
MonthCal_GetMinReqRect 363
MonthCal_GetMonthDelta 364
MonthCal_GetMonthRange 365
MonthCal_GetRange 366
MonthCal_GetSelRange 367
MonthCal_GetToday 368
MonthCal_GetUnicodeFormat 368
MonthCal_HitTest 369
MonthCal_SetColor 370
MonthCal_SetCurSel 371
MonthCal_SetDayState 372
MonthCal_SetFirstDayOfWeek 373
MonthCal_SetMaxSelCount 374
MonthCal_SetMonthDelta 375
MonthCal_SetRange 376
MonthCal_SetSelRange 377
MonthCal_SetToday 377
MonthCal_SetUnicodeFormat 378
MONTHDAYSTATE 385

N

NM_CHAR 95
NM_CLICK 95
NM_CLICK (status bar) 578
NM_CLICK (tab) 639
NM_CUSTOMDRAW 117
NM_CUSTOMDRAW (header) 299
NM_CUSTOMDRAW (rebar) 535
NM_CUSTOMDRAW (Tooltip) 687
NM_CUSTOMDRAW (trackbar) 728
NM_DBLCLK 96
NM_DBLCLK (status bar) 579
NM_HOVER 96
NM_KEYDOWN 97
NM_KILLFOCUS 98
NM_KILLFOCUS (date time) 219
NM_NCHITTEST 98
NM_NCHITTEST (rebar) 536
NM_OUTOFMEMORY 99
NM_RCLICK 99
NM_RCLICK (header) 300
NM_RCLICK (status bar) 579
NM_RCLICK (tab) 639
NM_RDBLCLK 100
NM_RDBLCLK (status bar) 580
NM_RELEASEDCAPTURE 101
NM_RELEASEDCAPTURE (header) 301
NM_RELEASEDCAPTURE (monthcal) 381
NM_RELEASEDCAPTURE (pager) 408
NM_RELEASEDCAPTURE (rebar) 537
NM_RELEASEDCAPTURE (tab) 640
NM_RELEASEDCAPTURE (trackbar) 729
NM_RELEASEDCAPTURE (up-down) 746
NM_RETURN 101
NM_SETCURSOR 102
NM_SETCURSOR (ComboBoxEx) 157
NM_SETFOCUS 102
NM_SETFOCUS (date time) 219
NM_TOOLTIPSCREATED 103
NMCBEDRAGBEGIN 161
NMCBEENDEDIT 160
NMCHAR 105
NMCOMBOBOXEX 161
NMCUSTOMDRAW 119
NMDATETIMECHANGE 220
NMDATETIMEFORMAT 221
NMDATETIMEFORMATQUERY 222
NMDATETIMESTRING 223
NMDATETIMEWMKEYDOWN 224
NMDAYSTATE 384
NMHDDISPINFO 307

NMHDFILTERBTNCLICK Structure............ 308
NMHDR 106
NMHEADER... 309
NMIPADDRESS 329
NMKEY.. 107
NMMOUSE... 107
NMOBJECTNOTIFY 108
NMPGCALCSIZE....................................... 410
NMPGSCROLL 411
NMRBAUTOSIZE....................................... 544
NMREBAR... 545
NMREBARCHEVRON 546
NMREBARCHILDSIZE.............................. 547
NMSELCHANGE.. 384
NMTCKEYDOWN 644
NMTOOLTIPSCREATED........................... 109
NMTTCUSTOMDRAW............................... 691
NMTTDISPINFO 691
NMUPDOWN 747

P

Pager_ForwardMouse................................ 399
Pager_GetBkColor 399
Pager_GetBorder 400
Pager_GetButtonSize................................. 401
Pager_GetButtonState 401
Pager_GetDropTarget................................ 402
Pager_GetPos 403
Pager_RecalcSize....................................... 403
Pager_SetBkColor...................................... 404
Pager_SetBorder.. 405
Pager_SetButtonSize 406
Pager_SetChild 406
Pager_SetPos 407
PBM_DELTAPOS 417
PBM_GETPOS... 417
PBM_GETRANGE 418
PBM_SETBARCOLOR 419
PBM_SETBKCOLOR................................. 419
PBM_SETPOS 420
PBM_SETRANGE....................................... 421
PBM_SETRANGE32.................................. 421
PBM_SETSTEP 422
PBM_STEPIT 423
PBRANGE.. 423
PGM_FORWARDMOUSE 390
PGM_GETBKCOLOR 391
PGM_GETBORDER 391
PGM_GETBUTTONSIZE........................... 392
PGM_GETBUTTONSTATE 392
PGM_GETDROPTARGET......................... 393
PGM_GETPOS 394
PGM_RECALCSIZE................................... 395
PGM_SETBKCOLOR................................. 395
PGM_SETBORDER 396
PGM_SETBUTTONSIZE...........................396
PGM_SETCHILD 397
PGM_SETPOS 398
PGN_CALCSIZE 409
PGN_SCROLL..409
PropertySheet 438
PropSheet_AddPage 461
PropSheet_Apply 461
PropSheet_CancelToClose 462
PropSheet_Changed 463
PropSheet_GetCurrentPageHwnd 464
PropSheet_GetTabControl 465
PropSheet_HwndToIndex...........................465
PropSheet_IdToIndex 466
PropSheet_IndexToHwnd...........................467
PropSheet_IndexToId 467
PropSheet_IndexToPage 468
PropSheet_InsertPage 469
PropSheet_IsDialogMessage 470
PropSheet_PageToIndex 471
PropSheet_PressButton 472
PropSheet_QuerySiblings 473
PropSheet_RebootSystem 474
PropSheet_RemovePage 474
PropSheet_RestartWindows.......................475
PropSheet_SetCurSel 476
PropSheet_SetCurSelByID.........................477
PropSheet_SetFinishText...........................477
PropSheet_SetHeaderSubTitle 478
PropSheet_SetHeaderTitle.........................479
PropSheet_SetTitle....................................480
PropSheet_SetWizButtons 481
PropSheet_UnChanged..............................482
PROPSHEETHEADER...............................493
PROPSHEETPAGE....................................499
PropSheetPageProc 439
PropSheetProc 440
PSHNOTIFY 503
PSM_ADDPAGE 441
PSM_APPLY...442
PSM_CANCELTOCLOSE 442
PSM_CHANGED 443
PSM_GETCURRENTPAGEHWND............444
PSM_GETTABCONTROL 445
PSM_HWNDTOINDEX...............................445
PSM_IDTOINDEX 446
PSM_INDEXTOHWND...............................446
PSM_INDEXTOID 447
PSM_INDEXTOPAGE 447
PSM_INSERTPAGE...................................448
PSM_ISDIALOGMESSAGE 449
PSM_PAGETOINDEX 450
PSM_PRESSBUTTON 451
PSM_QUERYSIBLINGS.............................451

PSM_REBOOTSYSTEM 452
PSM_REMOVEPAGE 453
PSM_RESTARTWINDOWS 453
PSM_SETCURSEL 454
PSM_SETCURSELID 455
PSM_SETFINISHTEXT 456
PSM_SETHEADERSUBTITLE 456
PSM_SETHEADERTITLE 457
PSM_SETTITLE 458
PSM_SETWIZBUTTONS 459
PSM_UNCHANGED 460
PSN_APPLY 483
PSN_GETOBJECT 484
PSN_HELP 484
PSN_KILLACTIVE 485
PSN_QUERYCANCEL 486
PSN_QUERYINITIALFOCUS 487
PSN_RESET 488
PSN_SETACTIVE 489
PSN_TRANSLATEACCELERATOR 489
PSN_WIZBACK 490
PSN_WIZFINISH 491
PSN_WIZNEXT 492

R

RB_BEGINDRAG 510
RB_DELETEBAND 511
RB_DRAGMOVE 511
RB_ENDDRAG 512
RB_GETBANDBORDERS 512
RB_GETBANDCOUNT 513
RB_GETBANDINFO 514
RB_GETBARHEIGHT 515
RB_GETBARINFO 515
RB_GETBKCOLOR 516
RB_GETCOLORSCHEME 516
RB_GETDROPTARGET 517
RB_GETPALETTE 518
RB_GETRECT 518
RB_GETROWCOUNT 519
RB_GETROWHEIGHT 519
RB_GETTEXTCOLOR 520
RB_GETTOOLTIPS 520
RB_GETUNICODEFORMAT 521
RB_HITTEST 522
RB_IDTOINDEX 522
RB_INSERTBAND 523
RB_MAXIMIZEBAND 524
RB_MINIMIZEBAND 524
RB_MOVEBAND 525
RB_PUSHCHEVRON 526
RB_SETBANDINFO 527
RB_SETBARINFO 527
RB_SETBKCOLOR 528
RB_SETCOLORSCHEME 529
RB_SETPALETTE 529
RB_SETPARENT 530
RB_SETTEXTCOLOR 531
RB_SETTOOLTIPS 532
RB_SETUNICODEFORMAT 532
RB_SHOWBAND 533
RB_SIZETORECT 534
RBHITTESTINFO 548
RBN_AUTOSIZE 537
RBN_BEGINDRAG 538
RBN_CHEVRONPUSHED 539
RBN_CHILDSIZE 539
RBN_DELETEDBAND 540
RBN_DELETINGBAND 541
RBN_ENDDRAG 541
RBN_GETOBJECT 542
RBN_HEIGHTCHANGE 543
RBN_LAYOUTCHANGED 543
REBARBANDINFO 548
REBARINFO 552

S

SB_GETBORDERS 565
SB_GETICON 566
SB_GETPARTS 566
SB_GETRECT 567
SB_GETTEXT 567
SB_GETTEXTLENGTH 569
SB_GETTIPTEXT 570
SB_GETUNICODEFORMAT 570
SB_ISSIMPLE 571
SB_SETBKCOLOR 572
SB_SETICON 572
SB_SETMINHEIGHT 573
SB_SETPARTS 574
SB_SETTEXT 574
SB_SETTIPTEXT 575
SB_SETUNICODEFORMAT 576
SB_SIMPLE 577
SBN_SIMPLEMODECHANGE 580
SECOND_IPADDRESS 328
ShowHideMenuCtl 85

T

TabCtrl_AdjustRect 619
TabCtrl_DeleteAllItems 620
TabCtrl_DeleteItem 620
TabCtrl_DeselectAll 621
TabCtrl_GetCurFocus 622
TabCtrl_GetCurSel 622
TabCtrl_GetExtendedStyle 623

TabCtrl_GetImageList 623
TabCtrl_GetItem 624
TabCtrl_GetItemCount 625
TabCtrl_GetItemRect 625
TabCtrl_GetRowCount 626
TabCtrl_GetToolTips 627
TabCtrl_GetUnicodeFormat 627
TabCtrl_HighlightItem 628
TabCtrl_HitTest 629
TabCtrl_InsertItem 629
TabCtrl_RemoveImage 630
TabCtrl_SetCurFocus 631
TabCtrl_SetCurSel 632
TabCtrl_SetExtendedStyle 632
TabCtrl_SetImageList 633
TabCtrl_SetItem 634
TabCtrl_SetItemExtra 634
TabCtrl_SetItemSize 635
TabCtrl_SetMinTabWidth 636
TabCtrl_SetPadding 637
TabCtrl_SetToolTips 637
TabCtrl_SetUnicodeFormat 638
TBM_CLEARSEL 705
TBM_CLEARTICS 706
TBM_GETBUDDY 706
TBM_GETCHANNELRECT 707
TBM_GETLINESIZE 708
TBM_GETNUMTICS 708
TBM_GETPAGESIZE 709
TBM_GETPOS 710
TBM_GETPTICS 710
TBM_GETRANGEMAX 711
TBM_GETRANGEMIN 711
TBM_GETSELEND 712
TBM_GETSELSTART 713
TBM_GETTHUMBLENGTH 713
TBM_GETTHUMBRECT 714
TBM_GETTIC 715
TBM_GETTICPOS 715
TBM_GETTOOLTIPS 716
TBM_GETUNICODEFORMAT 716
TBM_SETBUDDY 717
TBM_SETLINESIZE 718
TBM_SETPAGESIZE 719
TBM_SETPOS 719
TBM_SETRANGE 720
TBM_SETRANGEMAX 721
TBM_SETRANGEMIN 722
TBM_SETSEL 722
TBM_SETSELEND 723
TBM_SETSELSTART 724
TBM_SETTHUMBLENGTH 725
TBM_SETTIC 725
TBM_SETTIPSIDE 726
TBM_SETTOOLTIPS 727
TCHITTESTINFO 644
TCITEM 645
TCITEMHEADER 647
TCM_ADJUSTRECT 601
TCM_DELETEALLITEMS 601
TCM_DELETEITEM 602
TCM_DESELECTALL 602
TCM_GETCURFOCUS 603
TCM_GETCURSEL 604
TCM_GETEXTENDEDSTYLE 604
TCM_GETIMAGELIST 605
TCM_GETITEM 605
TCM_GETITEMCOUNT 606
TCM_GETITEMRECT 606
TCM_GETROWCOUNT 607
TCM_GETTOOLTIPS 607
TCM_GETUNICODEFORMAT 608
TCM_HIGHLIGHTITEM 609
TCM_HITTEST 609
TCM_INSERTITEM 610
TCM_REMOVEIMAGE 611
TCM_SETCURFOCUS 611
TCM_SETCURSEL 612
TCM_SETEXTENDEDSTYLE 613
TCM_SETIMAGELIST 614
TCM_SETITEM 614
TCM_SETITEMEXTRA 615
TCM_SETITEMSIZE 616
TCM_SETMINTABWIDTH 616
TCM_SETPADDING 617
TCM_SETTOOLTIPS 617
TCM_SETUNICODEFORMAT 618
TCN_FOCUSCHANGE 640
TCN_GETOBJECT 641
TCN_KEYDOWN 642
TCN_SELCHANGE 642
TCN_SELCHANGING 643
THIRD_IPADDRESS 329
TOOLINFO 693
TTHITTESTINFO 695
TTM_ACTIVATE 666
TTM_ADDTOOL 666
TTM_ADJUSTRECT 667
TTM_DELTOOL 668
TTM_ENUMTOOLS 669
TTM_GETBUBBLESIZE 669
TTM_GETCURRENTTOOL 670
TTM_GETDELAYTIME 671
TTM_GETMARGIN 671
TTM_GETMAXTIPWIDTH 672
TTM_GETTEXT 673
TTM_GETTIPBKCOLOR 674
TTM_GETTOOLCOUNT 674
TTM_GETTOOLINFO 675
TTM_HITTEST 675

TTM_NEWTOOLRECT 676
TTM_POP 677
TTM_RELAYEVENT 677
TTM_SETDELAYTIME 678
TTM_SETMARGIN 679
TTM_SETMAXTIPWIDTH 680
TTM_SETTIPBKCOLOR 681
TTM_SETTIPTEXTCOLOR 681
TTM_SETTITLE 682
TTM_SETTOOLINFO 683
TTM_TRACKACTIVATE 683
TTM_TRACKPOSITION 684
TTM_UPDATE 685
TTM_UPDATETIPTEXT 686
TTM_WINDOWFROMPOINT 686
TTN_GETDISPINFO 688
TTN_POP 689
TTN_SHOW 690

U

UDACCEL 748
UDM_GETACCEL 737
UDM_GETBASE 738
UDM_GETBUDDY 738
UDM_GETPOS 738
UDM_GETRANGE 739
UDM_GETRANGE32 740
UDM_GETUNICODEFORMAT 740
UDM_SETACCEL 741
UDM_SETBASE 742
UDM_SETBUDDY 742
UDM_SETPOS 743
UDM_SETRANGE 743
UDM_SETRANGE32 744
UDM_SETUNICODEFORMAT 745
UDN_DELTAPOS 746
UninitializeFlatSB 249

W

WM_NOTIFY 90
WM_NOTIFYFORMAT 91

APPENDIX B

Index B: Volume 5, Elements Listed Alphabetically

A

ABM_ACTIVATE 721
ABM_GETAUTOHIDEBAR 721
ABM_GETSTATE 722
ABM_GETTASKBARPOS 723
ABM_NEW 723
ABM_QUERYPOS 724
ABM_REMOVE 724
ABM_SETAUTOHIDEBAR 725
ABM_SETPOS 726
ABM_WINDOWPOSCHANGED 726
ABN_FULLSCREENAPP 727
ABN_POSCHANGED 728
ABN_STATECHANGE 728
ABN_WINDOWARRANGE 729
AssocCreate 670
ASSOCDATA 561
ASSOCF 561
ASSOCKEY 563
AssocQueryKey 671
AssocQueryString 672
AssocQueryStringByKey 674
ASSOCSTR 563

B

BrowseCallbackProc 481

C

ChrCmpI 575
ColorAdjustLuma 707
ColorHLSToRGB 708
ColorRGBToHLS 708
CPL_DBLCLK 730
CPL_EXIT 730
CPL_GETCOUNT 731
CPL_INIT 732
CPL_INQUIRE 732
CPL_NEWINQUIRE 733
CPL_STARTWPARMS 735
CPL_STOP 736
CPlApplet 409

D

DefScreenSaverProc 410
DllGetVersion 411
DLLGETVERSIONPROC 412
DllInstall 710
DoEnvironmentSubst 413
DragAcceptFiles 414
DragFinish 415
DragQueryFile 416
DragQueryPoint 417

F

FindEnvironmentString 418
FindExecutable 419
FM_GETDRIVEINFO 736
FM_GETFILESEL 737
FM_GETFILESELLFN 738
FM_GETFOCUS 739
FM_GETSELCOUNT 739
FM_GETSELCOUNTLFN 740
FM_REFRESH_WINDOWS 740
FM_RELOAD_EXTENSIONS 741
FMEVENT_HELPMENUITEM 742
FMEVENT_HELPSTRING 742
FMEVENT_INITMENU 743
FMEVENT_LOAD 744
FMEVENT_SELCHANGE 745
FMEVENT_TOOLBARLOAD 745
FMEVENT_UNLOAD 746
FMEVENT_USER_REFRESH 746
FMExtensionProc 483
FOLDERFLAGS 564
FOLDERVIEWMODE 566

G

GetMenuContextHelpId 420
GetWindowContextHelpId 420

H

HashData ... 711

I

IACList
  Expand ... 141
IACList2
  GetOptions ... 143
  SetOptions ... 143
IActiveDesktop
  AddDesktopItem Method ... 145
  AddDesktopItemWithUI Method ... 146
  AddUrl Method ... 148
  ApplyChanges ... 149
  GenerateDesktopItemHtml ... 150
  GetDesktopItem ... 150
  GetDesktopItemByID ... 151
  GetDesktopItemBySource ... 152
  GetPattern ... 153
  GetDesktopItemCount ... 153
  GetDesktopItemOptions ... 154
  GetWallpaper ... 154
  GetWallpaperOptions ... 155
  ModifyDesktopItem ... 156
  RemoveDesktopItem ... 157
  SetDesktopItemOptions ... 157
  SetPattern ... 158
  SetWallpaper ... 159
  SetWallpaperOptions ... 159
IASyncOperation
  EndOperation ... 161
  GetAsyncMode ... 162
  InOperation ... 163
  SetAsyncMode ... 163
  StartOperation ... 164
IAutoComplete
  Enable ... 167
  Init ... 168
IAutoComplete2
  GetOptions ... 170
  SetOptions ... 171
IColumnProvider
  GetColumnInfo ... 174
  GetItemData ... 175
  Initialize ... 176
ICommDlgBrowser
  IncludeObject ... 177
  OnDefaultCommand ... 178
  OnStateChange ... 178
ICommDlgBrowser2
  GetDefaultMenuText ... 180
  GetViewFlags ... 181
  Notify ... 182
IContextMenu
  GetCommandString ... 183
  InvokeCommand ... 185
  QueryContextMenu ... 186
IContextMenu2
  HandleMenuMsg ... 189
IContextMenu3
  HandleMenuMsg2 ... 191
ICopyHook
  CopyCallback ... 193
ICurrentWorkingDirectory
  GetDirectory ... 195
  SetDirectory ... 196
IDeskBand
  GetBandInfo ... 197
IDockingWindow
  CloseDW ... 199
  ResizeBorderDW ... 199
  ShowDW ... 201
IDockingWindowFrame
  AddToolbar ... 202
  FindToolbar ... 203
  RemoveToolbar ... 204
IDockingWindowSite
  GetBorderDW ... 214
  RequestBorderSpaceDW ... 215
  SetBorderSpaceDW ... 215
IDragSourceHelper
  InitializeFromBitmap ... 206
  InitializeFromWindow ... 207
IDropTargetHelper
  DragEnter ... 209
  DragLeave ... 210
  DragOver ... 210
  Drop ... 211
  Show ... 212
IEmptyVolumeCache
  Deactivate ... 217
  GetSpaceUsed ... 218
  Initialize ... 219
  Purge ... 221
  ShowProperties ... 222
IEmptyVolumeCache2
  InitializeEx ... 224
IEmptyVolumeCacheCallback
  PurgeProgress ... 227
  ScanProgress ... 228
IEnumExtraSearch
  Clone ... 229
  Next ... 230
  Reset ... 231
  Skip ... 231
IEnumIDList
  Clone ... 233
  Next ... 233

Reset 235
Skip 235
IExtractIcon
Extract 237
GetIconLocation 238
IExtractImage
Extract 241
GetLocation 241
IExtractImage2
GetDateStamp................................ 244
IFileViewer
PrintTo 245
Show.. 246
ShowInitialize.................................. 247
IFileViewerSite
GetPinnedWindow........................... 248
SetPinnedWindow 249
IInputObject
HasFocusIO.................................... 250
TranslateAcceleratorIO 251
UIActivateIO 251
IInputObjectSite
OnFocusChangeIS.......................... 253
InetIsOffline 421
INewShortcutHook
GetExtension.................................. 254
GetFolder.. 255
GetName 256
GetReferent..................................... 256
SetFolder.. 257
SetReferent 258
INotifyReplica
YouAreAReplica 259
IntlStrEqN 576
IntlStrEqNI 577
IntlStrEqWorker.. 578
IObjMgr
Append 260
Remove 261
IPersistFileSystemFolder
GetFolderTargetInfo........................ 265
InitializeEx 266
IPersistFolder
Initialize... 262
IPersistFolder2
GetCurFolder.................................. 263
IProgressDialog
HasUserCancelled 269
SetAnimation 269
SetCancelMsg 270
SetLine 271
SetProgress.................................... 272
SetProgress64................................ 273
SetTitle 274
StartProgressDialog 274
StopProgressDialog.........................276
Timer 276
IQueryAssociations
GetData...279
GetEnum...280
GetKey 280
GetString..281
Init 282
IQueryInfo
GetInfoFlags 284
GetInfoTip 285
IReconcilableObject
GetProgressFeedbackMax
Estimate 286
Reconcile 287
IReconcileInitiator
SetAbortCallback 292
SetProgressFeedback 293
IRemoteComputer
Initialize 294
IResolveShellLink
ResolveShellLink 296
IRunnableTask
IsRunning..298
Kill 299
Resume...299
Run..300
Suspend..300
IShellBrowser
BrowseObject..................................302
EnableModelessSB 304
GetControlWindow...........................304
GetViewStateStream 306
InsertMenusSB 307
OnViewWindowActive......................308
QueryActiveShellView 309
RemoveMenusSB............................310
SendControlMsg 311
SetMenuSB.....................................312
SetStatusTextSB.............................313
SetToolbarItems..............................314
TranslateAcceleratorSB...................315
IShellChangeNotify
OnChange.......................................316
IShellDetails
ColumnClick....................................319
GetDetailsOf 320
IShellExecuteHook
Execute 323
IShellExtInit
Initialize 324
IShellFolder
BindToObject 327
BindToStorage.................................328
CompareIDs....................................329

CreateViewObject ... 331
EnumObjects ... 332
GetAttributesOf ... 333
GetDisplayNameOf ... 335
GetUIObjectOf ... 337
ParseDisplayName ... 338
SetNameOf ... 342
IShellFolder2
EnumSearches ... 344
GetDefaultColumn ... 345
GetDefaultColumnState ... 346
GetDefaultSearchGUID ... 347
GetDetailsEx ... 347
GetDetailsOf ... 348
MapNameToSCID ... 349
IShellIcon
GetIconOf ... 351
IShellIconOverlay
GetOverlayIconIndex ... 353
GetOverlayIndex ... 354
IShellIconOverlayIdentifier
GetOverlayInfo ... 356
GetPriority ... 357
IsMemberOf ... 358
IShellLink
GetArguments ... 360
GetDescription ... 361
GetHotkey ... 361
GetIconLocation ... 362
GetIDList ... 363
GetPath ... 364
GetShowCmd ... 365
GetWorkingDirectory ... 366
Resolve ... 366
SetArguments ... 368
SetDescription ... 369
SetHotkey ... 370
SetIconLocation ... 371
SetIDList ... 371
SetPath ... 372
SetRelativePath ... 373
SetShowCmd ... 374
SetWorkingDirectory ... 375
IShellLinkDataList
AddDataBlock ... 376
CopyDataBlock ... 377
GetFlags ... 378
RemoveDataBlock ... 379
SetFlags ... 379
IShellPropSheetExt
AddPages ... 381
ReplacePage ... 382
IShellView
AddPropertySheetPages ... 384
CreateViewWindow ... 385
DestroyViewWindow ... 387
EnableModeless ... 387
EnableModelessSV ... 388
GetCurrentInfo ... 388
GetItemObject ... 389
Refresh ... 390
SaveViewState ... 391
SelectItem ... 392
TranslateAccelerator ... 393
UIActivate ... 394
IShellView2
CreateViewWindow2 ... 396
GetView ... 397
HandleRename ... 397
SelectAndPositionItem ... 398
ITaskbarList
ActivateTab ... 400
AddTab ... 400
DeleteTab ... 401
HrInit ... 402
SetActiveAlt ... 402
IUniformResourceLocator
GetURL ... 403
InvokeCommand ... 405
SetURL ... 406
IURL_SETURL_FLAGS ... 566
IURL_SETURL_INVOKECOMMAND_FLAGS ... 567
IURLSearchHook
Translate ... 407

M

MAKEDLLVERULL ... 571
MIMEAssociationDialog ... 421
MLLoadLibrary ... 579

P

PathAddBackslash ... 610
PathAddExtension ... 610
PathAppend ... 611
PathBuildRoot ... 612
PathCanonicalize ... 613
PathCombine ... 614
PathCommonPrefix ... 615
PathCompactPath ... 615
PathCompactPathEx ... 616
PathCreateFromUrl ... 617
PathFileExists ... 618
PathFindExtension ... 619
PathFindFileName ... 620
PathFindNextComponent ... 620
PathFindOnPath ... 621

PathFindSuffixArray 622
PathGetArgs 623
PathGetCharType 623
PathGetDriveNumber 624
PathIsContentType 625
PathIsDirectory 625
PathIsDirectoryEmpty 626
PathIsFileSpec 627
PathIsHTMLFile 627
PathIsLFNFileSpec 628
PathIsNetworkPath 629
PathIsPrefix 630
PathIsRelative 630
PathIsRoot 631
PathIsSameRoot 632
PathIsSystemFolder 632
PathIsUNC 633
PathIsUNCServer 634
PathIsUNCServerShare 634
PathIsURL 635
PathMakePretty 636
PathMakeSystemFolder 636
PathMatchSpec 637
PathParseIconLocation 638
PathQuoteSpaces 639
PathRelativePathTo 639
PathRemoveArgs 641
PathRemoveBackslash 641
PathRemoveBlanks 642
PathRemoveExtension 642
PathRemoveFileSpec 643
PathRenameExtension 644
PathSearchAndQualify 644
PathSetDlgItemPath 645
PathSkipRoot 646
PathStripPath 647
PathStripToRoot 647
PathUndecorate 648
PathUnExpandEnvStrings 649
PathUnmakeSystemFolder 650
PathUnquoteSpaces 651

R

RegisterDialogClasses 423
REGSAM 669

S

ScreenSaverConfigureDialog 424
ScreenSaverProc 425
SetMenuContextHelpId 426
SetWindowContextHelpId 427
SHAddToRecentDocs 428
SHAppBarMessage 429
SHAutoComplete 712
SHBindToParent 430
SHBrowseForFolder 431
SHChangeNotify 432
SHCONTF 568
SHCopyKey 675
SHCreateDirectoryEx 437
SHCreateProcessAsUser 438
SHCreateShellPalette 709
SHCreateStreamOnFile 714
SHCreateThread 714
SHDeleteEmptyKey 676
SHDeleteKey 677
SHDeleteValue 678
Shell_NotifyIcon 439
ShellAbout 441
ShellExecute 442
ShellExecuteEx 445
SHEmptyRecycleBin 447
SHEnumKeyEx 679
SHEnumValue 680
SHFileOperation 448
SHFreeNameMappings 449
SHGetDataFromIDList 450
SHGetDesktopFolder 451
SHGetDiskFreeSpace 452
SHGetFileInfo 453
SHGetFolderLocation 457
SHGetFolderPath 458
SHGetIconOverlayIndex 461
SHGetInstanceExplorer 462
SHGetMalloc 463
SHGetNewLinkInfo 464
SHGetPathFromIDList 466
SHGetSettings 466
SHGetSpecialFolderLocation 468
SHGetSpecialFolderPath 469
SHGetThreadRef 716
SHGetValue 681
SHGNO 569
SHInvokePrinterCommand 470
SHLoadInProc 472
SHOpenRegStream 717
SHOpenRegStream2 718
SHQueryInfoKey 683
SHQueryRecycleBin 473
SHQueryValueEx 684
SHRegCloseUSKey 685
SHRegCreateUSKey 686
SHREGDEL_FLAGS 705
SHRegDeleteEmptyUSKey 687
SHRegDeleteUSValue 688
SHRegDuplicateHKey 689
SHREGENUM_FLAGS 706

SHRegEnumUSKey 690
SHRegEnumUSValue 691
SHRegGetBoolUSValue 692
SHRegGetPath 693
SHRegGetUSValue 694
SHRegOpenUSKey 696
SHRegQueryInfoUSKey 697
SHRegQueryUSValue 698
SHRegSetPath 700
SHRegSetUSValue 701
SHRegWriteUSValue 702
SHSetThreadRef 719
SHSetValue 704
SHStrDup 580
SOANGLETENTHS 573
SoftwareUpdateMessageBox 473
SOPALETTEINDEX 573
SOPALETTERGB 573
SORGB 574
SOSETRATIO 574
StrCat 581
StrCatBuff 581
StrChr 582
StrChrI 583
StrCmp 584
StrCmpI 585
StrCmpN 585
StrCmpNI 586
StrCpy 587
StrCpyN 588
StrCSpn 589
StrCSpnI 590
StrDup 591
StrFormatByteSize 592
StrFormatByteSize64A 593
StrFormatKBSize 594
StrFromTimeInterval 595
StrIsIntlEqual 596
StrNCat 597
StrPBrk 598
StrRChr 598
StrRChrI 599
StrRetToBuf 600
StrRetToStr 601
StrRStrI 602
StrSpn 603
StrStr 604
StrStrI 604
StrToInt 605
StrToIntEx 606
StrTrim 607

T

TranslateURL 475
TRANSLATEURL_IN_FLAGS 570

U

UndeleteFile 484
UrlApplyScheme 651
URLAssociationDialog 476
URLASSOCIATIONDIALOG_IN_FLAGS 571
UrlCanonicalize 653
UrlCombine 654
UrlCompare 655
UrlCreateFromPath 656
UrlEscape 657
UrlEscapeSpaces 658
UrlGetLocation 659
UrlGetPart 660
UrlHash 661
UrlIs 662
UrlIsFileUrl 663
UrlIsNoHistory 664
UrlIsOpaque 665
UrlUnEscape 666
UrlUnEscapeInPlace 667

W

WinHelp 477
WM_CPL_LAUNCH 747
WM_CPL_LAUNCHED 747
WM_DROPFILES 748
WM_HELP 749
WM_TCARD 749
wnsprintf 608
wvnsprintf 609

Microsoft® Windows® Shell

This essential Windows 2000 and Windows 98/ Windows 95 reference volume is part of the five-volume Microsoft Win32® Developer's Reference Library. *In its printed form, this material is portable, easy to use, and easy to browse—a highly condensed, completely indexed, intelligently organized complement to the information available on line and through the Microsoft Developer Network (MSDN). Each volume includes an overview of the five-volume library, two appendixes of programming elements, and tips on how and where to find other Microsoft developer reference resources you may need.*

Microsoft Windows Shell

This volume provides complete reference materials about Windows shell programming, including the new Shell Programmer's Guide, completely revamped for Windows 2000. Included in this volume are shell basics and intermediate techniques, the extensive set of Windows Shell interfaces available to Win32 programmers, as well as functions, macros, lightweight utility APIs, messages and notifications, structures, and enumerations.

Part No. 097-0002310

Microsoft