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® Common Controls

BASED ON msdn™ library

Microsoft®

The essential reference to Win32® technologies and APIs

David Iseminger
Series Editor

Microsoft® Windows® Common Controls

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.

Acquisitions Editor: Ben Ryan
Project Editor: Wendy Zucker

Part No. 097-0002309

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

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 (Graphical 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 4 of the *Microsoft Win32 Developer's Reference Library* focuses on common controls that Windows applications developers use throughout the course of the development process. This volume—*Volume 4: Common Controls*—provides the reference material necessary for developers to take advantage of the wealth of ready-made common controls found in Windows.

When programming with these common controls, programmers must be prepared to deal with versioning issues that are associated with common control programming. Almost all of the common controls are contained within three .dll files (Comctl32.dll, Shell32.dll, and Shlwapi.dll), and all of these .dll files have versioning issues that must be kept in check throughout the development process. The Windows shell shares the versioning requirements of common controls, so when you're programming with either common controls (explained in this volume of the Win32 Library) or the Windows shell (explained in Volume 5), you must deal with the versioning requirements.

What are the versioning requirements, you ask? The introduction to Part 2 of this volume (and Volume 5 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 common controls detailed in this volume of the Win32 Library.

Once you've read the introduction to Part 2 and understand the versioning issues you'll need to address during development, you can jump into the common controls reference material found in Part 2. The list of common controls is long, and often the controls aren't necessarily grouped into sensible collections. Rather than forcing them into groups, I'm providing the somewhat long list here. Fortunately, the names of many of the common controls are reasonably self-explanatory. For more information about any of these given controls, jump to the table of contents and find the control's chapter in Part 2 of this book (hint: the chapters in Part 2 are in the same order as this list), and take a look at the introductory/explanatory information provided in the chapter associated with the common control you're interested in.

Win32 common controls include:

- Using Common Controls
- Common API
- Customizing a Control's Appearance
- Animation Controls
- ComboBoxEx Controls
- Creating Wizards
- Date and Time Picker Controls
- Drag List Boxes
- Flat Scroll Bars
- Header Controls
- Hot Key Controls
- Image Lists
- IP Address Controls
- Month Calendar Controls
- Pager Controls
- Progress Bar Controls
- Property Sheets
- Rebar Controls
- Status Bars
- Tab Controls
- Tooltip Controls
- Trackbar Controls
- Up-Down Controls

Part 2 of this volume goes into detail about each of these common controls individually; in fact, each item in this list corresponds to an individual chapter in Part 2 of this volume of the Win32 Library. But remember, you should read the introduction to Part 2 (found at the beginning of Part 2, which is a great place for introductions) to learn about the versioning issues you'll have to deal with when programming with these common controls.

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.

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.

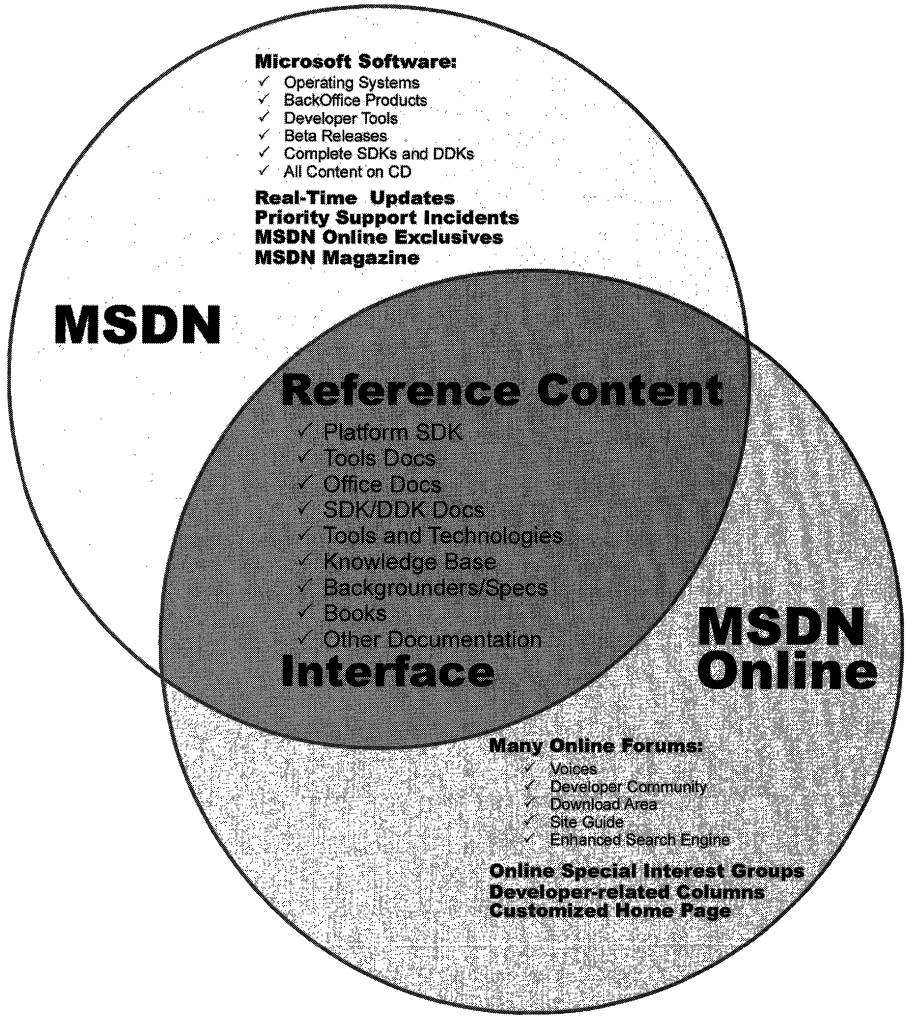


Figure 3-1: The similarities and differences in coverage between MSDN and MSDN Online.

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.

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.

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.

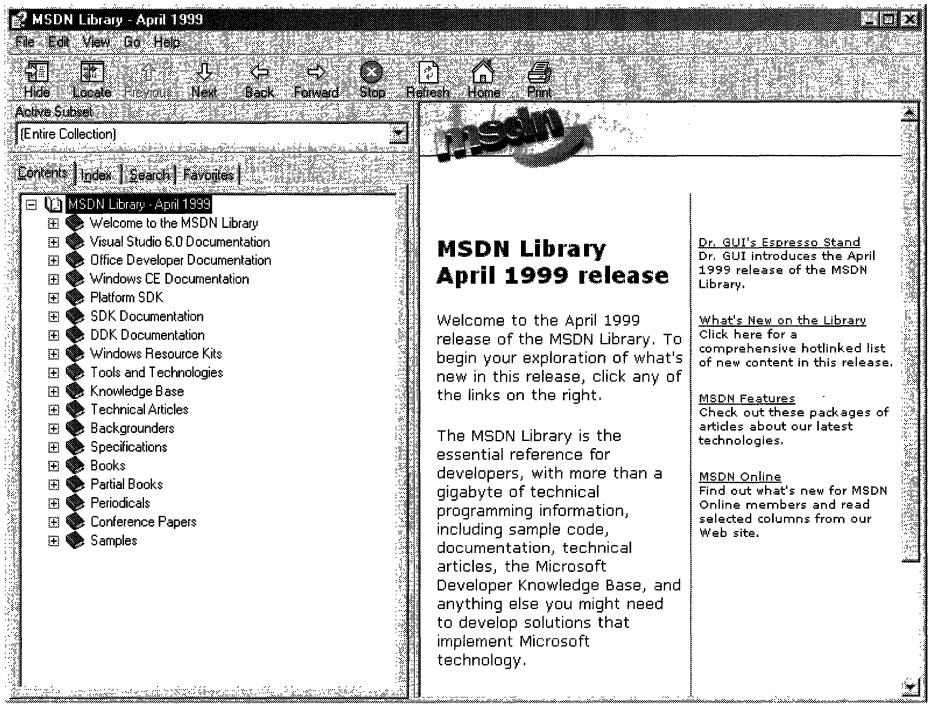


Figure 3-2: The MSDN interface.

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 ServicesPlatform 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

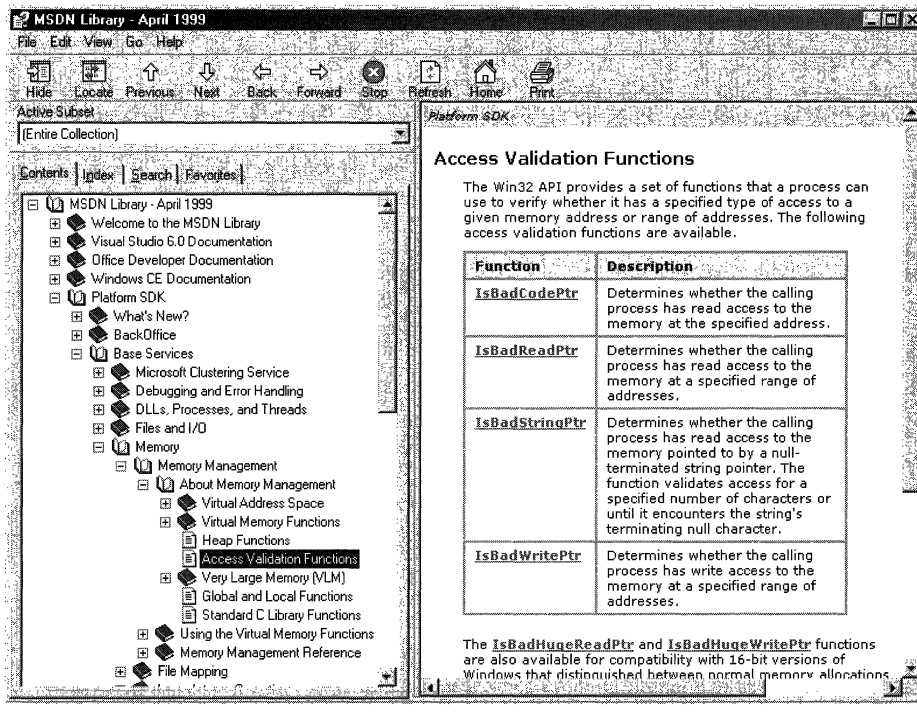


Figure 3-3: Basic navigation through MSDN.

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.

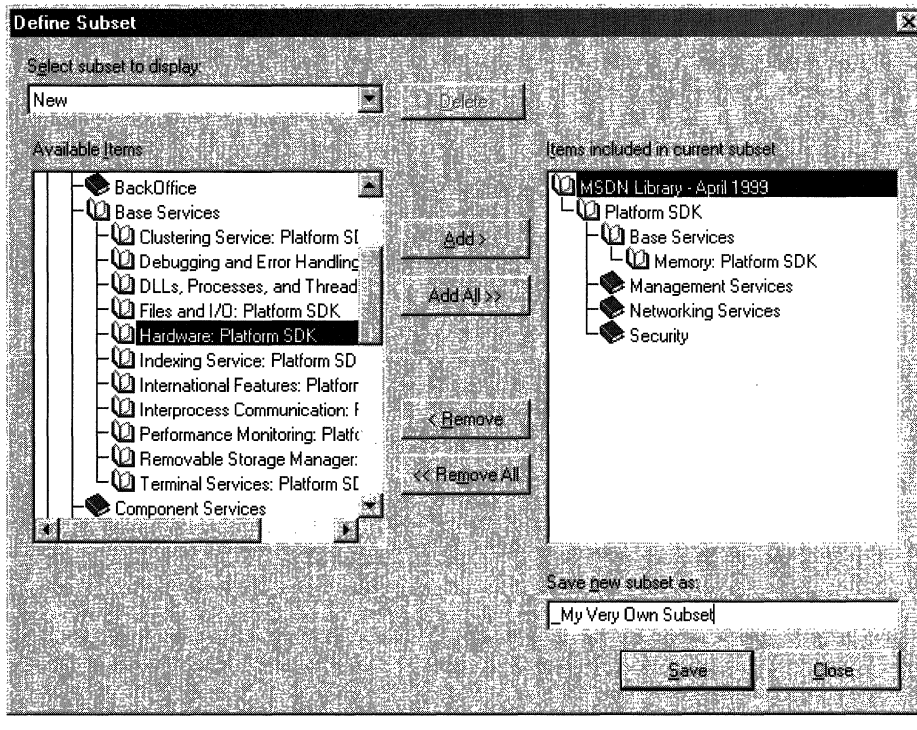


Figure 3-4: The Define Subsets window.

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.

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 on 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.

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	Voices
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*.

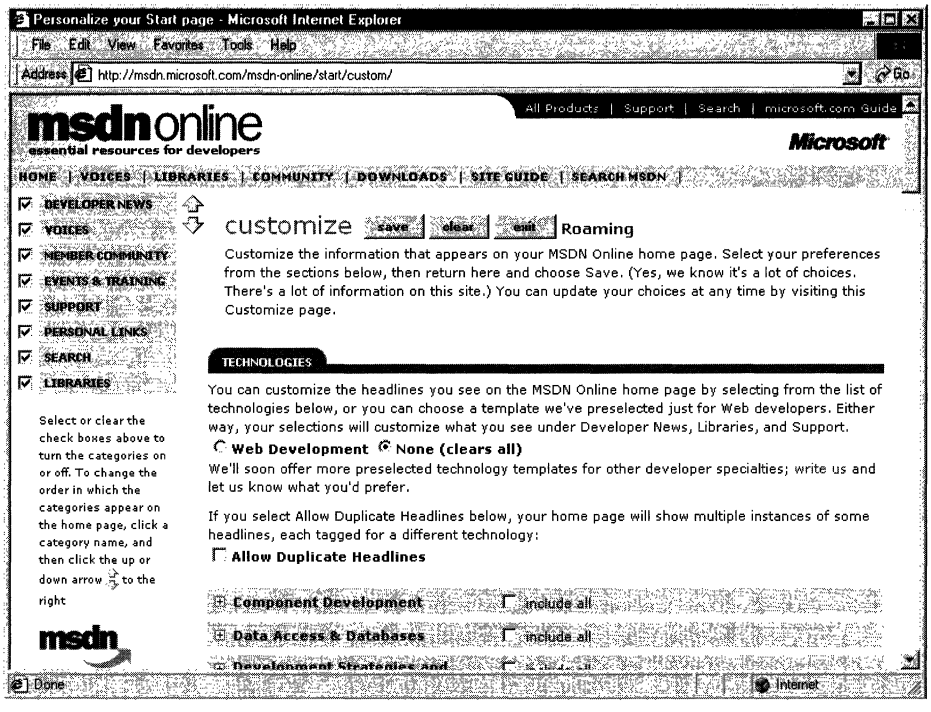


Figure 3-5: The MSDN Online configuration page.

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—include:

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.

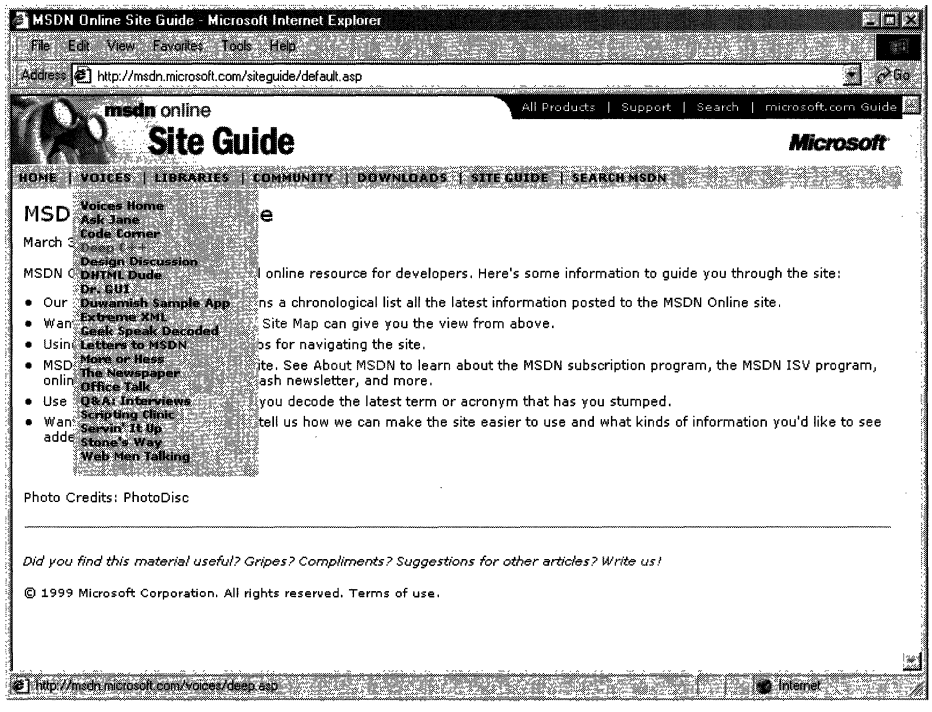


Figure 3-6: The MSDN Online Navigation bar with its drop-down menus.

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.

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*.

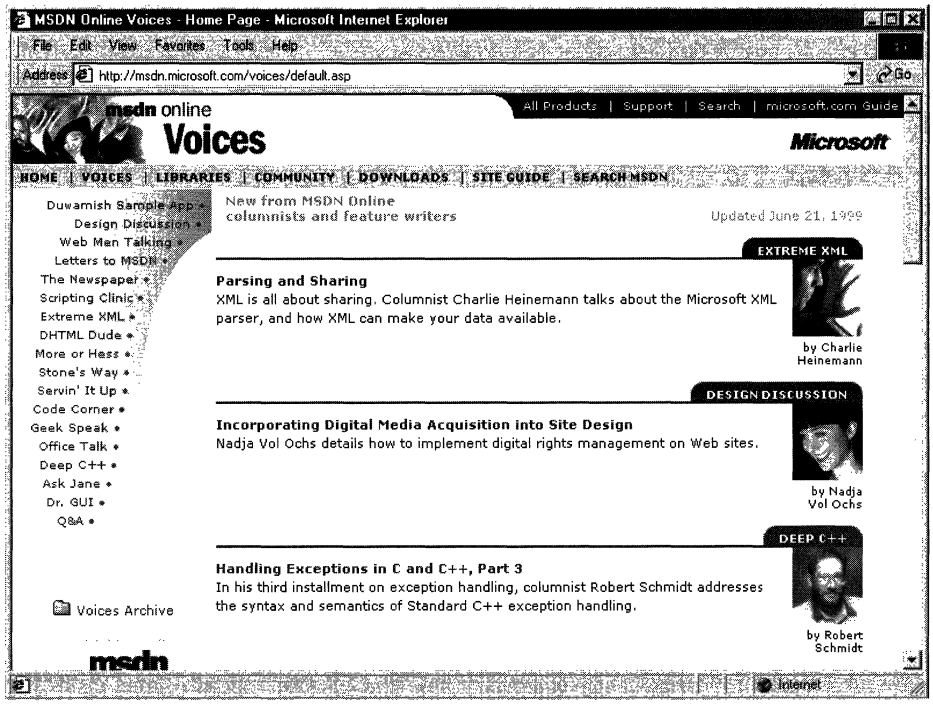


Figure 3-7: The Voices home page.

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 **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.

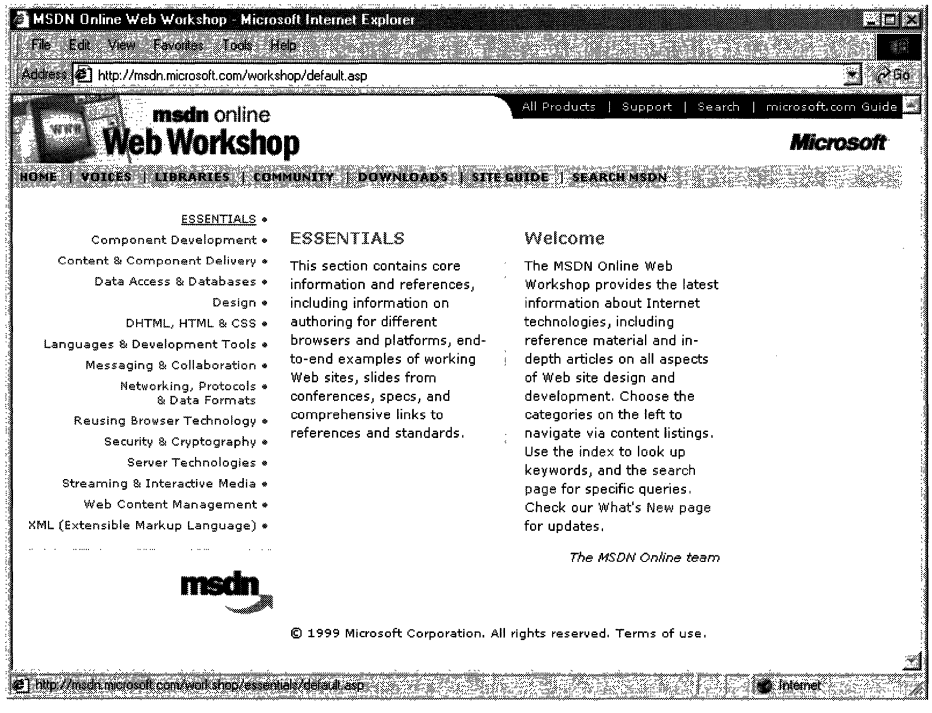


Figure 3-8: The Web Workshop home page, with its bulleted list of navigation start points.

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*.

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 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.

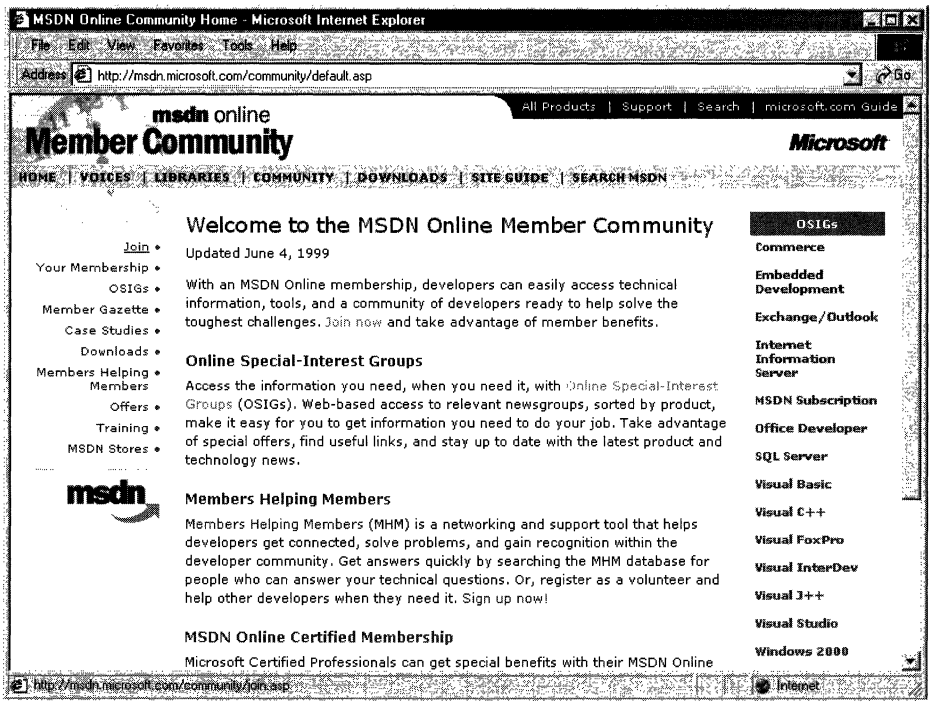


Figure 3-9: The Community home page.

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.

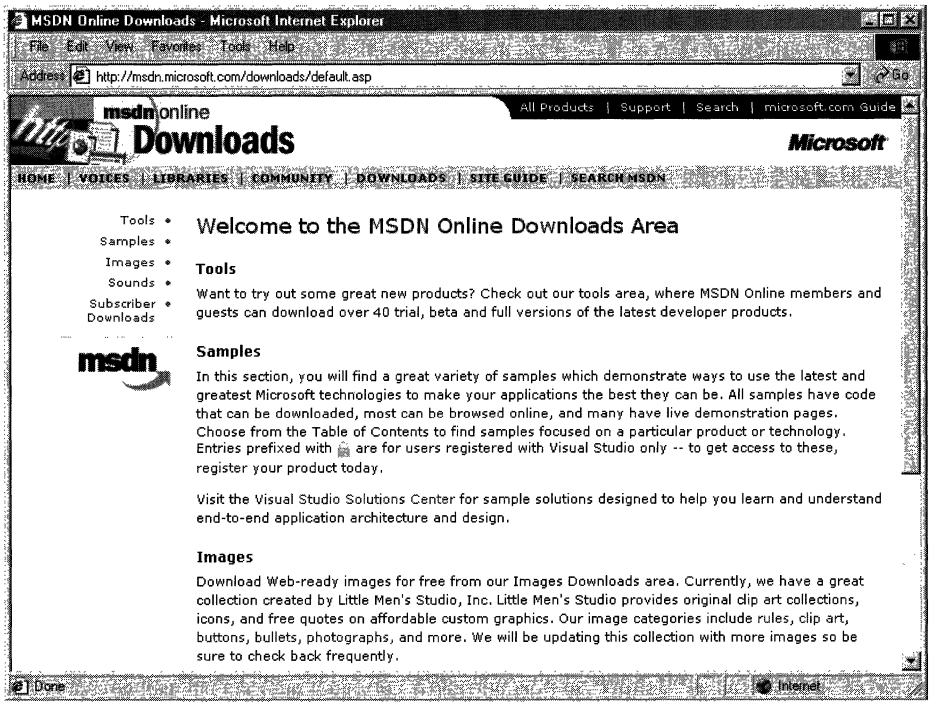


Figure 3-10: The Downloads home page.

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

Win32 Developer's Programming Reference 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

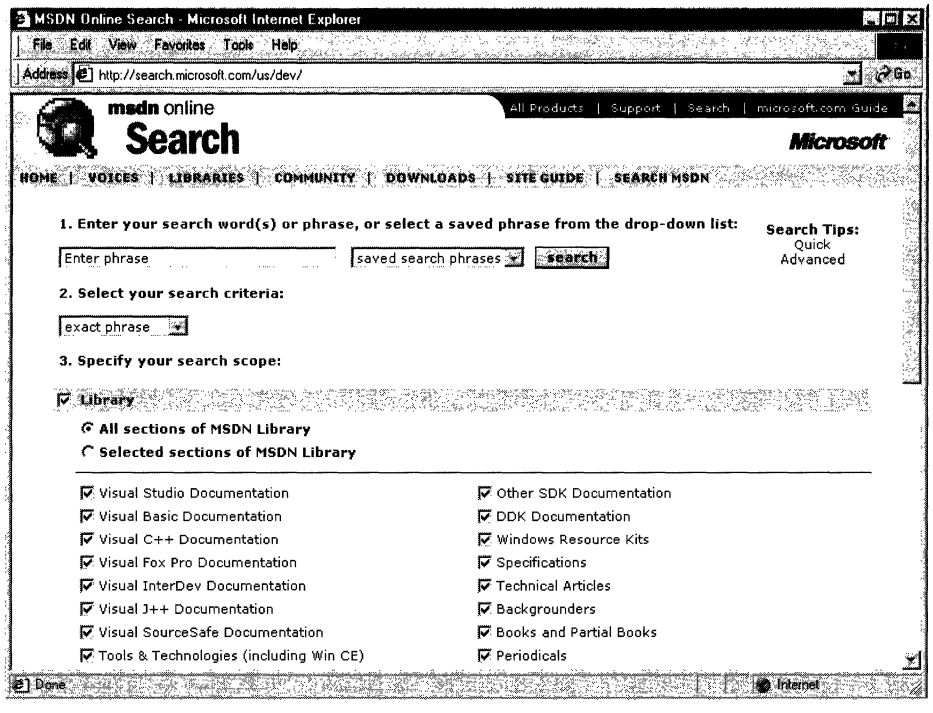


Figure 3-11: The Search home page.

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.

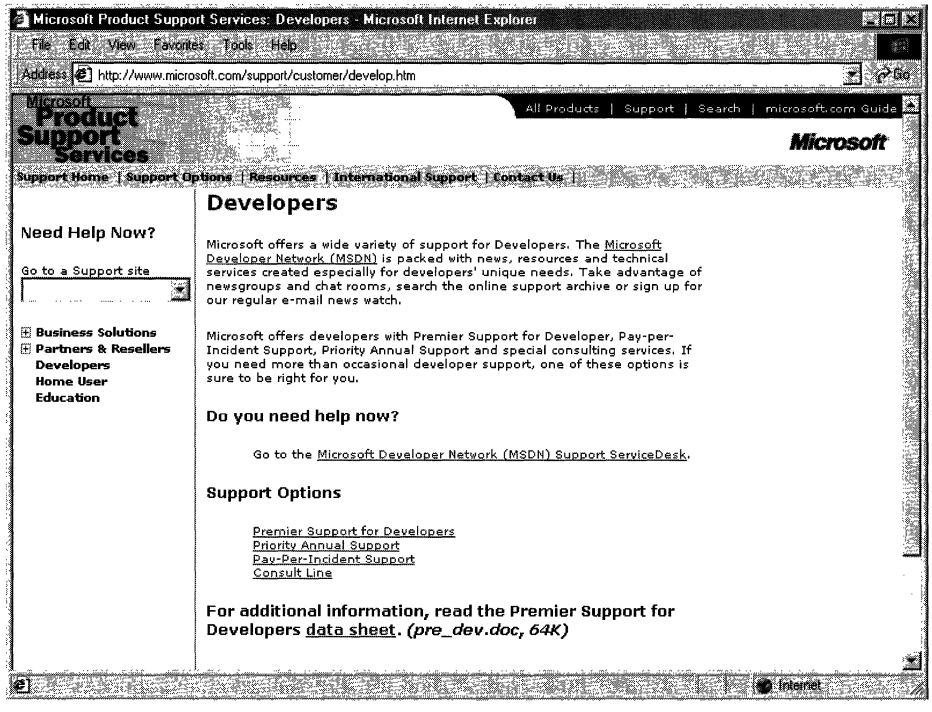


Figure 4-1: The Product Services Support page for developers.

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 Technologies: Part 4

This chapter is the fourth 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 4, and the errors and examples found in this chapter provide insights that can help you avoid problems with buffer overflows and an assortment of miscellaneous errors in your programming projects. So without further ado, here they are!

Buffer Overflows

Buffer overflows can cause all sorts of problems and can be the result of simple errors on the part of the developer or the result of a directed attack. Avoiding buffer overflow problems isn't difficult, but failing to do so can result in dire consequences. Follow the rules listed below, and their subsequent explanations, to avoid such problems.

- Always check the actual buffer size when accessing a buffer, rather than some known maximum.
- Be aware of arithmetic overflow, and ensure that checks don't go wrong because of them.
- Verify that arithmetic performed on enumerated types results in values within the enumeration.
- Test buffers sizes against expectations; don't assume they have been tested already.

You can also take the following additional precautions to avoid buffer overflows:

- When using an offset address, ensure that the location is not beyond either end of the buffer.
- On complex size calculations, ensure that the total size is greater than the fixed header.
- Beware of strings without **NULL** termination. If there is a size, use it!
- Check minimum and maximum values of enums after calculation.
- When comparing external and internal data, compare sizes first, and then use the minimum for comparisons.

Simple Buffer Overflow

The best solution to simple buffer flow is to check the bounds of a buffer before referencing it. However, there are a couple of cases that require extra attention.

One case that requires extra care is writing data to stack buffers; going beyond the bounds of a stack buffer can allow the return address to be set to an arbitrary value, resulting in execution of arbitrary code. Another case that requires extra care is non-terminated strings; in many cases—such as kernel mode and network structures—strings are sent with a size, but no **NULL** terminator. In these cases, do not rely on a **NULL** terminator, but rather use the size.

The general rule is that it's important to always check buffer accesses by their actual upper and lower bounds, and not to do so by a known minimum or maximum.

Example

```
NTSTATUS
AppBug(IN UNICODE_STRING *String)
{
    NTSTATUS Status;
    UNICODE_STRING CapturedString;
    WCHAR *Buffer = NULL;

    try {
        ProbeForRead(String,
                     sizeof (UNICODE_STRING),
                     4);
        CapturedString = *String;
        ProbeForRead(CapturedString.Buffer,
                     CapturedString.Length,
                     sizeof (WCHAR));
        Buffer = ExAllocatePoolWithQuotaTag(PagedPool,
                                            CapturedString.Length,
                                            ' guB');
```

```
        RtlCopyMemory(Buffer,
                      CapturedString.Buffer,
                      CapturedString.Length);
        Status = InternalFoo(Buffer);
        ExFreePool(Buffer);
    } except (EXCEPTION_EXECUTE_HANDLER) {
        if (Buffer) {
            ExFreePool(Buffer);
        }

        Status = GetExceptionCode();
    }
    return Status;
}
```

Remarks

In this example we passed the captured buffer to the internal function with neither an indication of how large the string actually is nor a zero terminating it.

Size Overflow or Underflow

In many instances, especially in network code, buffers are passed that have a fixed header and a variable tail. These types of buffers often lead to complex size calculations that require careful validation. The most common way these buffers break is when a large (effectively negative) size is provided in the variable-length section, such that the sum of header plus tail is less than the buffer size. Validation succeeds, resulting in a huge section of memory from the tail being copied into a too-small buffer. Another common attack is to send a partial packet that is shorter than the header section. Slight rearrangements of comparisons will often correct the problem.

Example

```
typedef struct _INFO {
    ULONG FlagBits;
    ULONG DataSize;
    UCHAR Data[1];
} INFO, *PINFO;

NTSTATUS
AppGetInfo(
    PINFO Info,
    ULONG Len
)
{
    if (Len < (ULONG)FIELD_OFFSET(INFO, Data)) {
        return STATUS_INFO_LENGTH_MISMATCH;
    }
```

(continued)

(continued)

```
    if (Len < (ULONG)FIELD_OFFSET(INFO, Data) + Info->DataSize){
        return STATUS_INFO_LENGTH_MISMATCH;
    }
    [...]
}
```

Remarks

The problem with the above code sample is that the addition in the second **if** statement may overflow. That overflow would cause the test to succeed even though the buffer isn't big enough to contain that much data. It's easy to rearrange the above **if** statement to get it working correctly:

```
if (Len - (ULONG)FIELD_OFFSET(INFO, Data) < Info->DataSize) {
    return STATUS_INFO_LENGTH_MISMATCH;
}
```

Remarks

We can do the first subtraction without underflow occurring because of the presence of the first **if** test in the earlier code.

Abuse of enumerated types

Enumerated types have a limited range of values, which means that some operations (most notably addition and subtraction) might yield values that will cause invalid memory references. Enumerated types should be checked for minimum and maximum after performing any arithmetic operation.

Example

```
typedef enum {
    FirstEntry,
    SecondEntry
} INFOCLASS;

INFOCLASS InformationClass;
[...]
Length = InfoLengths[InformationClass - 1];
```

Remarks

Consider `InformationClass == FirstEntry`, which evaluates to zero.

Using internal lengths for comparisons to external input

Some application components maintain internal length values for structures used by the component. This is fine so long as the data is internal to the application component; however, a problem arises when application-internal lengths are used with external input data. If lengths need not be identical, use the minimum of the internal and external

lengths. If lengths should be identical, a mismatch between the two sizes should cause the parameter to be rejected.

Example

```
case IOCTL_CRASH_SYSTEM:
    //
    // IOCTL_CRASH_SYSTEM is METHOD_BUFFERED, so the
    // buffer we get from the I/O system is the max of
    // the input and output buffer sizes.
    //
    Buffer = pIrp -> AssociatedIrp.SystemBuffer;
    RtlCopyMemory(Buffer,
                  GlobalSource,
                  sizeof GlobalSource);
    pIrp -> IoStatus.Information = sizeof GlobalSource;
```

Remarks

If the **RtlCopyMemory()** function call in this code example doesn't crash the system for writing too much data, the I/O system code to copy this data back into user mode might. **OutputLength** just might have been zero.

Miscellaneous Errors

This section functions as a catch-all for problems that are general enough to occur in any application code, but unusual enough not to fit into a simple category. The following list enumerates miscellaneous issues that developers should be wary of when developing applications:

- Be careful when casting input data to another type.
- Double-check precedence order in complex expressions.
- Ensure that all parts of the compound conditional are equivalent (each result should execute the same code) or are special-cased where appropriate.
- Check all pointer parameters for **NULL** (especially optional parameters).
- Don't hard-code strings in code (for example, "Administrators").
- Beware of multiple checks of volatile data.
- Always acquire locks in a consistent order.
- Beware of (and preferably eliminate or reduce) inconsistencies with common interfaces (for example, **GetLastError** and functions returning handles).

Dangers of typecasting

Casting an input value to another type without sufficient checks can lead to a number of problems. One common faulty assumption (that pointers are 4 bytes long) could lead to significant problems if an application is ported to a 64-bit operating system. Casting input

data to floating-point types can have even more significant repercussions, because many bit combinations are not valid floating-point values. Finally, casting to different types can give very different behavior if the sign bit is set depending on whether the types are signed; unsigned values are zero-extended, while signed values are sign-extended.

In general, the best approach to typecasting in your applications is to make as few assumptions as possible. Casts from pointers to longs might chop the value, making it look good even though it's bad. Any float passed from user mode should be assumed to contain all possible bit patterns, not just properly formed floating-point values.

Example

```
NTSTATUS BufferCheck(
    BUFFER InputBuffer,
    USHORT InputBufferLength
)
{
    USHORT AddressLength;
    if (InputBufferLength > (USHORT)(InputBuffer->DataOffset +
        InputBuffer->DataLength)) {
        //
        //  The data buffer checks out OK.  Get the
        //  length of the target address (first
        //  USHORT after the data buffer).
        //
        AddressLength = *(USHORT *) ((PCHAR)InputBuffer +
            InputBuffer->DataOffset + InputBuffer->DataLength));
[…]
```

Remarks

In this example, **DataOffset** and **DataLength** are **ULONG** values whose sum has been cast to a **USHORT** for comparison to the **USHORT** value *InputBufferLength*. Because the values are truncated, it's possible to succeed on this conditional and still dereference far beyond the scope of the indicated buffer when **AddressLength** is retrieved, because the variables in question are not recast.

Operator precedence

Many common problems occur because of a misunderstanding of the precedence rules—most commonly **&** and **|** versus **==** and **!=**. Equality has higher precedence, but code is often written **if (a & c2 == c2)**, which is really **if (a & (c1 == c2))**. To avoid this problem, fully parenthesize the intended order of operations, or look it up to verify that precedence is correct.

Example

```
if (Value & CONSTANT == CONSTANT) {
    RetVal = ERROR_INVALID_PARAMETER;
}
```

Remarks

Thus **if (Value & !0)**. There's a PERL script called **TYPO** that can find these easily. A smart compiler will optimize both statements away; a smarter one will generate warnings.

Conditional termination confusion

Conditional termination confusion occurs when a compound condition is used and subsequent code assumes that one particular clause was satisfied. This particular programming error is frequently discovered in **while** and **for** loops with compound termination clauses.

Example

```
while (index < BufferLength && Buffer[index] != '\0'){
    index++;
}
StringLength = strlen(Buffer);

[…]
```

Remarks

The loop in this example seems to be attempting to check that the buffer is properly **NULL**-terminated without overflowing the end of the buffer; however, the statement immediately following assumes that the terminator was found, and thus the second condition fulfilled the **while** loop termination. If the first clause fulfilled the termination condition, the **strlen** call would read past the specified length in the buffer.

Misuse of OPTIONAL parameters

OPTIONAL parameters can be **NULL**. However, some functions dereference **OPTIONAL** parameters without verifying that they are non-**NULL**, or check for **NULL** in some paths without checking others. Avoiding this common programming error is simple: Check all **OPTIONAL** parameters for **NULL** before using them.

Example

```
BOOL
DoubleDup(
    char *StringSrcA,
    char *StringSrcB,
    char *StringDstA OPTIONAL,
```

(continued)

(continued)

```
    DWORD *pLenA,
    char *StringDstB OPTIONAL,
    DWORD *pLenB)
{
    if ((StringSrcA == NULL) || (StringSrcB == NULL)) {
        return FALSE;
    }

    if (StringDstA == NULL) {
        *pLenA=strlen(StringSrcA) + 1;
        *pLenB=strlen(StringSrcB) + 1;

    } else {
        strncpy(StringDstA,
                StringSrcA
                *pLenA);
        strncpy(StringDstB,
                StringSrcB
                *pLenB);
    }

    return TRUE;
}
```

Remarks

Consider the case where **StringDstA != NULL** and **StringDstB == NULL**.

Return value confusion and inconsistencies

The Win32 API includes several features that are expected to be general to all system API functions, but in reality, they are not. The two most commonly misused features are **INVALID_HANDLE_VALUE** and **GetLastError**. **GetLastError** can be called after most Win32 API functions, but there are some functions (registry API functions, for example) that don't call **SetLastError**. Similarly, the **Nt*** API functions don't call **SetLastError**. **INVALID_HANDLE_VALUE** is returned only from Win32 file-system API functions (**CreateFile, FindFirstFile**, and so on). Passing **INVALID_HANDLE_VALUE** to the **GetKernelObjectSecurity()** function will return the security on the current process because **INVALID_HANDLE_VALUE == GetCurrentProcess()**.

To avoid these common programming errors, carefully check the appropriate error return code. Some Win32 handle functions return **NULL** to indicate an error, while others use **INVALID_HANDLE_VALUE**. **GetLastError()** inconsistencies are problems that should be fixed, but be sure to use return codes for error checking, not just **GetLastError()**.

Example

```
HANDLE
OpenFile(
    …
)
{
    NtOpenFile(…);
    if (GetLastError() == STATUS_SUCCESS) {
        //
        //  Open succeeded. Now read the data.
        //
[…]
```

Remarks

The call to the **NtOpenFile()** function does not call the **SetLastError()** function, so the call to **GetLastError()** returns the result from the last call that did call **SetLastError()**. If the last call that called **SetLastError()** succeeded, a *false* positive response may result.

Don't rely on volatile objects

Any multithreaded environment can run into synchronization problems if global data is checked multiple times expecting the same result. If a kernel or network server makes decisions based on multiple checks of a volatile object, special care must be taken to ensure that different values for the object will not break the algorithm. The best way to avoid this problem is to avoid doing the same queries more than once. If multiple queries are required, make sure that differing results don't cause a problem. For example, if access to a file is determined to be possible, don't assume that further accesses to the same file by name will also succeed.

Example

```
NTSTATUS
ReadWriteFileByName(
    PCHAR  FileName,
    BUFFER ReadBuffer,
    BUFFER WriteBuffer
)
{
    HANDLE FileHandle;
    //
    //  Check that the calling user has
    //  sufficient privileges.
    //
    if (!UserHasReadWritePrivileges(FileName)) {
        return(STATUS_ACCESS_DENIED);
    }
```

(continued)

(continued)

```
    FileHandle = OpenForRead(FileName);
    […Code to read data from the file into the supplied buffer…]
    CloseHandle(FileHandle);
    if (WriteBuffer != NULL) {
        //
        //  A write buffer was supplied; reopen the
        //  file and write the indicated data.
        //
        FileHandle = OpenForWrite(FileName);
[…]
```

Results

The privileges on the file in this example might have changed in the time between the *read* open and the *write* open. Because this is a privileged component and no impersonation was performed, this code may end up writing data to a file that has since been marked read-only to the user in question.

Avoid spinlock order problems

Spinlocks (or any other locking/mutex mechanisms) acquired in the wrong order create timing windows that can deadlock computers. Many components are particularly susceptible to this in their IRP cancel routines, where spinlocks may be acquired without dropping the implicitly acquired **CancelSpinlock**. To avoid this situation, always acquire spinlocks in a consistent order, even when one is implicitly acquired.

Example

```
VOID
LockingFunction1(PSESSION    Session)
{
    PCONNECTION Connection;
    KIRQL       OldIrql;
    KeAcquireSpinLock(&Session->Lock, &OldIrql);
    Connection = Session->Connection;
    if (Connection->Session != Session) {
        KeReleaseSpinLock(&Session->Lock, OldIrql);
        return;
    }
    KeAcquireSpinLockAtDpcLevel(&Connection->Lock);
    […]
    KeReleaseSpinLockFromDpcLevel(&Connection->Lock);
    KeReleaseSpinLock(&Session->Lock, OldIrql);
    return;
}
```

```
VOID
LockingFunction2(PCONNECTION Connection)
{
    PSESSION Session;
    KIRQL    OldIrql;
    KeAcquireSpinLock(&Connection->Lock, &OldIrql);
    Session = Connection->Session;
    if (Session == NULL) {
        KeReleaseSpinLock(&Connection->Lock, OldIrql);
        return;
    }
    KeAcquireSpinLockAtDpcLevel(&Session->Lock);
[…]
```

Remarks

If **LockingFunction1()** and **LockingFunction2()** were to acquire the session and connection locks respectively at nearly the same time, both threads would deadlock waiting for the other to release the lock that they next attempt to acquire.

Determining membership in Administrators group

Many applications check whether a user is an administrator before allowing an operation, but determining group membership is often performed incorrectly. The most common method for determining membership in the Administrators group is to build the appropriate SID and look in the user's token for that SID. With "restricted" tokens, however, this is no longer sufficient. Another common method was to look up that SID by specifying the name "Administrators"; that approach is not localizable, and therefore not the best approach. The best approach is to use **CheckTokenMembership()** to check a user's membership in any group.

Example

```
//
// See if this user is an Administrator.
//
BOOL IsMember = FALSE;
TCHAR SidBuffer[128], RefDom[128];
ULONG AdminSidSize = 128 * sizeof (TCHAR);
ULONG RefDomSize = 128 * sizeof (TCHAR);
ULONG Size = 0;
NTSTATUS Status;
PSID AdminSid = (PSID)SidBuffer;
PSID_NAME_USE NameUse;

if (LookupAccountName(NULL,
                      TEXT("Administrator"),
```

(continued)

(continued)

```
                    &AdminSid,
                    &AdminSidSize,
                    &RefDom,
                    &RefDomSize,
                    &NameUse)) {
    if (!GetTokenInformation(Token,
                        TokenGroups,
                        NULL,
                        0,
                        &Size)) {
        Groups = RtlAllocateHeap(RtlProcessHeap(),
                            0,
                            Size);
        if (Groups) {
            if (GetTokenInformation(Token,
                                TokenGroups,
                                Groups,
                                Size,
                                &Size)) {
                for (i = 0; i < Groups->GroupCount; i++) {
                    if (RtlEqualSid(AdminSid,
                                Groups->Groups[i].Sid) {
                        *IsAdmin = TRUE;
                        break;
                    }
                }
            }
            RtlFreeHeap(RtlProcessHeap(),
                        0,
                        Groups);
        }
    }
}
```

To fix the problem in this code sample, convert the above code to the following:

```
BOOL IsMember;
PSID AdminSid;
PSID_IDENTIFIER_AUTHORITY NtAuthority = SECURITY_NT_AUTHORITY;

if (AllocateAndInitializeSid(&NtAuthority,
                        2,
                        SECURITY_BUILTIN_DOMAIN_RID,
                        DOMAIN_ALIAS_RID_ADMINS,
                        0, 0, 0, 0, 0, 0,
```

```
                                &AdminSid)) {
    if (!CheckTokenMembership(Token,
                                AdminSid,
                                &IsMember)) {
        IsMember = FALSE;
    }
    GlobalFree(AdminSid);
}
```

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.

Buffer Overflows

1. Simple buffer overflow: Always check actual buffer size when accessing a buffer, rather than some known maximum.
2. Size overflow or underflow: When using an offset address, ensure that the location is not beyond either end of the buffer.
3. Abuse of enumerated types: On complex size calculations, ensure that total size is greater than the fixed header.
4. Using internal lengths for comparisons to external input: Beware of strings without **NULL** termination. If there is a size, use it!

Miscellaneous Errors

1. Dangers of typecasting: Be careful when casting input data to another type.
2. Operator precedence: Double-check precedence order in complex expressions.
3. Conditional termination confusion: Ensure that all clauses of a compound conditional are equivalent (each result should execute the same code), or are special-cased where appropriate.
4. Misuse of **OPTIONAL** parameters: Check all pointer parameters for **NULL** (especially optional parameters).
5. Return value confusion and inconsistencies: Don't hard-code strings in code (for example, "Administrators").
6. Don't rely on volatile objects: Beware of multiple checks of volatile data.
7. Avoid spinlock order problems: Always acquire locks in a consistent order.
8. Determining membership in Administrators group: Beware of (and preferably eliminate or reduce) inconsistencies with common interfaces (for example, GetLastError and functions returning handles).

PART 2

Introduction

The common controls are an important part of the user interface in virtually all Microsoft products, as well as in many applications produced by third-party developers. However, there are a lot of controls, and finding the information you need to perform certain common or important tasks is not always easy. This chapter is designed to make you aware of versioning considerations you must take into account during the development process, as well as to highlight several important tasks that are of importance to developers. The individual sections are task-oriented and designed to provide you with enough information about the procedure, so that you will be able to implement controls in your application with a minimum of fuss.

Getting Information About List-View, Toolbar, and Tree-View Controls

In order to adhere to the mission of the Windows Programming Reference Series—which is to provide concise, compact, and portable reference books—three common controls were omitted from this printed volume: List-View, Toolbar, and Tree-View Controls. Together, these three controls are approximately as long as the book you are holding currently, so in order to provide you with the most comprehensive and useful collection, these three controls have been provided in a more compact form—that is, they have been provided on the companion CD.

The companion CD found in the Base Services volume contains all the information about these three controls (along with all the other controls that did get into this volume, and loads of other reference information). If you have not done so, you should fire up the installation CD and get the electronic reference companion CD installed on your computer.

General Introduction to the Common Controls

The *common controls* are a set of windows that are implemented by the common control library, which is a dynamic-link library (DLL) included with the Microsoft Windows operating system. Like other control windows, a common control is a child window that an application uses in conjunction with another window to perform I/O tasks.

Using Common Controls

Most common controls belong to a window class defined in the common control DLL. The window class and the corresponding window procedure define the properties, appearance, and behavior of the control. To ensure that the common control DLL is

loaded, include the **InitCommonControlsEx** function in your application. You create a common control by specifying either the name of the window class when calling the **CreateWindowEx** function or the appropriate class name in a dialog-box template.

DLL Versions

All 32-bit versions of Windows include a common controls DLL, known as Comctl32.dll. However, this DLL has been updated several times since it was first introduced. Each successive version supports the features and application programming interface (API) of earlier versions. However, each new version also contains a number of new features and a correspondingly larger API. Applications must be aware of which version of Comctl32.dll is installed on a system, and use only the features and API that the DLL supports.

Because new versions of the common controls were distributed with Internet Explorer, the version of Commctl32.dll that is present is commonly different from the version that was shipped with the operating system. It might be several versions more recent, actually. Thus, it is not enough for your application to know which operating system it is running on—it must determine directly which version of Comctl32.dll is present. For a detailed discussion of common controls versions and how to determine which version of Comctl32.dll is installed, see *Shell and Common Controls Versions*.

Structure sizes for different common control versions

Ongoing enhancements to common controls have resulted in the need to extend many of the structures. This, in turn, results in the size of the structures changing between different versions of Commctrl.h. Because most of the common control structures take a structure size as one of the parameters, this can result in a message or function failing if the size is not recognized. To remedy this, structure-size constants have been defined to aid in targeting different versions of Comctl32.dll. The following list defines the new structure-size constants:

Control	Constant
HDITEM_V1_SIZE	The size of the **HDITEM** structure in version 4.00.
LVCOLUMN_V1_SIZE	The size of the **LVCOLUMN** structure in version 4.00.
LVHITTESTINFO_V1_SIZE	The size of the **LVHITTESTINFO** structure in version 4.00.
LVITEM_V1_SIZE	The size of the **LVITEM** structure in version 4.00.
NMLVCUSTOMDRAW_V3_SIZE	The size of the **NMLVCUSTOMDRAW** structure in version 4.70.
NMTTDISPINFO_V1_SIZE	The size of the **NMTTDISPINFO** structure in version 4.00.
NMTVCUSTOMDRAW_V3_SIZE	The size of the **NMTVCUSTOMDRAW** structure in version 4.70.

Control	Constant
PROPSHEETHEADER_V1_SIZE	The size of the **PROPSHEETHEADER** structure in version 4.00.
PROPSHEETPAGE_V1_SIZE	The size of the **PROPSHEETPAGE** structure in version 4.00.
REBARBANDINFO_V3_SIZE	The size of the **REBARBANDINFO** structure in version 4.70.
TTTOOLINFO_V1_SIZE	The size of the **TOOLINFO** structure in version 4.00.
TVINSERTSTRUCT_V1_SIZE	The size of the **TVINSERTSTRUCT** structure in version 4.00.

Common Control Styles

Each type of common control has a set of control styles that you can use to vary the appearance and behavior of the control. The common control library also includes a set of control styles that apply to two or more types of common controls. The common control styles are described in the *Common Control Styles* section.

Common Control Messages

Because common controls are windows, an application can manipulate them by using messages, such as **WM_GETFONT** or **WM_SETTEXT**. In addition, the window class of each common control supports a set of control-specific messages that an application can use to manipulate the control. An application can use any of the message sending or posting functions to pass messages to the control. In addition, some common controls have a set of macros that an application can use instead of the sending or posting functions. The macros are typically easier to use than the functions.

When a change is made to the system color settings, Windows sends a **WM_SYSCOLORCHANGE** message to all top-level windows. Your top-level window must forward the **WM_SYSCOLORCHANGE** message to its common controls; otherwise, the controls will not be notified of the color change. This ensures that the colors used by your common controls are consistent with those used by other user interface objects. For example, a toolbar control uses the 3D Objects color to draw its buttons. If the user changes the 3D Objects color but the **WM_SYSCOLORCHANGE** message is not forwarded to the toolbar, the toolbar buttons will remain in their original color (or even change to a combination of old and new colors) while the color of other buttons in the system changes.

Common Control Notification Messages

Common controls are child windows that send notification messages to the parent window when events, such as input from the user, occur in the control. The application relies on these notification messages to determine what action the user wants it to take. Except for trackbars, which use the **WM_HSCROLL** and **WM_VSCROLL** messages to notify its parent of changes, common controls send notification messages as

WM_NOTIFY messages. The *lParam* parameter of **WM_NOTIFY** is either the address of an **NMHDR** structure or the address of a larger structure that includes **NMHDR** as its first member. The structure contains the notification code and identifies the common control that sent the notification message. The meaning of the remaining structure members, if any, varies depending on the notification code.

Common controls notifications support both ANSI and UNICODE formats. The system determines which format to use by sending your window a **WM_NOTIFYFORMAT** message. To specify a format, return NFR_ANSI for ANSI notifications, and NFR_UNICODE for Unicode notifications. If you do not handle this message, the system calls **IsWindowUnicode** to determine the format. Since Windows 95 and Windows 98 always return FALSE to this function call, they use ANSI notifications by default.

Note Not all controls will send **WM_NOTIFY** messages. In particular, the standard Windows controls (edit controls, combo boxes, list boxes, buttons, scroll bars, and static controls) do not send **WM_NOTIFY** messages. Consult the documentation for the control to determine if it will send any **WM_NOTIFY** messages and, if it does, which notification codes it will send.

Each type of common control has a corresponding set of notification codes. The common control library also provides notification codes that can be sent by more than one type of common control. See the documentation for the control of interest to determine which notification codes it will send and what format they take.

Common Control Updates in Internet Explorer

Common controls in Internet Explorer support the following new features.

Common Control Initialization

The common controls are now initialized with the **InitCommonControlsEx** function. This function allows you to specify which controls should be initialized for your application instead of initializing all of the controls. The **InitCommonControls** function is still supported, but new applications should use **InitCommonControlsEx**.

New Common Control Styles

There are four new common control styles defined. These are **CCS_LEFT**, **CCS_RIGHT**, **CCS_VERT**, and **CCS_NOMOVEX**. For more information, see *Common Control Styles*.

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 1: 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 might have different version numbers, and test each one separately.

Note 2: 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 3: 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 Windows 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

Using Common Controls

Creating a Customizable Toolbar

Most Microsoft Windows applications use toolbar controls to provide their users with convenient access to various tools. However, static toolbars have some shortcomings, such as too little space to effectively display all the available tools.

The solution to this problem is to make your application's toolbars customizable. Users can then move, add, and delete tools to select only the ones they need and organize them in whatever way they find convenient.

To enable customization, include the **CCS_ADJUSTABLE** common controls style flag when you create the toolbar control. There are two basic approaches to customization:

- The customization dialog box. This system-provided dialog box is the simplest approach. It gives users a graphic user interface that allows them to add, delete, or move icons.
- Dragging and dropping tools. Implementing drag-and-drop allows users to move tools to another location on the toolbar or delete them by dragging them off the toolbar. It provides users a quick and easy way to organize their toolbar, but does not allow them to add tools.

You can implement either or both, depending on the needs of the application.

Neither of these two approaches to customization provides a built-in mechanism, such as a **Cancel** or **Undo** button, to return the toolbar to its former state. You must explicitly use the toolbar control API to store the toolbar's precustomization state. If necessary, you can later use this stored information to restore the toolbar to its original state.

This document discusses how to enable toolbar customization with the customization dialog box and with drag-and-drop. It also briefly discusses saving and restoring a toolbar's state.

The Customization Dialog Box

The customization dialog box is provided by the toolbar control to give users a simple way to add, move, or delete tools. Users can launch it by double-clicking the toolbar. Applications can launch the customization dialog box by sending the toolbar control a **TB_CUSTOMIZE** message. Figure 6-1 shows an example of the toolbar customization dialog box.

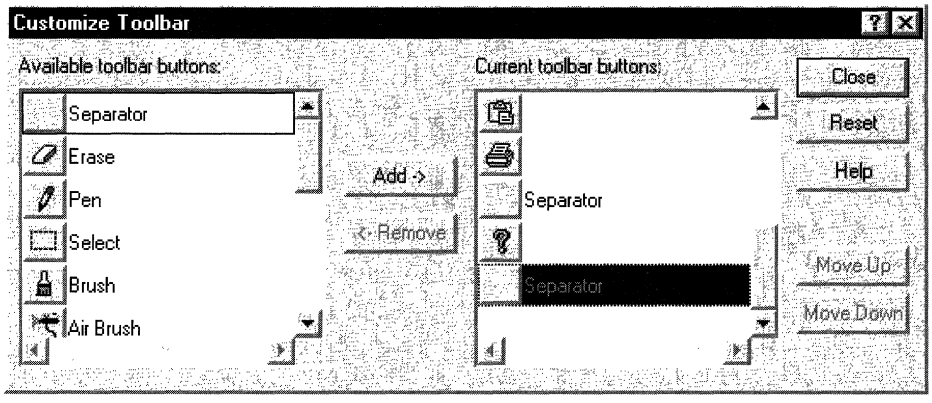


Figure 6-1: The toolbar customization dialog box.

The tools in the right-hand list box are those currently on the toolbar. Initially, this list will contain the tools that you specify when you create the toolbar. The left-hand list box contains the tools that are available to add to the toolbar. Your application is responsible for populating that list and keeping track of what tools are currently on the toolbar.

Implementing the Customization Dialog Box

The toolbar control notifies your application that it is launching a customization dialog box by sending its parent window a **TBN_BEGINADJUST** notification. It then sends a **TBN_INITCUSTOMIZE** notification. If you don't want the toolbar to display a **Help** button, handle this notification and return TBNRF_HIDEHELP.

The toolbar control then collects the information it needs to initialize the dialog box by sending three series of notifications in the following order:

1. A **TBN_QUERYINSERT** notification for each button on the toolbar to determine where buttons can be inserted. Return FALSE to prevent a button from being inserted to the left of the button specified in the notification. If you return FALSE to all TBN_QUERYINSERT notifications, the dialog box will not be displayed.
2. A **TBN_QUERYDELETE** notification for each tool currently on the toolbar. Return TRUE if a tool can be deleted, or FALSE if not. If all your tools can be deleted, you do not need to handle this notification.
3. A series of **TBN_GETBUTTONINFO** notifications to populate the list of available tools. To add a tool to the list, fill in the **NMTOOLBAR** structure that is passed with the notification and return TRUE. When you have no more tools to add, return FALSE.

The dialog box is then displayed, and users can begin to customize the toolbar.

Once the dialog box is displayed, your application can receive a variety of notifications, depending on the users' actions:

- **TBN_QUERYINSERT**. Each time the user changes the location of a tool on the toolbar, or adds a tool. Return FALSE to prevent the tool from being inserted at that location.
- **TBN_DELETINGBUTTON**. The user is about to remove a tool from the toolbar.
- **TBN_CUSTHELP**. The user has clicked the **Help** button.
- **TBN_TOOLBARCHANGE**. The user has added, moved, or deleted a tool.
- **TBN_RESET**. The user has clicked the **Reset** button.

After the dialog box is destroyed, your application will receive a **TBN_ENDADJUST** notification.

Dragging and Dropping Tools

Users also can rearrange the buttons on a toolbar by pressing the SHIFT key and dragging the button to another location. The drag-and-drop process is handled automatically by the toolbar control. It displays a ghost image of the button as it is dragged, and rearranges the toolbar after it is dropped. Users cannot add buttons in this way, but they can delete a button by dropping it off the toolbar.

Although the toolbar control normally does this operation automatically, it also sends your application two notifications: **TBN_QUERYDELETE** and **TBN_QUERYINSERT**. To control the drag-and-drop process, handle these notifications as follows:

- The **TBN_QUERYDELETE** notification is sent as soon as the user attempts to move the button, before the ghost button is displayed. Return FALSE to prevent the button from being moved. If you return TRUE, users will be able to either move the tool or delete it by dropping it off the toolbar. Once you have allowed users to move a tool, you cannot prevent them from deleting it. However, if users delete a tool, the toolbar control will send your application a **TBN_DELETINGBUTTON** notification.
- The **TBN_QUERYINSERT** notification is sent when the user attempts to drop the button on the toolbar. To prevent the button being moved from being dropped to the left of the button specified in the notification, return FALSE. This notification is not sent if the user drops the tool off the toolbar.

If the user attempts to drag a button without also pressing the SHIFT key, the toolbar control will not handle the drag-and-drop operation. However, it will send your application a **TBN_BEGINDRAG** notification to indicate the start of a drag operation, and a **TBN_ENDDRAG** notification to indicate the end. If you want to enable this form of drag-and-drop, your application must handle these notifications, provide the necessary user interface, and modify the toolbar to reflect any changes.

Saving and Restoring the Toolbar State

After a toolbar has been customized, you might want to return it to its former state. However, when the user customizes the toolbar, the toolbar control does not

automatically keep a record of its precustomization state. Your application must save the toolbar state explicitly in order to restore it later. Briefly:

- To save a toolbar state, send the toolbar control a **TB_SAVERESTORE** message with *lParam* set to TRUE. By default, the toolbar control will save the information automatically. With common controls **version 5.80** and later, you can gain more control over the save operation by implementing a handler for the **TBN_SAVE** notification.
- To restore a toolbar state, send the toolbar control a **TB_SAVERESTORE** message with *lParam* set to FALSE. By default, the toolbar control will send your application a series of **TBN_GETBUTTONINFO** notifications to request information on each button as it is restored. With common controls **version 5.0** and later, you can gain more control over the restore operation by implementing a handler for the **TBN_RESTORE** notification.

For a detailed discussion of this process, see *Saving and Restoring Toolbars*.

Creating In-Place Tooltips

Text strings are often used for purposes such as labeling small objects. Unfortunately, if they are long enough to display useful information, they might extend beyond the bounds of the object's display area and get clipped. A common example is file names, as seen in Microsoft Windows Explorer, which is shown in Figure 6-2.

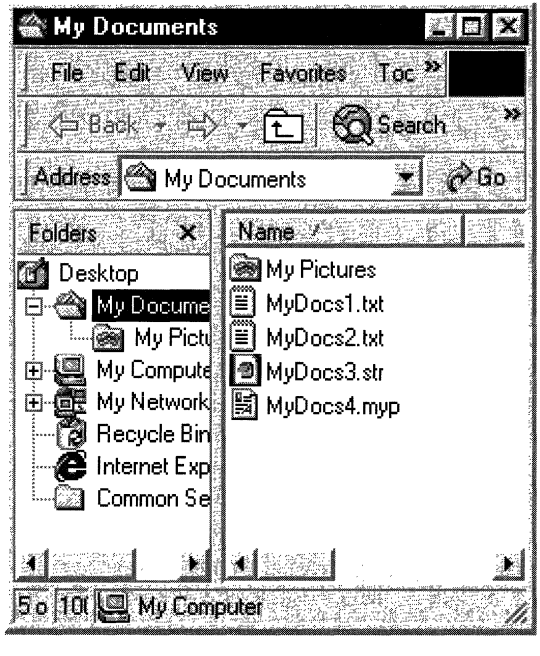


Figure 6-2: An example of clipped file names.

When the label is clipped, its usefulness can be severely limited. However, with the example in the Figure 6-2, users can see the full file name by hovering over it with the cursor. When they do so, an *in-place tooltip* with the full name is displayed on top of the clipped file name, as shown in Figure 6-3.

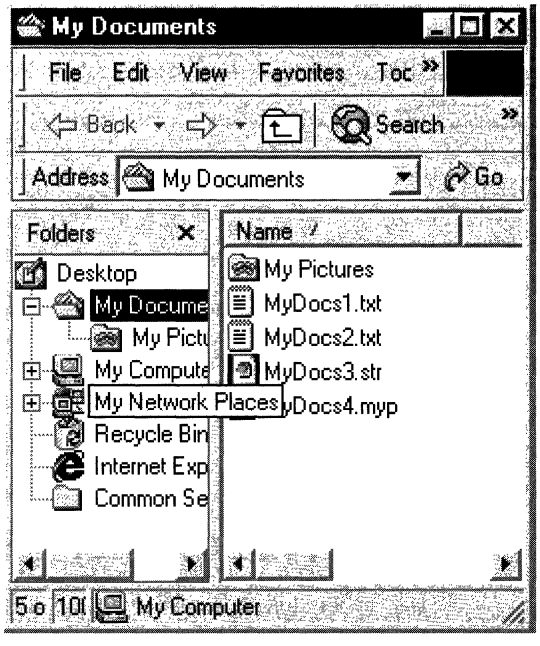


Figure 6-3: An in-place tooltip displays the full file name.

The difference between ordinary and in-place tooltips is positioning. By default, when the cursor hovers over a region that has a tooltip associated with it, the tooltip is displayed adjacent to the region. However, tooltips are windows, and they can be positioned anywhere you choose by calling **SetWindowPosition**. Creating an in-place tooltip is simply a matter of positioning the tooltip window so that it overlays the text string.

Positioning an In-Place Tooltip

You need to keep track of three rectangles when positioning an in-place tooltip:

- The rectangle that surrounds the complete label text.
- The rectangle that surrounds the tooltip text. The tooltip text is identical to the complete label text, and normally is the same size and font. The two text rectangles will thus usually be the same size.
- The tooltip's window rectangle. This rectangle is somewhat larger than the tooltip text rectangle that it encloses.

The three rectangles are shown schematically in Figure 6-4. The hidden portion of the label text is indicated by a gray background.

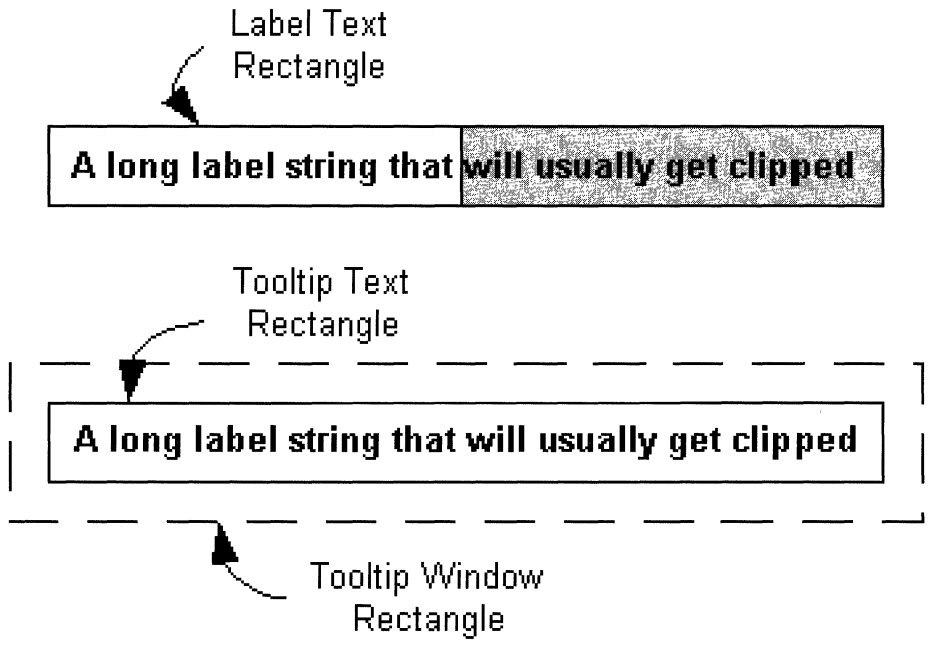


Figure 6-4: Three rectangles involved in positioning an in-place tooltip.

To create an in-place tooltip, you must position the tooltip text rectangle so that it overlays the label text rectangle. The procedure for aligning the two rectangles is relatively straightforward:

1. Define the label text rectangle.
2. Position the tooltip window so that the tooltip text rectangle overlays the label text rectangle.

In practice, it is usually sufficient to align the upper-left corner of the two text rectangles. Attempting to resize the tooltip text rectangle to exactly match the label text rectangle could cause problems with the tooltip display.

The problem with this simple scheme is that you cannot position the tooltip text rectangle directly. Instead, you must position the tooltip window rectangle just far enough above and to the left of the label text rectangle so that the corners of the two text rectangles coincide. In other words, you need to know the offset between the tooltip window rectangle and its enclosed text rectangle. In general, there is no simple way to determine this offset.

Using TTM_ADJUSTRECT to Position a Tooltip

Common controls version 5.0 simplifies the use of in-place tooltips by the addition of a new message, **TTM_ADJUSTRECT**. Send this message with the coordinates of the label text rectangle that you want the tooltip to overlay, and it will return the coordinates of an appropriately positioned tooltip window rectangle.

The following code fragment illustrates how to use **TTM_ADJUSTRECT** in a **TTN_SHOW** handler to display an in-place tooltip. Your application indicates that the label text is truncated by setting the private fMyStringIsTruncated variable to TRUE. The handler calls an application-defined function, GetMyItemRect, to get the label text rectangle. This rectangle is passed to the tooltip control with TTM_ADJUSTRECT, which returns the corresponding window rectangle. **SetWindowPosition** then is called to position the tooltip over the label.

```
case TTN_SHOW:

if (fMyStringIsTruncated) {
    RECT rc;

    GetMyItemRect(&rc);
    SendMessage(hwndToolTip, TTM_ADJUSTRECT, TRUE,
(LPARAM)&rc);
    SetWindowPos(hwndToolTip,
                 NULL,
                 rc.left, rc.top,
                 0, 0,
                 SWP_NOSIZE|SWP_NOZORDER|
                 SWP_NOACTIVATE);
}
```

This example does not change the size of the tooltip, just its position. The two text rectangles will be aligned at their upper-left corners, but not necessarily with the same dimensions. In practice, the difference is usually small, and this approach is recommended for most purposes. While you can, in principle, use SetWindowPos to resize as well as reposition the tooltip, doing so might have unpredictable consequences.

Creating an Internet Explorer-Style Toolbar

One of the key user-interface features of Microsoft Internet Explorer is the toolbar. It not only gives users access to a wide array of features, it also allows users to customize its layout to suit their personal preferences. Figure 6-5 shows the Internet Explorer toolbar, and highlights some of the key features.

This toolbar essentially consists of a rebar control with four bands: three toolbars and a menu bar. Because it is implemented with the common controls API, developers can create toolbars with any or all of its features. This document discusses the essential features of the Internet Explorer toolbar and how to implement them in your application.

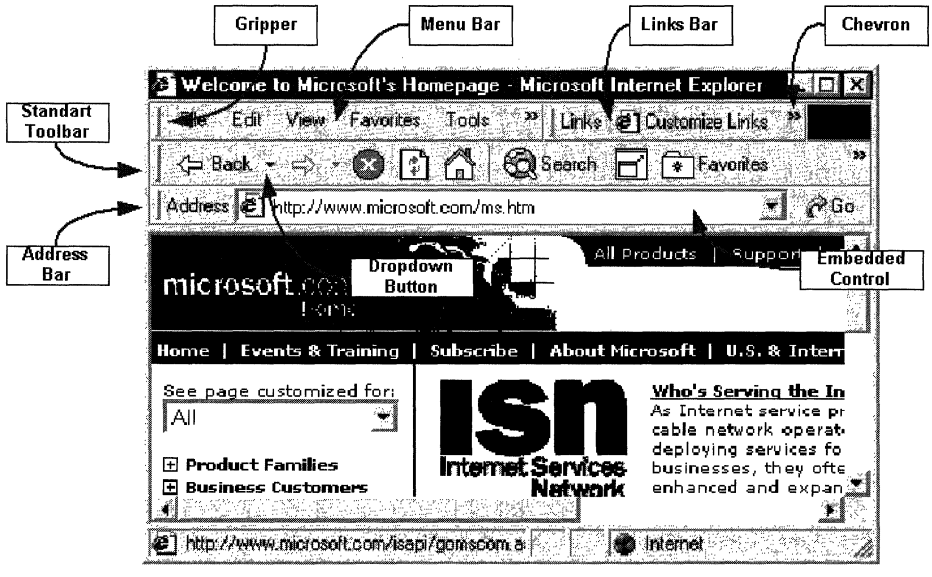


Figure 6-5: Internet Explorer toolbar.

The Rebar Control

The underlying structure of the Internet Explorer toolbar is provided by a rebar control. This control provides a way for users to customize the arrangement of a collection of tools. Each rebar contains one or more *bands*, which are typically long, narrow rectangles that contain a child window, commonly a toolbar control.

The rebar control displays its bands in a rectangular area, typically at the top of the window. This rectangle is subdivided into one or more strips that are the height of a band. Each band can be on a separate strip, or multiple bands can be placed on the same strip.

A rebar control provides users with two ways to arrange their tools:

- Each band usually has a *gripper* at its left-hand edge. Grippers are used when two or more bands on a single strip exceed the width of the window. By dragging the gripper to the left or right, users can control how much space is allocated to each band.
- Users can move the bands within the rebar's display rectangle by dragging and dropping. The rebar control then changes the display to accommodate the new arrangement of bands. If all the bands are removed from a strip, the height of the rebar will be reduced, enlarging the viewing area.
- An application can add or remove bands as needed. Typically, applications allow users to select which bands they want to have displayed through the View menu or a context menu.

If the combined width of the bands on a strip exceeds the width of the window, the rebar control will adjust their widths as needed. Some of the tools might be covered by the adjacent band.

Version 5.80 of the common controls provides a way to make tools that have been covered by another band accessible to the user. If you set the RBBS_USECHEVRON flag in the **fStyle** member of the band's **REBARBANDINFO** structure, a *chevron* will be displayed for toolbars that have been covered. When a user clicks the chevron, a menu is displayed that allows him or her to use the hidden tools. Figure 6-6, taken from Internet Explorer 5.0, shows the menu that is displayed when part of the standard toolbar is covered by the address bar.

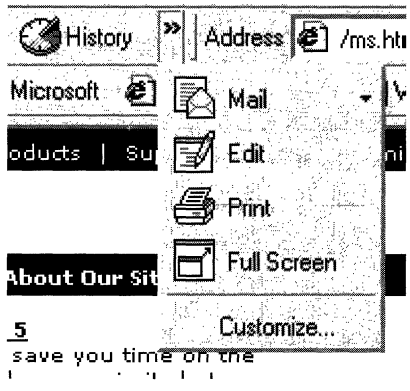


Figure 6-6: Portion of a menu that is displayed when the address bar covers the standard toolbar.

Since each band contains a control, you can provide additional flexibility through the control's API. For example, you can implement toolbar customization to allow the user to add, move, or delete buttons on a toolbar.

Implementing the Rebar Control

Most of the features of the Internet Explorer toolbar are actually implemented in the individual bands. The implementation of the rebar control itself is relatively straightforward:

1. Create the rebar control with **CreateWindowEx**. Set *dwExStyle* to WS_EX_TOOLWINDOW and *lpClassName* to REBARCLASSNAME. Internet Explorer uses the following window styles:
   - **CCS_NODIVIDER**
   - **CCS_NOPARENTALIGN**
   - **RBS_BANDBORDERS**
   - **RBS_DBLCLKTOGGLE**
   - **RBS_REGISTERDROP**
   - **RBS_VARHEIGHT**

- **WS_BORDER**
- **WS_CHILD**
- **WS_CLIPCHILDREN**
- **WS_CLIPSIBLINGS**
- **WS_VISIBLE**

  Set the other parameters as appropriate for your application.

2. Create a control with **CreateWindowEx** or a specialized control creation function such as **CreateToolbarEx**.
3. Initialize a band for the control by filling in the members of **REBARBANDINFO**. Include the RBBS_USECHEVRON style with the **fStyle** member to enable chevrons.
4. Add the band to the rebar control with an **RB_INSERTBAND** message.
5. Repeat steps 2–4 for the remaining bands.
6. Implement handlers for the rebar notifications. In particular, you will need to handle **RBN_CHEVRONPUSHED** to display a dropdown menu when a *chevron* is clicked. For further information, see **Handling Chevrons**.

The grippers are included by default. To omit the gripper for a band, set the RBBS_NOGRIPPER flag in the **fStyle** member of the band's **REBARBANDINFO** structure. For further information on implementing rebar controls, see *Rebar Controls*.

Handling Chevrons

When a user clicks a chevron, the rebar control sends your application an **RBN_CHEVRONPUSHED** notification. The **NMREBARCHEVRON** structure that is passed with the notification contains the band's identifier and a **RECT** structure with the rectangle occupied by the chevron. Your handler must determine which buttons are hidden and display the associated commands on a pop-up menu.

The following procedure outlines how to handle an **RBN_CHEVRONPUSHED** notification:

1. Get the current bounding rectangle for the selected band by sending the rebar control an **RB_GETRECT** message.
2. Get the total number of buttons by sending the band's toolbar control a **TB_BUTTONCOUNT** message.
3. Starting from the leftmost button, get the button's bounding rectangle by sending the toolbar control a **TB_GETITEMRECT** message.
4. Pass the band and button rectangles to **IntersectRect**. This function will return a **RECT** structure that corresponds to the visible portion of the button.
5. Pass the button rectangle and the rectangle for the visible portion of the button to **EqualRect**.
6. If **EqualRect** returns TRUE, the entire button is visible. Repeat steps 3–5 for the next button on the toolbar. If **EqualRect** returns FALSE, the button is at least partially hidden and all remaining buttons will be hidden completely. Continue to the next step.

7. Create a pop-up menu with items for each of the hidden buttons.
8. Display the pop-up menu with **TrackPopupMenu**. Use the chevron rectangle passed with the **RBN_CHEVRONPUSHED** notification to position the menu. The menu should be immediately below the chevron, with the left edges aligned.
9. Handle the menu commands.

The Toolbars

Most of the complexity of the Internet Explorer toolbar lies in the implementation of controls that make up the rebar bands. Internet Explorer commonly displays four bands:

- The menu bar
- The standard toolbar
- The links toolbar
- The address toolbar

All of these bands, including the menu bar, actually hold toolbar controls. This section discusses the implementation of the standard and links toolbars. The menu bar is somewhat more complicated and is discussed separately in *Creating an Internet Explorer-Style Menu Bar*.

The basic procedures for implementing toolbar controls are discussed in *Toolbar Controls*. This section focuses on some of the newer toolbar features that are used by Internet Explorer to increase the usability of the control.

Drop-Down Buttons

Drop-down buttons support multiple commands. When the user clicks a drop-down button, the button displays a pop-up menu instead of launching a command. The user launches a command by selecting it from the menu. Figure 6-7 shows a drop-down button and menu from the Internet Explorer standard toolbar.

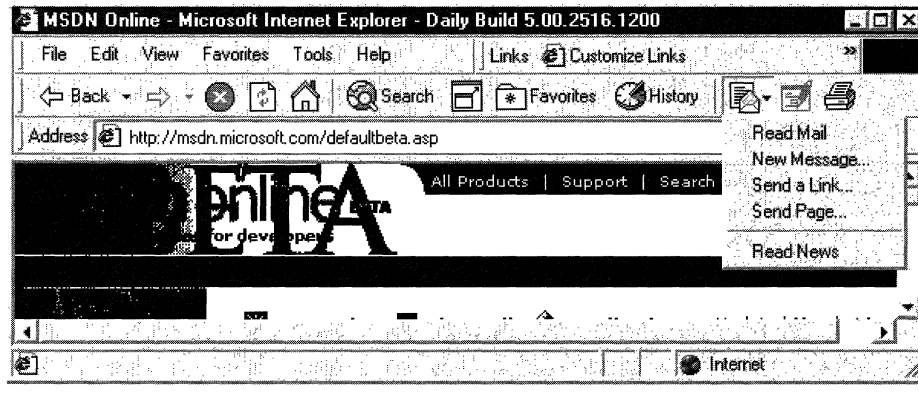


Figure 6-7: A drop-down button and menu from the standard toolbar in Internet Explorer.

Drop-down functionality can be added to any button style by adding a style flag to the **fStyle** member of the button's **TBBUTTON** structure. There are three styles of drop-down button, all of which are used by Internet Explorer:

- Plain drop-down buttons have the **BTNS_DROPDOWN** style. They look like normal buttons, but they display a menu when clicked instead of launching a command.
- Simple drop-down arrow buttons have the **BTNS_WHOLEDROPDOWN** style. They have an arrow displayed next to the button image or text. Other than the difference in appearance, they are identical to plain drop-down buttons. The Mail button used as the example in the preceding illustration is a drop-down arrow button.
- Drop-down arrow buttons that add the **TBSTYLE_EX_DRAWDDARROWS** extended style to **BTNS_DROPDOWN** have an arrow that is separated from the text or image. This button style combines the functionality of drop-down and standard buttons. If the user clicks the arrow, a menu is displayed and the user can choose from several commands. If the user clicks the adjacent button, it launches a default command. Figure 6-8 shows the Internet Explorer **Back** button, which uses a separated arrow.

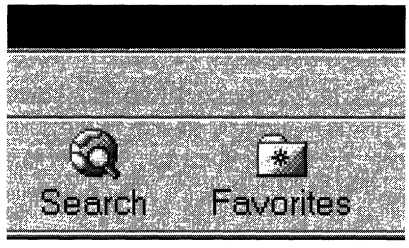
Figure 6-8: The Back button in Internet Explorer.

When the user clicks a drop-down button with either the plain or simple arrow styles, the toolbar control sends your application a **TBN_DROPDOWN** notification. When your application receives this message, it is responsible for creating and displaying the menu, and for handling the selected command. For further discussion, see *Toolbar Controls*.

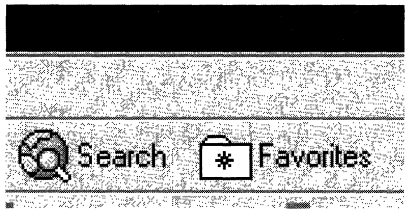
When the user clicks a separated arrow, the toolbar control sends your application a **TBN_DROPDOWN** notification. Your application should handle it the same way as it handles the other two types of drop-down buttons. If the user clicks the main button, your application receives a **WM_COMMAND** message with the button's command ID, just as if it were a standard button. Applications typically respond by launching the top command in the drop-down menu, but you are free to respond in any suitable way.

List-Style Buttons

With standard buttons, if you add text, it is displayed below the bitmap. The following illustration shows the Internet Explorer **Search** and **Favorites** buttons with standard button text.



Internet Explorer 5 uses the **TBSTYLE_LIST** style. The text is to the right of the bitmap, reducing the height of the button and enlarging the viewing region. The following illustration shows the Internet Explorer 5 Search and Favorites buttons with the TBSTYLE_LIST style.

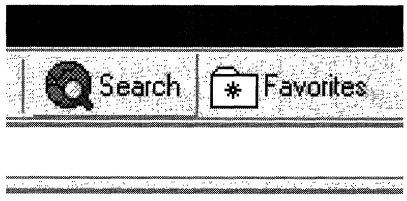


Chevrons

When the user rearranges the bands in the rebar control, part of a toolbar might be covered up. If the band was created with the RBBS_USECHEVRON style, the rebar control will display a chevron at the right edge of the toolbar. The user clicks the chevron to display a menu with the hidden tools. For a discussion on how to implement chevrons, see *Handling Chevrons*.

Hot-Tracking

When hot-tracking is enabled, a button becomes *hot* when the cursor is over it. The hot button is normally distinguished from the other buttons on the toolbar by a distinctive image. By default, a hot button appears to be raised above the rest of the toolbar. When a new button becomes hot, your application receives a **TBN_HOTITEMCHANGE** notification. The following illustration shows the Internet Explorer 5.0 **Search** and **Favorites** buttons, with a hot **Search** button. In addition to having a raised appearance, the button's gray bitmap has been replaced with a colored one.



To enable hot-tracking, create a toolbar control with either the **TBSTYLE_FLAT** or **TBSTYLE_LIST** style. These are referred to as *flat* toolbars because the individual buttons are not ordinarily highlighted in any way. The bitmaps are simply displayed next to each other. They take on a button-like appearance only when they are hot. These two styles are also transparent, which means the background of the icons will be the color of the underlying client window.

To have a different bitmap displayed when the button is hot, create a second **image list** containing hot images for all the buttons on the toolbar. The size and order of these images should be the same as in the default image list. Send the toolbar control a **TB_SETIMAGELIST** to set the hot image list.

Creating an Internet Explorer-Style Menu Bar

At a glance, the Microsoft Internet Explorer 5.0 menu bar looks much like a standard menu. However, it looks quite different in use. Figure 6-9 shows the Internet Explorer menu bar with the **Tools** menu selected.

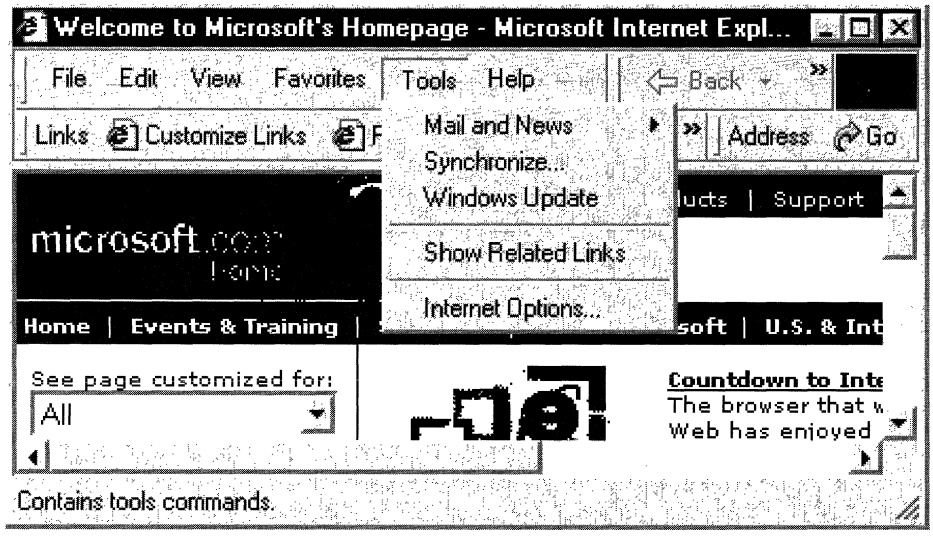


Figure 6-9: Tools menu is selected on the Internet Explorer menu bar.

The menu bar is actually a toolbar control that looks and functions very much like a standard menu. Instead of top-level menu items, a menu bar has a series of text-only

buttons that display a drop-down menu when clicked. However, as a specialized type of toolbar, a menu bar has some capabilities that standard menus lack. As a toolbar control:

- It can be customized using standard toolbar techniques, allowing the user to move, delete, or add items.
- It can have hot-tracking, so that users will know when they are over a top-level item without having to click it first.

A menu bar can be incorporated into a rebar control, giving it the following features:

- It can have a gripper, which allows the user to move or resize the band.
- It can share a strip in the rebar control with other bands, such as the standard toolbar in the preceding illustration.
- It can display a chevron when it is covered by an adjacent band, giving the user access to the hidden items.
- It can have an application-defined background bitmap.

This document discusses how to implement a menu bar. Since a menu bar is a specialized implementation of a toolbar control, the focus will be on topics that are specific to menu bars. For a discussion of how to incorporate a toolbar into a rebar control, see *Creating an Internet Explorer-Style Toolbar* and *Rebar Controls*.

Making a Toolbar into a Menu Bar

To make a toolbar into a menu bar:

- Create a flat toolbar by including **TBSTYLE_FLAT** with the other window style flags. The TBSTYLE_FLAT style also enables hot-tracking. With this style, the menu bar looks much like a standard menu until the user activates a button. Then, the button appears to stand out from the toolbar and depress when it is clicked, just like a standard button. Because hot-tracking is enabled, all that is needed to activate a button is for the cursor to hover over it. If the cursor moves to another button, it will be activated and the old button deactivated.
- Create list-style buttons by including **TBSTYLE_LIST** with the other window style flags. This style creates a thinner button that looks more like a standard top-level menu item.
- Make the buttons text-only by setting the **iBitmap** member of the button's **TBBUTTON** structure to I_IMAGENONE and the **iString** member to the button text.
- Give each button the **BTNS_DROPDOWN** style. When the button is clicked, the toolbar control sends your application a **TBN_DROPDOWN** notification to prompt it to display the button's menu.
- Incorporate the menu bar into a rebar band. Enable both grippers and chevrons, as discussed in *Creating an Internet Explorer-Style Toolbar*.

- Implement a **TBN_DROPDOWN** handler to display the button's *drop-down menu* when it is clicked. The drop-down menu is a type of pop-up menu. It is created with **TrackPopupMenu**, with its upper-left corner aligned with the lower-left corner of the button.
- Implement keyboard navigation, so that the menu bar is fully accessible without a mouse.
- Implement menu hot-tracking. With standard menus, once a top level menu item's menu has been displayed, moving the cursor over another top-level item automatically displays its menu and collapses the menu of the previous item. The toolbar control will hot-track the cursor and change the button image, but it does automatically display the new menu. Your application must do so explicitly.

Most of these items are straightforward to implement and are discussed elsewhere. See *Creating an Internet Explorer-Style Toolbar*, *Toolbar Controls*, or *Rebar Controls* for a general discussion of how to use toolbars and rebar controls. See *Menus* for a discussion of how to handle pop-up menus. The final two items, keyboard navigation and menu hot-tracking, are discussed in the remainder of this document.

Handling Navigation with Menu Hot-Tracking Disabled

Users can choose to navigate the menu bar with the mouse, the keyboard, or a mixture of both. To implement menu bar navigation, your application needs to handle toolbar notifications and monitor the mouse and keyboard. This task can be broken into two distinct parts: with and without menu hot-tracking. This section discusses how to handle navigation when no menus are displayed and menu hot-tracking is not enabled.

Mouse Navigation

If menu hot-tracking is disabled, you can treat a menu bar as a normal toolbar. If the user is navigating with a mouse, all your application needs to do is handle the **TBN_DROPDOWN** notification. When this notification is received, display the appropriate drop-down menu, and enable menu hot-tracking. From then on, you are in the menu hot-tracking regime, discussed in **Implementing Menu Hot-Tracking**.

As discussed in **Mixed Navigation**, you also need to handle the **TBN_HOTITEMCHANGE** notification to keep track of the active button. This notification is irrelevant if the user is only navigating with the mouse, but it is necessary when mouse and keyboard navigation are mixed.

Keyboard Navigation

As noted in the previous section, the user can do a number of navigation operations with the keyboard when menu hot-tracking is disabled. Toolbar controls support keyboard navigation only if they have focus. They also do not handle all the key presses that are needed for menu bars. The simplest solution to handling keyboard navigation is to process the relevant key press events explicitly.

If menu hot-tracking is disabled, your application needs to handle these key press events in the following way:

- The F10 key. Activate the first button with **TB_SETHOTITEM**.
- The LEFT ARROW and RIGHT ARROW keys. Activate the adjacent button with **TB_SETHOTITEM**. If the user attempts to navigate off either end of the menu bar, activate the first button at the opposite end.
- The ESCAPE key. Deactivate the active button with **TB_SETHOTITEM**. The menu bar should be returned to the state it had prior to activating the first button.
- An ALT-*Key* accelerator key. Use the **TB_MAPACCELERATOR** message to determine which button the *Key* character corresponds to. Display the button's drop-down menu and enable menu hot-tracking.
- The DOWN ARROW key. If a button is active but no menu has been displayed, display the button's menu and enable menu hot-tracking.
- The ENTER key. Deactivate the active button with **TB_SETHOTITEM**. The menu bar should be returned to the state it had prior to activating the first button.

Mixed Navigation

The keyboard navigation handlers outlined in the preceding section basically do two tasks: keep track of the active button and display the appropriate menu when a button is selected. These handlers are sufficient as long as the user navigates only with the keyboard. However, users often mix keyboard and mouse navigation. For example, they might activate the first button with the F10 key, use the mouse to activate a different button, and then open its menu with the DOWN ARROW key. If you are only monitoring key presses to keep track of the active key, you will display the wrong menu. You must also handle the **TBN_HOTITEMCHANGE** notification to accurately keep track of the active button.

Handling Navigation with Menu Hot-Tracking Enabled

Once menu hot-tracking is enabled, your application must change the way it responds to user navigation. To replicate the behavior of standard menus, you must implement the following features explicitly.

With mouse navigation:

- If the user moves the cursor over another button, that menu immediately appears and the previous menu disappears.
- Menu hot-tracking stays active until the user selects a command or clicks a point that is not part of the menu region. Either action deactivates menu hot-tracking until another button is clicked.
- If the cursor moves off the menu, the current drop-down menu remains until the cursor moves back onto, or the user clicks a point outside, the area covered by the menu. If the cursor returns to a different button than the one currently being displayed, the old menu is collapsed and the new menu is displayed.

With keyboard navigation:

- The RIGHT ARROW key. If the item has a submenu, display the submenu. If the item does not have a submenu, collapse the menu and any submenus, activate the next button with **TB_SETHOTITEM**, and display the menu for the adjacent button. If the last button is active when this message is received, display the menu for the first button.
- The LEFT ARROW key. If the item is a submenu, collapse it and shift focus to its parent menu. If the item is not a submenu, collapse the menu, activate the next button with **TB_SETHOTITEM**, and display the menu for that button.
- Pressing the LEFT ARROW key moves the focus to the left.
  - If the highlighted menu item is on the primary menu, that menu is collapsed, and the menu for the adjacent button is displayed. If the active button was at the left end of the toolbar, the menu for the last button is displayed.
  - If the highlighted menu item is on a submenu, the submenu is collapsed, shifting the focus back to its parent.
- Pressing the RIGHT ARROW key moves the focus to the right.
  - If the highlighted menu item does not have a submenu, the menu for the adjacent button is displayed. If the active button was at the right end of the toolbar, the menu for the last button is displayed.
  - If the highlighted menu item has a submenu, the submenu is displayed.
- The ESCAPE key backs the display up one step.
  - If a submenu is displayed, it is collapsed and focus shifts to the parent menu.
  - If a button's menu is displayed, it is collapsed and menu hot-tracking is disabled. The item's button remains active.
  - If no menus are displayed but a button is active, the button is deactivated and menu hot-tracking is disabled.
- The UP ARROW and DOWN ARROW keys are used only to navigate within a particular menu.
- The ENTER key launches the command associated with a menu item. If the menu item has a submenu, the ENTER key displays it.

As with the menu hot-tracking disabled case, your application needs to handle mouse, keyboard, and mixed navigation. However, the fact that a menu is displayed means that messaging will have to be handled somewhat differently.

Message Processing for Menu Hot-Tracking

Menu hot-tracking requires that a menu be displayed at all times, apart from the brief interval required to switch to a new menu. However, the drop-down menu that is displayed by **TrackPopupMenu** is modal. Your application will continue to receive some messages directly, including **WM_COMMAND**, **TBN_HOTITEMCHANGE**, and normal menu-related messages such as **WM_MENUSELECT**. However, it will not receive low-level keyboard or mouse messages directly. To handle messages such as **WM_MOUSEMOVE**, you must set a message hook to intercept messages directed to the menu.

When a drop-down menu is displayed, set the message hook by calling **SetWindowsHookEx** with the *idHook* parameter set to WH_MSGFILTER. All messages intended for the menu will be passed to your hook procedure. For example, the following code fragment sets a message hook that will trap messages going to a drop-down menu. MsgHook is the name of the hook procedure, and hhookMsg is the handle to the procedure.

```
hhookMsg = SetWindowsHookEx(WH_MSGFILTER, MsgHook,HINST_THISDLL, 0);
```

Menu messages are identified by setting the hook procedure's *nCode* parameter to MSGF_MENU. The *lParam* parameter will point to a **MSG** structure with the message. The details of which messages need to be handled, and how, are discussed in the remainder of this document.

Your application should pass all messages on to the next message hook by calling **CallNextHookEx**. Doing so ensures, for instance, that the toolbar control receives the mouse messages it needs to hot-track its buttons.

When a new button is activated, your application must collapse the old drop-down menu with a **WM_CANCELMODE** message, and display a new menu. It must also collapse the drop-down menu when focus is taken from the menu bar with keyboard navigation or by clicking outside the menu area. Whenever you collapse a menu, you should free its message hook with **UnhookWindowsHookEx**. If you need to display another drop-down menu, create a new message hook. When a command is launched, the menu will be collapsed automatically but you must explicitly free the message hook.

The following code fragment frees the message hook that was set in the previous example:

```
UnhookWindowsHookEx(hhookMsg);
```

Mouse Navigation

When a normal toolbar control hot-tracks buttons, it highlights the active button and sends the application a **TBN_HOTITEMCHANGE** notification each time a new button is activated. Your application is responsible for displaying the appropriate drop-down menu. It must:

- Handle the **TBN_HOTITEMCHANGE** notification to keep track of the active button. When the active button changes, collapse the old menu and create a new one.
- Handle the **TBN_DROPDOWN** notification that is sent when a button is clicked. It should then collapse the menu and disable menu hot-tracking. The button remains active.
- Handle the **WM_LBUTTONDOWN**, **WM_RBUTTONDOWN**, and **WM_MOUSEMOVE** messages in your message hook procedure, to keep track of the mouse position. If the mouse is clicked outside the menu area, collapse the current drop-down menu, deactivate menu hot-tracking, and return the menu bar to its preactivation state.
- When a menu command is launched, disable menu hot-tracking. The menu will be collapsed automatically but you must free the message hook explicitly.

You also must handle menu-related messaging, such as the **WM_INITMENUPOPUP** message that is sent when a menu item needs to display a submenu. For a discussion of how to handle such messages, see *Menus*.

Keyboard Navigation

Your application must process keyboard messages in the message hook procedure and act on those that affect menu hot-tracking. The following key presses need to be handled:

- The ESCAPE key. The ESCAPE key backs the display up one level. If a submenu is being displayed, it must be collapsed. If a button's primary menu is displayed, collapse it and disable menu hot-tracking. The button remains active.
- The RIGHT ARROW key. If the item has a submenu, display it. If the item does not have a submenu, collapse the menu and any submenus, activate the next button with **TB_SETHOTITEM**, and display its menu. If the last button was active when this notification was received, display the menu for the first button.
- The LEFT ARROW key. If the item is in a submenu, collapse it and shift focus to its parent menu. If the item is not a submenu, collapse the menu, activate the adjacent button with **TB_SETHOTITEM**, and display its menu. If the first button was active when this notification was received, display the menu for the last button.
- The UP ARROW and DOWN ARROW keys. These keys are used to navigate within a menu but do not directly affect menu hot-tracking.
- An ALT-*Key* accelerator key. Use the **TB_MAPACCELERATOR** message to determine which button the *Key* character corresponds to. If it corresponds to a different button than the currently active one, collapse the current drop-down menu, activate the new button with **TB_SETHOTITEM**, and display the menu for the adjacent button. If the *Key character* corresponds to the currently displayed button, collapse the drop-down menu and disable menu hot-tracking. The button should remain active.

Localization Support for the Common Controls

The common controls have built-in support for national languages. These features simplify the implementation of localized applications.

Specifying a Language for the Common Controls

If you want to specify a language for the common controls that is different than the system language, call **InitMUILanguage**. The language specified by this function applies only to the process from which it is called.

To determine the language currently being used by the common controls, call **GetMUILanguage**. It returns the value that was set by a previous call to **InitMUILanguage**. This function returns the language that is specified for the process it is called from. If **InitMUILanguage** has not been called, or was called from another process, **GetMUILanguage** will return a default value.

Specifying a Language for Controls in a Dialog Box

Unlike common controls, predefined controls such as buttons or edit boxes do not use the current system language by default. The native font control is an invisible control that works in the background to allow a dialog box's predefined controls to display the current system language.

To use the native font control:

1. Initialize the native font control by calling **InitCommonControlsEx**. Set the **dwICC** member of the **INITCOMMONCONTROLSEX** structure pointed to by *lpInitCtrls* to ICC_NATIVEFNTCTL_CLASS.
2. Add the control to the resource script for the dialog box. Set one or more of the following style flags to specify which controls will be affected:

Flag	Applies to
NFS_ALL	All controls.
NFS_BUTTON	Button controls.
NFS_EDIT	Edit controls.
NFS_LISTCOMBO	List, ComboBox, ListView, and ComboBoxEx controls.
NFS_STATIC	Static controls
NFS_USEFONTASSOC	The control will use the font association feature instead of switching to the native font. This flag is only valid for the Far East platform. All other platforms will ignore it.

The following example illustrates how to add a native font control to a resource script. It will cause the dialog box's edit, list, and combo-box controls to display text using the current system language:

```
CONTROL   "",-1,'NativeFontCtl',NFS_EDIT|NFS_LISTCOMBO,0,0,0,0
```

CHAPTER 7

Common API

Common Control Window Classes

The following window class names are provided by the common control library:

ANIMATE_CLASS	Creates animation controls. These controls silently display an audio video interleaved (AVI) clip.
DATETIMEPICK_CLASS	Creates date and time picker controls. These controls provide a simple and intuitive interface to exchange date and time information with a user.
HOTKEY_CLASS	Creates hot-key controls. These controls make it easy for the user to define hot keys.
MONTHCAL_CLASS	Creates month calendar controls. These controls provide a simple and intuitive way for a user to select a date from a familiar interface.
PROGRESS_CLASS	Creates progress bars. These controls indicate the progress of a lengthy operation.
REBARCLASSNAME	Creates rebar controls. These controls act as a container for child windows.
STATUSCLASSNAME	Creates status windows. These controls display status information in a horizontal window.
TOOLBARCLASSNAME	Creates toolbars. These controls contain buttons that carry out menu commands.
TOOLTIPS_CLASS	Creates tooltip controls. These controls display a small pop-up window containing a line of text that describes the purpose of a tool in an application.
TRACKBAR_CLASS	Creates trackbars. These controls let the user select from a range of values by moving a slider.
UPDOWN_CLASS	Creates up-down controls. These controls combine a pair of arrows with an edit control. Clicking the arrows increments or decrements the value in the edit control.
WC_COMBOBOXEX	Creates ComboBoxEx controls. These controls provide an extension of the combo box control that provides native support for item images.

(continued)

(continued)

WC_HEADER	Creates header controls. These controls display headings at the top of columns of information and let the user sort the information by clicking the headings.
WC_IPADDRESS	Creates IP-address controls. These controls are similar to an edit control, but they allow you to enter a numeric address in Internet protocol (IP) format.
WC_LISTVIEW	Creates list-view controls. These controls display a collection of items, each consisting of an icon and a label, and provide several ways to arrange the items.
WC_PAGESCROLLER	Creates pager controls. These controls are used to contain and scroll another window.
WC_TABCONTROL	Creates tab controls. These controls define multiple pages for the same area of a window or dialog box. Each page consists of a set of information or a group of controls that an application displays when the user selects the corresponding tab.
WC_TREEVIEW	Creates tree-view controls. These controls display a hierarchical list of items. Each item consists of a label and an optional bitmap.

Common Control Styles

Following are the common control styles. Except where noted, these styles apply to header controls, toolbar controls, and status windows:

CCS_ADJUSTABLE	Enables a toolbar's built-in customization features, which allow the user to drag a button to a new position or to remove a button by dragging it off the toolbar. In addition, the user can double-click the toolbar to display the **Customize Toolbar** dialog box, which allows the user to add, delete, and rearrange toolbar buttons.
CCS_BOTTOM	Causes the control to position itself at the bottom of the parent window's client area and sets the width to be the same as the parent window's width. Status windows have this style by default.
CCS_LEFT	**Version 4.70**. Causes the control to be displayed vertically on the left side of the parent window.
CCS_NODIVIDER	Prevents a two-pixel highlight from being drawn at the top of the control.

CCS_NOMOVEX	**Version 4.70**. Causes the control to resize and move itself vertically, but not horizontally, in response to a **WM_SIZE** message. If CCS_NORESIZE is used, this style does not apply.
CCS_NOMOVEY	Causes the control to resize and move itself horizontally, but not vertically, in response to a **WM_SIZE** message. If CCS_NORESIZE is used, this style does not apply. Header windows have this style by default.
CCS_NOPARENTALIGN	Prevents the control from automatically moving to the top or bottom of the parent window. Instead, the control keeps its position within the parent window despite changes to the size of the parent. If CCS_TOP or CCS_BOTTOM is also used, the height is adjusted to the default, but the position and width remain unchanged.
CCS_NORESIZE	Prevents the control from using the default width and height when setting its initial size or a new size. Instead, the control uses the width and height specified in the request for creation or sizing.
CCS_RIGHT	**Version 4.70**. Causes the control to be displayed vertically on the right side of the parent window.
CCS_TOP	Causes the control to position itself at the top of the parent window's client area and sets the width to be the same as the parent window's width. Toolbars have this style by default.
CCS_VERT	**Version 4.70**. Causes the control to be displayed vertically.

Common API Reference

Common API Functions

GetEffectiveClientRect

Calculates the dimensions of a rectangle in the client area.

```
void GetEffectiveClientRect(
  HWND hWnd,
  LPRECT lprc,
  LPINT lpInfo
);
```

Parameters

hWnd
Handle to the window that has the client area to check.

lprc
Address of a **RECT** structure that receives the dimensions of the rectangle.

lpInfo
Address of an array of integers that identify controls in the client area. Each control requires a pair of consecutive elements. The first element of the pair must be nonzero and the second element of the pair must be the control identifier. The last element pair in the array must be zero to identify the end of the array.

Return Values

No return value.

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 commctrl.h.
Import Library: comctl32.lib.

GetMUILanguage

Returns the language currently in use by the common controls for a particular process.

```
LANGID GetMUILanguage( VOID );
```

Parameters

None

Return Values

Returns the **LANGID** of the language an application has specified for the common controls by calling **InitMUILanguage**. **GetMUILanguage** returns the value for the process that it is called from. If **InitMUILanguage** has not been called or was not called from the same process, **GetMUILanguage** returns the language-neutral LANGID, **MAKELANGID**(LANG_NEUTRAL, SUBLANG_NEUTRAL).

Remarks

See *National Language Support* for further discussion of localization.

Requirements

Version 5.80 or later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later installed).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.
Import Library: comctl32.lib.

InitCommonControls

Registers and initializes the common control window classes. This function is obsolete. New applications should use the **InitCommonControlsEx** function.

```
void InitCommonControls(VOID);
```

Return Values

No return value.

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 commctrl.h.
Import Library: comctl32.lib.

InitCommonControlsEx

Registers specific common control classes from the common control dynamic-link library (DLL).

```
BOOL InitCommonControlsEx(
  LPINITCOMMONCONTROLSEX lpInitCtrls
);
```

Parameters

lpInitCtrls
Address of an **INITCOMMONCONTROLSEX** structure that contains information specifying which control classes will be registered.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

Note The effect of each call to **InitCommonControlsEx** is cumulative. For example, if InitCommonControlsEx is called with the **ICC_UPDOWN_CLASS** flag, then is later called with the **ICC_HOTKEY_CLASS** flag, the result is that both the up-down and hot key common control classes are registered and available to the application.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.
Import Library: comctl32.lib.

InitMUILanguage

Enables an application to specify a language to be used with the common controls that is different than the system language.

```
VOID InitMUILanguage(
  LANGID uiLang
);
```

Parameters

uiLang
The **LANGID** value of the language to be used by the common controls.

Return Values

None.

Remarks

This function allows an application to override the system language setting, and specify a different language for the common controls. The selected language only applies to the process that **InitMUILanguage** is called from. See *National Language Support* for further discussion of localization.

Requirements

Version 5.80 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4 with Internet Explorer 5.0 or later installed).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.
Import Library: comctl32.lib.

See Also

GetMUILanguage

ShowHideMenuCtl

Sets or removes the specified menu item's check mark attribute and shows or hides the corresponding control. The function adds a check mark to the specified menu item if it does not have one and then displays the corresponding control. If the menu item already has a check mark, the function removes the check mark and hides the corresponding control.

```
BOOL ShowHideMenuCtl(
  HWND hWnd,
  UINT uFlags,
  LPINT lpInfo
);
```

Parameters

hWnd
Handle to the window that contains the menu and controls.

uFlags
Identifier of the menu item to receive or lose a check mark.

lpInfo
Address of an array that contains pairs of values. The second value in the first pair must be the handle to the application's main menu. Each subsequent pair consists of a menu item identifier and a control window identifier. The function searches the array for a value that matches *uFlags* and, if the value is found, checks or unchecks the menu item and shows or hides the corresponding control.

Return Values

Returns nonzero if successful, 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 commctrl.h.
Import Library: comctl32.lib.

Common API Messages

CCM_GETUNICODEFORMAT

The CCM_GETUNICODEFORMAT message retrieves the Unicode character format flag for the control.

```
CCM_GETUNICODEFORMAT
    wParam = 0;
    lParam = 0;
```

Parameters

This message has no parameters.

Return Values

Returns the Unicode format flag for the control. If this value is nonzero, the control is using Unicode characters. If this value is zero, the control is using ANSI characters.

Example

The following function can be used with a Microsoft Windows 95 or Microsoft Windows 98 system to test whether or not a property sheet control supports Unicode. For more information about testing controls for Unicode support, see Remarks.

```
BOOL IsComctl32Unicode(void)
{
    PROPSHEETPAGEW pspW = { PROPSHEETPAGEW_V1_SIZE };
    HPROPSHEETPAGE hpage = CreatePropertySheetW(&pspW);
    if (hpage) {
        DestroyPropertySheetPage(hpage);
        return TRUE;
    }
    else {
        return FALSE;
    }
}
```

Remarks

The Unicode format flag is used by Microsoft Windows NT systems with version 4.71 of Comctl32.dll or later. This message, thus, is supported by Windows 2000 and later, and by Windows NT 4.0 with Microsoft Internet Explorer 4.0 or later. It is useful only on Windows 95 or Windows 98 systems with version 5.80 or later of Comctl32.dll. This means that they must have Internet Explorer version 5.0 or later installed. Windows 95 and Windows 98 systems with earlier versions of Internet Explorer ignore the Unicode format flag, and its value has no bearing on whether a control supports Unicode. With these systems, you will need to test instead something that requires Unicode support.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

See Also

CCM_SETUNICODEFORMAT

CCM_GETVERSION

Returns the version number for a control set by the most recent **CCM_SETVERSION** message.

```
CCM_GETVERSION
    wParam = 0;
    lParam = 0;
```

Parameters

None

Return Values

Returns the version number set by the most recent **CCM_SETVERSION** message. If no such message has been sent, it returns zero.

Remarks

This message does not return the DLL version. See **Shell Versions** for a discussion of how to use **DllGetVersion** to get the current DLL version.

Note The version number is set on a control by control basis, and may not be the same for all controls.

Requirements

Version 5.80 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later installed).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

CCM_SETUNICODEFORMAT

THE CCM_SETUNICODEFORMAT message sets the Unicode character format flag for the control. This message allows you to change the character set used by the control at run time instead of having to re-create the control.

```
CCM_SETUNICODEFORMAT
    wParam = (WPARAM)(BOOL)fUnicode;
    lParam = 0;
```

Parameters

fUnicode
Value that determines the character set that is used by the control. If this value is TRUE, the control will use Unicode characters. If this value is FALSE, the control will use ANSI characters.

Return Values

Returns the previous Unicode format flag for the control.

Remarks

The Unicode format flag is used by Microsoft Windows NT systems with version 4.71 of Comctl32.dll or later. This message is thus supported by Windows 2000 and later, and by Windows NT 4.0 with Microsoft Internet Explorer 4.0 or later. It is only useful on Microsoft Windows 95 or Microsoft Windows 98 systems with version 5.80 or later of Comctl32.dll. This means that they must have Internet Explorer 5.0 or later installed. Windows 95 and Windows 98 systems with earlier versions of Internet Explorer ignore the Unicode format flag, and its value has no bearing on whether or not a control supports Unicode. For a discussion about how to test whether a control supports Unicode, see **CCM_GETUNICODEFORMAT**.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

See Also

CCM_GETUNICODEFORMAT

CCM_SETVERSION

This message is used to inform the control that you are expecting a behavior associated with a particular version.

```
CCM_SETVERSION
  wParam = (int) iVersion;
  lParam = 0;
```

Parameters

iVersion
The version number.

Return Values

Returns the version specified in the previous **CCM_SETVERSION** message. If iVersion is set to a value greater than the current DLL version, it returns –1.

Remarks

In a few cases, a control may behave differently, depending on the version. This primarily applies to bugs that were fixed in later versions. The **CCM_SETVERSION** allows you to inform the control which behavior is expected. You can determine which version you have specified by sending a **CCM_GETVERSION** message. For an example of how to use this message, see **Custom Draw**.

Note This message only sets the version number for the control to which it is sent.

Requirements

Version 5.80 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later installed).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

WM_NOTIFY

The **WM_NOTIFY** message is sent by a common control to its parent window when an event has occurred or the control requires some information.

```
WM_NOTIFY
  idCtrl = (int) wParam;
  pnmh = (LPNMHDR) lParam;
```

Parameters

idCtrl
Identifier of the common control sending the message. This identifier is not guaranteed to be unique. An application should use the **hwndFrom** or **idFrom** member of the **NMHDR** structure (passed as the *lParam* parameter) to identify the control.

pnmh
Pointer to an **NMHDR** structure that contains the notification code and additional information. For some notification messages, this parameter points to a larger structure that has the **NMHDR** structure as its first member.

Return Values

The return value is ignored except for notification messages that specify otherwise.

Remarks

If the message handler is in a dialog box procedure, you must use the **SetWindowLong** function with DWL_MSGRESULT to set a return value.

The standard Windows controls (edit controls, combo boxes, list boxes, buttons, scroll bars, and static controls) do not send WM_NOTIFY messages. To determine if a common control will send a WM_NOTIFY message and, if it will, which notification codes it will send, see the documentation for the control.

For Windows 2000 and later systems, the WM_NOTIFY message can not be sent between processes.

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_NOTIFYFORMAT

Used to determine if a window accepts ANSI or Unicode structures in the **WM_NOTIFY** notification message. WM_NOTIFYFORMAT messages are sent from a common control to its parent window and from the parent window to the common control.

```
WM_NOTIFYFORMAT
  hwndFrom = (HWND) wParam;
  Command = lParam;
```

Parameters

hwndFrom
Handle to the window that is sending the WM_NOTIFYFORMAT message. If *Command* is NF_QUERY, this parameter is the handle to a control. If *Command* is NF_REQUERY, this parameter is the handle to the parent window of a control.

Command
Command value that specifies the nature of the WM_NOTIFYFORMAT message. This will be one of the following values:

NF_QUERY	The message is a query to determine whether ANSI or Unicode structures should be used in **WM_NOTIFY** messages. This command is sent from a control to its parent window during the creation of a control and in response to an NF_REQUERY command.
NF_REQUERY	The message is a request for a control to send an NF_QUERY form of this message to its parent window. This command is sent from the parent window. The parent window is asking the control to requery it about the type of structures to use in **WM_NOTIFY** messages.

Return Values

Returns one of the following:

NFR_ANSI	ANSI structures should be used in **WM_NOTIFY** messages sent by the control.
NFR_UNICODE	Unicode structures should be used in **WM_NOTIFY** messages sent by the control.
0	An error occurred.

If *Command* is NF_REQUERY, the return value is the result of the requery operation.

Remarks

When a common control is created, the control sends a WM_NOTIFYFORMAT message to its parent window to determine the type of structures to use in **WM_NOTIFY** messages. If the parent window does not handle this message, the **DefWindowProc** function responds according to the type of the parent window. That is, if the parent window is a Unicode window, **DefWindowProc** returns NFR_UNICODE, and if the parent window is an ANSI window, **DefWindowProc** returns NFR_ANSI. If the parent window is a dialog box and does not handle this message, the **DefDlgProc** function similarly responds according to the type of the dialog box (Unicode or ANSI).

A parent window can change the type of structures a common control uses in **WM_NOTIFY** messages by setting *lParam* to NF_REQUERY and sending a WM_NOTIFYFORMAT message to the control. This causes the control to send an NF_QUERY form of the WM_NOTIFYFORMAT message to the parent window.

All common controls will send WM_NOTIFYFORMAT messages. However, the standard Windows controls (edit controls, combo boxes, list boxes, buttons, scroll bars, and static controls) do not.

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.

Common API Macros

FORWARD_WM_NOTIFY

Sends or posts the **WM_NOTIFY** message.

```
VOID FORWARD_WM_NOTIFY(
  hwnd,
  idFrom,
  pnmhdr,
  fn
);
```

Parameters

hwnd

Handle to the window that receives the **WM_NOTIFY** message.

idFrom
Identifier of the control sending the message.

pnmhdr
Address of an **NMHDR** structure that contains the notification code and additional information. For some notification messages, this parameter points to a larger structure that has the **NMHDR** structure as its first member.

fn
Function that sends or posts the **WM_NOTIFY** message. This parameter can be either the **SendMessage** or **PostMessage** function.

Return Values

Returns a value whose meaning depends on the *fn* parameter.

Remarks

The **FORWARD_WM_NOTIFY** macro is defined as follows:

```
#define FORWARD_WM_NOTIFY(hwnd, idFrom, pnmhdr, fn) \
  (void)(fn)((hwnd), WM_NOTIFY, (WPARAM)(int)(id), \
  (LPARAM)(NMHDR FAR*)(pnmhdr))
```

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 commctrl.h.

HANDLE_WM_NOTIFY

Calls a function that processes the **WM_NOTIFY** message.

```
HANDLE_WM_NOTIFY(
  hwnd,
  wParam,
  lParam,
  fn
);
```

Parameters

hwnd
Handle to the window that received **WM_NOTIFY**.

wParam
First parameter of **WM_NOTIFY**.

lParam
Second parameter of **WM_NOTIFY**.

fn
Function that is to process **WM_NOTIFY**.

Return Values

Returns a value whose meaning depends on the *fn* parameter.

Remarks

The **HANDLE_WM_NOTIFY** macro is defined as follows:

```
#define HANDLE_WM_NOTIFY(hwnd, wParam, lParam, fn) \
  (fn)((hwnd), (int)(wParam), (NMHDR FAR*)(lParam))
```

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 commctrl.h.

INDEXTOSTATEIMAGEMASK

Prepares the index of a state image so that a tree-view control or list-view control can use the index to retrieve the state image for an item.

```
UINT INDEXTOSTATEIMAGEMASK(
  UINT i
);
```

Parameters

i
Index of a state image.

Remarks

The **INDEXTOSTATEIMAGEMASK** macro is defined as follows:

```
#define INDEXTOSTATEIMAGEMASK(i) ((i) << 12)
```

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

Common API Notifications

NM_CHAR

The NM_CHAR notification message is sent by a control when a character key is processed. This notification message is sent in the form of a **WM_NOTIFY** message.

```
NM_CHAR
  lpnmc = (LPNMCHAR) lParam;
```

Parameters

lpnmc
Pointer to an **NMCHAR** structure that contains additional information about the character that caused the notification message.

Return Values

The return value is ignored by most controls. For more information, see the documentation for the individual controls.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

See Also

NM_CHAR (toolbar)

NM_CLICK

Notifies a control's parent window that the user has clicked the left mouse button within the control. NM_CLICK is sent in the form of a **WM_NOTIFY** message.

```
NM_CLICK
    lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh

Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The return value is ignored by the control.

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 2.0 or later.
Header: Declared in winuser.h.

NM_DBLCLK

Notifies a control's parent window that the user has double-clicked the left mouse button within the control. NM_DBLCLK is sent in the form of a **WM_NOTIFY** message.

```
NM_DBLCLK
  lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh

Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The return value is ignored by the control.

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.

NM_HOVER

Sent by a control when the mouse hovers over an item. This notification message is sent in the form of a **WM_NOTIFY** message.

```
NM_HOVER
  lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh

Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

Unless otherwise specified, return zero to allow the control to process the hover normally, or nonzero to prevent the hover from being processed.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

NM_KEYDOWN

Sent by a control when the control has the keyboard focus and the user presses a key. This notification message is sent in the form of a **WM_NOTIFY** message.

```
NM_KEYDOWN
  lpnmk = (LPNMKEY) lParam;
```

Parameters

lpnmk

Address of an **NMKEY** structure that contains additional information about the key that caused the notification message.

Return Values

Return nonzero to prevent the control from processing the key, or zero otherwise.

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

NM_KILLFOCUS

Notifies a control's parent window that the control has lost the input focus. NM_KILLFOCUS is sent in the form of a **WM_NOTIFY** message.

```
NM_KILLFOCUS
  lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh
Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The return value is ignored by the control.

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.

NM_NCHITTEST

Sent by a control when the control receives a **WM_NCHITTEST** message. This notification message is sent in the form of a **WM_NOTIFY** message.

```
NM_NCHITTEST
  lpnmmouse = (LPNMMOUSE) lParam;
```

Parameters

lpnmmouse
Address of a **NMMOUSE** structure that contains information about the notification. The *pt* member contains the mouse coordinates of the hit test message.

Return Values

Unless otherwise specified, return zero to allow the control to perform default processing of the hit test message, or return one of the HT* values documented under **WM_NCHITTEST** to override the default hit test processing.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

NM_OUTOFMEMORY

Notifies a control's parent window that the control could not complete an operation because there was not enough memory available. NM_OUTOFMEMORY is sent in the form of a **WM_NOTIFY** message.

```
NM_OUTOFMEMORY
    lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh
Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The return value is ignored by the control.

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 commctrl.h.

NM_RCLICK

Notifies a control's parent window that the user has clicked the right mouse button within the control. NM_RCLICK is sent in the form of a **WM_NOTIFY** message.

```
NM_RCLICK
  lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh

Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The return value is ignored by the control.

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 commctrl.h.

NM_RDBLCLK

Notifies a control's parent window that the user has double-clicked the right mouse button within the control. NM_RDBLCLK is sent in the form of a **WM_NOTIFY** message.

```
NM_RDBLCLK
  lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh

Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The return value is ignored by the control.

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 commctrl.h.

NM_RELEASEDCAPTURE

Notifies a control's parent window that the control is releasing mouse capture. This notification is sent in the form of a **WM_NOTIFY** message.

```
NM_RELEASEDCAPTURE
  lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh

Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

Unless otherwise specified, the control ignores the return value from this notification.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

NM_RETURN

Notifies a control's parent window that the control has the input focus and that the user has pressed the ENTER key. NM_RETURN is sent in the form of a **WM_NOTIFY** message.

```
NM_RETURN
  lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh

Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The return value is ignored by the control.

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 commctrl.h.

NM_SETCURSOR

Notifies a control's parent window that the control is setting the cursor in response to a WM_SETCURSOR message. This notification is sent in the form of a **WM_NOTIFY** message.

```
NM_SETCURSOR
  lpnmm = (LPNMMOUSE) lParam;
```

Parameters

lpnmm
Address of an **NMMOUSE** structure that contains additional information about this notification message.

Return Values

Unless otherwise specified, return nonzero to allow the control to set the cursor or zero to prevent the control from setting the cursor.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

NM_SETFOCUS

Notifies a control's parent window that the control has received the input focus. NM_SETFOCUS is sent in the form of a **WM_NOTIFY** message.

```
NM_SETFOCUS
  lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh

Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The return value is ignored by the control.

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 commctrl.h.

NM_TOOLTIPSCREATED

Notifies a control's parent window that the control has created a tooltip control. This notification is sent in the form of a **WM_NOTIFY** message.

Syntax

```
NM_TOOLTIPSCREATED
  lpnmttc = (LPNMHDR) lParam;
```

Parameters

lpnmttc

Address of an **NMTOOLTIPSCREATED** structure that contains additional information about this notification message.

Return Value

Unless otherwise specified, the control ignores the return value from this notification.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

Common API Structures

COLORSCHEME

Contains information for the drawing of buttons in a toolbar or rebar.

```
typedef struct tagCOLORSCHEME{
    DWORD dwSize;
    COLORREF clrBtnHighlight;
    COLORREF clrBtnShadow;
} COLORSCHEME, *LPCOLORSCHEME;
```

Members

dwSize
Size of this structure, in bytes.

clrBtnHighlight
COLORREF value that represents the highlight color of the buttons. Use **CLR_DEFAULT** for the default highlight color.

clrBtnShadow
COLORREF value that represents the shadow color of the buttons. Use **CLR_DEFAULT** for the default shadow color.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

INITCOMMONCONTROLSEX

Carries information used to load common control classes from the dynamic-link library (DLL). This structure is used with the **InitCommonControlsEx** function.

```
typedef struct tagINITCOMMONCONTROLSEX {
    DWORD dwSize;
    DWORD dwICC;
} INITCOMMONCONTROLSEX, *LPINITCOMMONCONTROLSEX;
```

Members

dwSize
Size of the structure, in bytes.

dwICC
Set of bit flags that indicate which common control classes will be loaded from the DLL. This value can be a combination of the following:

ICC_ANIMATE_CLASS	Load animate control class.
ICC_BAR_CLASSES	Load toolbar, status-bar, trackbar, and tooltip control classes.
ICC_COOL_CLASSES	Load rebar control class.
ICC_DATE_CLASSES	Load date and time picker control class.
ICC_HOTKEY_CLASS	Load hot-key control class.
ICC_INTERNET_CLASSES	Load IP address class.
ICC_LISTVIEW_CLASSES	Load list-view and header control classes.
ICC_PAGESCROLLER_CLASS	Load pager control class.
ICC_PROGRESS_CLASS	Load progress bar control class.
ICC_TAB_CLASSES	Load tab and tooltip control classes.
ICC_TREEVIEW_CLASSES	Load tree-view and tooltip control classes.
ICC_UPDOWN_CLASS	Load up-down control class.
ICC_USEREX_CLASSES	Load ComboBoxEx class.
ICC_WIN95_CLASSES	Load animate control, header, hot-key, list-view, progress bar, status-bar, tab, tooltip, toolbar, trackbar, tree-view, and up-down control classes.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

NMCHAR

Contains information used with character notification messages.

```
typedef struct tagNMCHAR{
    NMHDR hdr;
    UINT ch;
    DWORD dwItemPrev;
    DWORD dwItemNext;
} NMCHAR, FAR *LPNMCHAR;
```

Members

hdr
NMHDR structure that contains additional information about this notification.

ch
Character that is being processed.

dwItemPrev
32-bit value that is determined by the control that is sending the notification.

dwItemNext
32-bit value that is determined by the control that is sending the notification.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

NMHDR

Contains information about a notification message.

```
typedef struct tagNMHDR {
    HWND hwndFrom;
    UINT idFrom;
    UINT code;
} NMHDR;
```

Members

hwndFrom
Window handle to the control sending a message.

idFrom
Identifier of the control sending a message.

code
Notification code. This member can be either a control-specific notification code or one of the common notification codes.

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.

NMKEY

Contains information used with key notification messages.

```
typedef struct tagNMKEY {
    NMHDR hdr;
    UINT nVKey;
    UINT uFlags;
} NMKEY, FAR *LPNMKEY;
```

Members

hdr
NMHDR structure that contains additional information about this notification.

nVKey
Virtual key code of the key that caused the event.

uFlags
Flags associated with the key. These are the same flags that are passed in the high word of the *lParam* parameter of the **WM_KEYDOWN** message.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

NMMOUSE

Contains information used with mouse notification messages.

```
typedef struct tagNMMOUSE {
    NMHDR   hdr;
    DWORD   dwItemSpec;
    DWORD   dwItemData;
    POINT   pt;
    LPARAM  dwHitInfo;
} NMMOUSE, FAR* LPNMMOUSE;
```

Members

hdr
NMHDR structure that contains additional information about this notification.

dwItemSpec
Control-specific item identifier.

dwItemData
Control-specific item data.

pt
POINT structure that contains the screen coordinates of the mouse when the click occurred.

dwHitInfo
Carries information about where on the item or control the cursor is pointing.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

NMOBJECTNOTIFY

Contains information used with the **TBN_GETOBJECT**, **TCN_GETOBJECT**, and **PSN_GETOBJECT** notification messages.

```
typedef struct tagNMOBJECTNOTIFY {
    NMHDR     hdr;
    int       iItem;
    IID*      piid;
    IUnknown* pObject;
    HRESULT   hResult;
} NMOBJECTNOTIFY, FAR* LPNMOBJECTNOTIFY;
```

Members

hdr
NMHDR structure that contains additional information about this notification.

iItem
Control-specific item identifier. This value will comply to item identification standards for the control sending the notification. However, this member is not used with the **PSN_GETOBJECT** notification message.

piid
Interface identifier of the requested object.

pObject
Address of an object provided by the window processing the notification message. The application processing the notification message sets this member.

hResult
COM success or failure flags. The application processing the notification message sets this member.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

NMTOOLTIPSCREATED

Contains information used with **NM_TOOLTIPSCREATED** notification messages.

```
typedef struct tagNMTOOLTIPSCREATED {
    NMHDR hdr;
    HWND hwndToolTips;
} NMTOOLTIPSCREATED, * LPNMTOOLTIPSCREATED;
```

Parameters

Members

hdr
NMHDR structure that contains additional information about this notification.

hwndToolTips
Window handle to the tooltip control created.

Requirements

Version 5.80 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later installed).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

CHAPTER 8

Customizing a Control's Appearance

Custom draw is not a common control; it is a service that many common controls provide. Custom draw services allow an application greater flexibility in customizing a control's appearance. Your application can harness custom draw notifications to easily change the font used to display items or manually draw an item without having to do a full-blown owner draw.

About Custom Draw

This section contains general information about custom draw functionality and provides a conceptual overview of how an application can support custom draw.

Currently, the following controls support custom draw functionality:

- Header controls
- List-view controls
- Rebar controls
- Toolbar controls
- Tooltip controls
- Trackbar controls
- Tree-view controls

Note Custom draw is implemented in version 4.70 and later of Comctl32.dll.

About Custom Draw Notification Messages

All common controls that support custom draw send **NM_CUSTOMDRAW** notification messages at specific points during drawing operations. These notifications describe drawing operations that apply to the entire control as well as drawing operations specific to items within the control. Like many notification messages, NM_CUSTOMDRAW notifications are sent as **WM_NOTIFY** messages.

The *lParam* parameter of a custom draw notification message will be the address of an **NMCUSTOMDRAW** structure or a control-specific structure that contains an **NMCUSTOMDRAW** structure as its first member. The following table illustrates the relationship between the controls and the structures they use:

Structure	Used by
NMCUSTOMDRAW	Rebar, trackbar, and header controls
NMLVCUSTOMDRAW	List-view controls
NMTBCUSTOMDRAW	Toolbar controls
NMTTCUSTOMDRAW	Tooltip controls
NMTVCUSTOMDRAW	Tree-view controls

Paint Cycles, Drawing Stages, and Notification Messages

Like all Microsoft Windows applications, common controls periodically paint and erase themselves based on messages received from the system or other applications. The process of a control painting or erasing itself is called a *paint cycle*. Controls that support custom draw send **NM_CUSTOMDRAW** notification messages periodically through each paint cycle. This notification message is accompanied by an **NMCUSTOMDRAW** structure or another structure that contains an **NMCUSTOMDRAW** structure as its first member.

One piece of information that the **NMCUSTOMDRAW** structure contains is the current stage of the paint cycle. This is referred to as the *draw stage* and is represented by the value in the structure's **dwDrawStage** member. A control informs its parent about four basic draw stages. These basic, or global, draw stages are represented in the structure by the following flag values (defined in Commctrl.h):

Global draw stage values	Description
CDDS_POSTERASE	After the erase cycle is complete
CDDS_POSTPAINT	After the paint cycle is complete
CDDS_PREERASE	Before the erase cycle begins
CDDS_PREPAINT	Before the paint cycle begins

Each of the preceding values can be combined with the CDDS_ITEM flag to specify draw stages specific to items. For convenience, Commctrl.h contains the following item-specific values:

Item-specific draw stage values	Description
CDDS_ITEMPOSTERASE	After an item has been erased.
CDDS_ITEMPOSTPAINT	After an item has been drawn.
CDDS_ITEMPREERASE	Before an item is erased.
CDDS_ITEMPREPAINT	Before an item is drawn.
CDDS_SUBITEM	**Version 4.71**. Flag combined with CDDS_ITEMPREPAINT or CDDS_ITEMPOSTPAINT if a subitem is being drawn. This will only be set if CDRF_NOTIFYITEMDRAW is returned from CDDS_PREPAINT.

Your application must process the **NM_CUSTOMDRAW** notification message and then return a specific value that informs the control what it must do. See the following sections for more information about these return values.

Taking Advantage of Custom Draw Services

The key to harnessing custom draw functionality is in responding to the **NM_CUSTOMDRAW** notification messages that a control sends. The return values your application sends in response to these notifications determine the control's behavior for that paint cycle.

This section contains information about how your application can use **NM_CUSTOMDRAW** notification return values to determine the control's behavior.

Responding to the Prepaint Notification

At the beginning of each paint cycle, the control sends the **NM_CUSTOMDRAW** notification message, specifying the CDDS_PREPAINT value in the **dwDrawStage** member of the accompanying **NMCUSTOMDRAW** structure. The value that your application returns to this first notification dictates how and when the control sends subsequent custom draw notifications for the rest of that paint cycle. Your application can return a combination of the following flags in response to the first notification:

Return value	Effect
CDRF_DODEFAULT	The control will draw itself. It will not send additional **NM_CUSTOMDRAW** notifications for this paint cycle. This flag cannot be used with any other flag.
CDRF_NOTIFYITEMDRAW	The control will notify the parent of any item-specific drawing operations. It will send **NM_CUSTOMDRAW** notification messages before and after it draws items.
CDRF_NOTIFYPOSTPAINT	The control will send an **NM_CUSTOMDRAW** notification when the painting cycle for the entire control is complete.
CDRF_SKIPDEFAULT	The control will not perform any painting at all.

Requesting Item-Specific Notifications

If your application returns CDRF_NOTIFYITEMDRAW to the initial prepaint custom draw notification, the control will send notifications for each item it draws during that paint cycle. These item-specific notifications will have the CDDS_ITEMPREPAINT value in the **dwDrawStage** member of the accompanying **NMCUSTOMDRAW** structure. You can request that the control send another notification when it is finished drawing the item by returning CDRF_NOTIFYPOSTPAINT to these item-specific notifications. Otherwise, return CDRF_DODEFAULT and the control will not notify the parent window until it starts to draw the next item.

Drawing the Item Manually

If your application draws the entire item, return CDRF_SKIPDEFAULT. This allows the control to skip items that it does not need to draw, thereby decreasing system overhead. Keep in mind that returning this value means the control will not draw any portion of the item.

Changing Fonts and Colors

Your application can use custom draw to change an item's font. Simply, select the HFONT you want into the device context specified by the **hdc** member of the **NMCUSTOMDRAW** structure associated with the custom draw notification. Since the font you select might have different metrics than the default font, make sure you include the CDRF_NEWFONT bit in the return value for the notification message. For more information on using this functionality, see the sample code in *Using Custom Draw*. The font that your application specifies is used to display that item when it is not selected. Custom draw does not allow you to change the font attributes for selected items.

To change text colors for all controls that support custom draw, except for the list view and tree view, set the desired text and background colors in the device context supplied in the custom draw notification structure with the **SetTextColor** and **SetBkColor** functions. To modify the text colors in the list view or tree view, you need to place the desired color values in the **clrText** and **clrTextBk** members of the **NMLVCUSTOMDRAW** or **NMTVCUSTOMDRAW** structure.

Custom Draw with List-View and Tree-View Controls

Most common controls can be handled in essentially the same way. However, the list-view and tree-view controls have some features that require a somewhat different approach to custom draw.

For Version 5.0 of the common controls, these two controls might display clipped text if you change the font by returning CDRF_NEWFONT. This behavior is necessary for backward compatibility with earlier versions of the common controls. If you want to change the font of a list-view or tree-view control, you will get better results if you send a CCM_SETVERSION message with the *wParam* value set to 5 before adding any items to the control.

Custom Draw with List-View Controls

Because list-view controls have subitems and multiple display modes, you will need to handle the **NM_CUSTOMDRAW** notification somewhat differently than for the other common controls.

For report mode:

1. The first **NM_CUSTOMDRAW** notification will have the **dwDrawStage** member of the associated **NMCUSTOMDRAW** structure set to CDDS_PREPAINT. Return CDRF_NOTIFYITEMDRAW.
2. You will then receive an **NM_CUSTOMDRAW** notification with **dwDrawStage** set to CDDS_ITEMPREPAINT. If you specify new fonts or colors and return CDRF_NEWFONT, all subitems of the item will be changed. If you instead want to handle each subitem separately, return CDRF_NOTIFYSUBITEMDRAW.
3. If you returned CDRF_NOTIFYITEMDRAW in the previous step, you will then receive an **NM_CUSTOMDRAW** notification for each subitem with **dwDrawStage** set to CDDS_SUBITEM | CDDS_PREPAINT. To change the font or color for that subitem, specify a new font or color and return CDRF_NEWFONT.

For the large icon, small icon, and list modes:

1. The first **NM_CUSTOMDRAW** notification will have the **dwDrawStage** member of the associated **NMCUSTOMDRAW** structure set to CDDS_PREPAINT. Return CDRF_NOTIFYITEMDRAW.
2. You will then receive an **NM_CUSTOMDRAW** notification with **dwDrawStage** set to CDDS_ITEMPREPAINT. You can change the fonts or colors of an item by specifying new fonts and colors and returning CDRF_NEWFONT. Because these modes do not have subitems, you will not receive any additional NM_CUSTOMDRAW notifications.

An example of a list view NM_CUSTOMDRAW notification handler is given in the next section.

Using Custom Draw

The following code fragment is a portion of a WM_NOTIFY handler that illustrates how to handle custom draw notifications sent to a list view control:

```
LPNMLISTVIEW  pnm    = (LPNMLISTVIEW)lParam;

switch (pnm->hdr.code){
...
case NM_CUSTOMDRAW:

  LPNMLVCUSTOMDRAW  lplvcd = (LPNMLVCUSTOMDRAW)lParam;

  switch(lplvcd->nmcd.dwDrawStage) {
  case CDDS_PREPAINT :
      return CDRF_NOTIFYITEMDRAW;

  case CDDS_ITEMPREPAINT:
      SelectObject(lplvcd->nmcd.hdc,
                   GetFontForItem(lplvcd->nmcd.dwItemSpec,
```

(continued)

(continued)

```
                                        lplvcd->nmcd.lItemlParam) );
      lplvcd->clrText = GetColorForItem(lplvcd->nmcd.dwItemSpec,
                                        lplvcd->nmcd.lItemlParam);
      lplvcd->clrTextBk = GetBkColorForItem(lplvcd->nmcd.dwItemSpec,
                                            lplvcd->nmcd.lItemlParam);

/* At this point, you can change the background colors
for the item and any subitems and return CDRF_NEWFONT.
If the list-view control is in report mode, you can
return CDRF_NOTIFYSUBITEMREDRAW to customize the item's
subitems individually */

      ...
      return CDRF_NEWFONT;
      //  or return CDRF_NOTIFYSUBITEMREDRAW;

  case CDDS_SUBITEM | CDDS_PREPAINT:
      SelectObject(lplvcd->nmcd.hdc,
                   GetFontForSubItem(lplvcd->nmcd.dwItemSpec,
                                     lplvcd->nmcd.lItemlParam,
                                     lplvcd->iSubItem));
      lplvcd->clrText = GetColorForSubItem(lplvcd->nmcd.dwItemSpec,
                                           lplvcd->nmcd.lItemlParam,
                                           lplvcd->iSubItem));
      lplvcd->clrTextBk = GetBkColorForSubItem(lplvcd->nmcd.dwItemSpec,
                                               lplvcd->nmcd.lItemlParam,
                                               lplvcd->iSubItem));

/* This notification is received only if you are in report
mode and returned CDRF_NOTIFYSUBITEMREDRAW in the previous
step. At this point, you can change the background colors
for the subitem and return CDRF_NEWFONT.*/

      ...
      return CDRF_NEWFONT;
  }
...
}
```

The first **NM_CUSTOMDRAW** notification has the **dwDrawStage** member of the **NMCUSTOMDRAW** structure set to CDDS_PREPAINT. The handler returns CDRF_NOTIFYITEMDRAW to indicate that it wishes to modify one or more items individually. The control then sends an NM_CUSTOMDRAW notification with

dwDrawStage set to CDDS_PREPAINT for each item. The handler returns CDRF_NOTIFYITEMDRAW to indicate that it wishes to modify the item.

If CDRF_NOTIFYITEMDRAW was returned in the previous step, the next NM_CUSTOMDRAW notification has **dwDrawStage** set to CDDS_ITEMPREPAINT. The handler gets the current color and font values. At this point, you can specify new values for small icon, large icon, and list modes. If the control is in report mode, you can also specify new values that will apply to all subitems of the item. If you have changed anything, return CDRF_NEWFONT. If the control is in report mode and you want to handle the subitems individually, return CDRF_NOTIFYSUBITEMREDRAW.

The final notification is only sent if the control is in report mode and you returned CDRF_NOTIFYSUBITEMREDRAW in the previous step. The procedure for changing fonts and colors is the same as that step, but it only applies to a single subitem. Return CDRF_NEWFONT to notify the control if the color or font was changed.

Custom Draw Reference

Custom Draw Notification Messages

NM_CUSTOMDRAW

Sent by some common controls to notify their parent windows about drawing operations. This notification is sent in the form of a **WM_NOTIFY** message.

```
NM_CUSTOMDRAW
#ifdef LIST_VIEW_CUSTOM_DRAW
    lpNMCustomDraw = (LPNMLVCUSTOMDRAW) lParam;
#elif TOOL_TIPS_CUSTOM_DRAW
    lpNMCustomDraw = (LPNMTTCUSTOMDRAW) lParam;
#elif TREE_VIEW_CUSTOM_DRAW
    lpNMCustomDraw = (LPNMTVCUSTOMDRAW) lParam;
#elif TOOL_BAR_CUSTOM_DRAW
    lpNMCustomDraw = (LPNMTBCUSTOMDRAW) lParam;
#else
    lpNMCustomDraw = (LPNMCUSTOMDRAW) lParam;
#endif
```

Parameters

lpNMCustomDraw

Address of a custom draw-related structure that contains information about the drawing operation. The following table lists the controls and their associated structures:

Control	Structure
List-view	**NMLVCUSTOMDRAW**
Toolbar	**NMTBCUSTOMDRAW**
Tooltip	**NMTTCUSTOMDRAW**
Tree-view	**NMTVCUSTOMDRAW**
All other supported controls	**NMCUSTOMDRAW**

Return Values

The value your application can return depends on the current drawing stage. The **dwDrawStage** member of the associated **NMCUSTOMDRAW** structure holds a value that specifies the drawing stage. You must return one of the following values.

When **dwDrawStage** equals CDDS_PREPAINT:

Return value	Description
CDRF_DODEFAULT	The control will draw itself. It will not send any additional NM_CUSTOMDRAW messages for this paint cycle.
CDRF_NOTIFYITEMDRAW	The control will notify the parent of any item-related drawing operations. It will send NM_CUSTOMDRAW notification messages before and after drawing items.
CDRF_NOTIFYPOSTERASE	The control will notify the parent after erasing an item.
CDRF_NOTIFYPOSTPAINT	The control will notify the parent after painting an item.

When **dwDrawStage** equals CDDS_ITEMPREPAINT:

Return value	Description
CDRF_NEWFONT	Your application specified a new font for the item; the control will use the new font. For more information on changing fonts, see **Changing Fonts and Colors**.
CDRF_NOTIFYSUBITEMDRAW	**Version 4.71**. Your application will receive an **NM_CUSTOMDRAW** message with dwDrawStage set to CDDS_ITEMPREPAINT | CDDS_SUBITEM before each list view subitem is drawn. You then can specify font and color for each subitem separately or return CDRF_DODEFAULT for default processing.
CDRF_SKIPDEFAULT	Your application drew the item manually. The control will not draw the item.

Remarks

Currently, the following controls support custom draw functionality: header, list-view, rebar, toolbar, tooltip, trackbar, and tree-view.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

See Also

Using Custom Draw

Custom Draw Structures

NMCUSTOMDRAW

Contains information specific to an **NM_CUSTOMDRAW** notification message.

```
typedef struct tagNMCUSTOMDRAWINFO {
    NMHDR  hdr;
    DWORD  dwDrawStage;
    HDC    hdc;
    RECT   rc;
    DWORD  dwItemSpec;
    UINT   uItemState;
    LPARAM lItemlParam;
} NMCUSTOMDRAW, FAR * LPNMCUSTOMDRAW;
```

Members

hdr
NMHDR structure that contains information about this notification message.

dwDrawStage
Current drawing stage. This value is one of the following:

Global values	Description
CDDS_POSTERASE	After the erasing cycle is complete

(continued)

(continued)

Global values	Description
CDDS_POSTPAINT	After the painting cycle is complete
CDDS_PREERASE	Before the erasing cycle begins
CDDS_PREPAINT	Before the painting cycle begins

Item-specific values	Description
CDDS_ITEM	Indicates that the **dwItemSpec**, **uItemState**, and **lItemlParam** members are valid.
CDDS_ITEMPOSTERASE	After an item has been erased.
CDDS_ITEMPOSTPAINT	After an item has been drawn.
CDDS_ITEMPREERASE	Before an item is erased.
CDDS_ITEMPREPAINT	Before an item is drawn.
CDDS_SUBITEM	**Version 4.71**. Flag combined with CDDS_ITEMPREPAINT or CDDS_ITEMPOSTPAINT if a subitem is being drawn. This will be set only if CDRF_NOTIFYITEMDRAW is returned from CDDS_PREPAINT.

hdc
Handle to the control's device context. Use this HDC to perform any GDI functions.

rc
RECT structure that describes the bounding rectangle of the area being drawn. This member is initialized only by the CDDS_ITEMPREPAINT notification.

Version 5.80. This member is initialized also by the CDDS_PREPAINT notification.

dwItemSpec
Item number. What is contained in this member will depend on the type of control that is sending the notification. See the **NM_CUSTOMDRAW** notification reference for the specific control to determine what, if anything, is contained in this member.

uItemState
Current item state. This value is a combination of the following:

Value	Description
CDIS_CHECKED	The item is checked.
CDIS_DEFAULT	The item is in its default state.
CDIS_DISABLED	The item is disabled.
CDIS_FOCUS	The item is in focus.
CDIS_GRAYED	The item appears dimmed.
CDIS_HOT	The item is currently under the pointer ("hot").
CDIS_INDETERMINATE	The item is in an indeterminate state.

Value	Description
CDIS_MARKED	The item is marked. The meaning of this is up to the implementation.
CDIS_SELECTED	The item is selected.

lItemlParam
Application-defined item data.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

CHAPTER 9

Animation Controls

Animation Control Overview

An *animation control* is a window that displays an AVI clip. An AVI clip is a series of bitmap frames like a movie. Animation controls can only display AVI clips that do not contain audio.

About Animation Controls

One common use for an animation control is to indicate system activity during a lengthy operation. This is possible because the operation thread continues executing while the AVI clip is displayed. For example, the **Find** dialog box of Microsoft Windows Explorer displays a moving magnifying glass as the system searches for a file.

An animation control can display an AVI clip originating from either an uncompressed AVI file or from an AVI file that was compressed using run-length (BI_RLE8) encoding. You can add the AVI clip to your application as an AVI resource, or the clip can accompany your application as a separate AVI file.

Note The AVI file, or resource, must not have a sound channel. The capabilities of the animation control are very limited and are subject to change. If you need a control to provide multimedia playback and recording capabilities for your application, you can use the MCIWnd control. For more information about the MCIWnd control, see the multimedia documentation in the Platform SDK.

Animation Control Creation

An animation control belongs to the ANIMATE_CLASS window class. You create an animation control by using the **CreateWindow** function or the **Animate_Create** macro. The macro positions the animation control in the upper-left corner of the parent window and, if the **ACS_CENTER** style is not specified, sets the width and height of the control based on the dimensions of a frame in the AVI clip. If **ACS_CENTER** is specified, **Animate_Create** sets the width and height of the control to zero. You can use the **SetWindowPos** function to set the position and size of the control.

If you create an animation control within a dialog box or from a dialog box resource, the control is automatically destroyed when the user closes the dialog box. If you create an animation control within a window, you must explicitly destroy the control.

About Animation Control Messages

An application sends messages to an animation control to open, play, stop, and close the corresponding AVI clip. Each message has one or more macros that you can use instead of sending the message explicitly.

After creating an animation control, an application sends the **ACM_OPEN** message to open an AVI clip and load it into memory. The message specifies either the path of an AVI file or the name of an AVI resource. The system loads the AVI resource from the module that created the animation control.

If the animation control has the **ACS_AUTOPLAY** style, the control begins playing the AVI clip immediately after the AVI file or AVI resource is opened. Otherwise, an application can use the **ACM_PLAY** message to start the AVI clip. An application can stop the clip at any time by sending the **ACM_STOP** message. The last frame played remains displayed when the control finishes playing the AVI clip or when **ACM_STOP** is sent.

An animation control can send two notification messages, **ACN_START** and **ACN_STOP**, to its parent window. Most applications do not handle either notification.

To close the AVI file or AVI resource and remove it from memory, an application can use the **Animate_Close** macro, which sends **ACM_OPEN** with the file name or resource name set to NULL.

Default Message Processing

This section describes the window messages handled by the window procedure for the ANIMATE_CLASS window class.

Message	Processing performed
WM_CLOSE	Frees the AVI file or AVI resource associated with the animation control.
WM_DESTROY	Frees the AVI file or AVI resource, frees an internal data structure, and then calls the **DefWindowProc** function.
WM_ERASEBKGND	Erases the window background using the current background color for static controls.
WM_NCCREATE	Allocates and initializes an internal data structure and then calls **DefWindowProc**.
WM_NCHITTEST	Returns the HTTRANSPARENT hit-test value.
WM_PAINT	Draws an AVI frame in the animation control.
WM_SIZE	Checks if the control has the **ACS_CENTER** style. If the control does not, it calls **DefWindowProc**. Otherwise, it centers the animation in the control, invalidates the control, and then calls **DefWindowProc**.

Using Animation Controls

This section provides examples that demonstrate how to create an animation control and display an AVI clip in the control.

Creating an Animation Control

The following function creates an animation control in a dialog box. The animation control is positioned below the specified control, and the dimensions of the animation control are based on the dimensions of a frame in the AVI clip.

```
// CreateAnimationCtrl - creates an animation control,
//     positions it below the specified control in a
//     dialog box, and opens the AVI clip for the
//     animation control.
// Returns the handle to the animation control.
// hwndDlg - handle to the dialog box.
// nIDCtl - identifier of the control below which the
//          animation control is to be positioned.
//
// Constants
//     IDC_ANIMATE - identifier of the animation control.
//     CX_FRAME, CY_FRAME - width and height of the frames
//     in the AVI clip.
HWND CreateAnimationCtrl(HWND hwndDlg, int nIDCtl)
{
    HWND hwndAnim = NULL;
    RECT rc;
    POINT pt;

    // Create the animation control.
    hwndAnim = Animate_Create(hwndDlg, IDC_ANIMATE,
        WS_BORDER | WS_CHILD, g_hinst);

    // Get the screen coordinates of the specified control
    // button.
    GetWindowRect(GetDlgItem(hwndDlg, nIDCtl), &rc);

    // Convert the coordinates of the lower-left corner to
    // client coordinates.
    pt.x = rc.left;
    pt.y = rc.bottom;
    ScreenToClient(hwndDlg, &pt);

    // Position the animation control below the Stop button.
    SetWindowPos(hwndAnim, 0, pt.x, pt.y + 20,
```

(continued)

(continued)

```
        CX_FRAME, CY_FRAME,
        SWP_NOZORDER | SWP_DRAWFRAME);

    // Open the AVI clip, and show the animation control.
    Animate_Open(hwndAnim, "SEARCH.AVI");
    ShowWindow(hwndAnim, SW_SHOW);

    return hwndAnim;
}
```

Controlling the AVI Clip

The following function uses the animation control macros to control the display of the AVI clip in an animation control.

```
// DoAnimation - plays, stops, or closes an animation
//      control's AVI clip, depending on the value of an
//      action flag.
// hwndAnim - handle to an animation control
// nAction - flag that determines whether to play, stop,
//           or close the AVI clip.
void DoAnimation(HWND hwndAnim, int nAction)
{
    switch (nAction) {
        case PLAYIT:

            // Play the clip continuously starting with the
            // first frame.
            Animate_Play(hwndAnim, 0, -1, -1);
            break;

        case STOPIT:
            Animate_Stop(hwndAnim);
            break;

        case CLOSEIT:
            Animate_Close(hwndAnim);
            break;

        default:
            break;
    }
    return;
}
```

Animation Control Styles

The following window styles are used with animation controls:

ACS_AUTOPLAY Starts playing the animation as soon as the AVI clip is opened.

ACS_CENTER Centers the animation in the animation control's window.

ACS_TIMER By default, the control creates a thread to play the AVI clip. If you set this flag, the control plays the clip without creating a thread; internally, the control uses a Win32 timer to synchronize playback.

ACS_TRANSPARENT Allows you to match an animation's background color to that of the underlying window, creating a "transparent" background. The control will send a **WM_CTLCOLORSTATIC** message to its parent. Use **SetBkColor** to set the background color for the device context to an appropriate value. The control interprets the upper-left pixel of the first frame as the animation's default background color. It will remap all pixels with that color to the value you supplied in response to WM_CTLCOLORSTATIC.

Animation Control Reference

Animation Control Messages

ACM_OPEN

Opens an AVI clip and displays its first frame in an animation control. You can send this message explicitly or use the **Animate_Open** or **Animate_OpenEx** macro.

```
ACM_OPEN
#if (_WIN32_IE >= 0x0400)
    wParam = (WPARAM)(HINSTANCE)hinst;
#else
    wParam = 0;
#endif
    lParam = (LPARAM) (LPSTR) lpszName;
```

Parameters

hinst

Version 4.71. Instance handle to the module that the resource should be loaded from. Set this value to NULL to have the control use the HINSTANCE value used to create

the window. Note that if the window is created by a DLL, the default value for *hinst* is the HINSTANCE value of the DLL, not the application that calls the DLL.

lpszName

Address of a buffer that contains the path of the AVI file or the name of an AVI resource. Alternatively, this parameter can consist of the AVI resource identifier in the low-order word and zero in the high-order word. To create this value, use the **MAKEINTRESOURCE** macro. The control loads the AVI resource from the module specified by the instance handle passed to the **CreateWindow** function, the **Animate_Create** macro, or the dialog-box creation function that created the control. In **Version 4.71** and later, the resource is loaded from the module specified by *hinst*. An AVI resource must have the "AVI" type.

If this parameter is NULL, the system closes the AVI file that was opened previously for the specified animation control, if any.

Return Values

Returns nonzero if successful, or zero otherwise.

Remarks

The AVI file or resource specified by *lpszName* must not contain audio.

With Windows 95, the animation control only responds to the ANSI version of the message (ACM_OPENA) with an ANSI string for *lpszName*. The Unicode version, ACM_OPENW, will fail.

You can only open silent AVI clips. ACM_OPEN and **Animate_Open** fail if *lpszSource* specifies an AVI clip that contains sound.

You can use **Animate_Close** to close an AVI file or AVI resource that was opened previously for the specified animation control.

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 commctrl.h.

ACM_PLAY

Plays an AVI clip in an animation control. The control plays the clip in the background while the thread continues executing. You can send this message explicitly or by using the **Animate_Play** macro.

```
ACM_PLAY
    wParam = (WPARAM) (UINT) cRepeat;
    lParam = (LPARAM) MAKELONG(wFrom, wTo);
```

Parameters

cRepeat
Number of times to replay the AVI clip. A value of –1 means replay the clip indefinitely.

wFrom
Zero-based index of the frame where playing begins. The value must be less than 65,536. A value of zero means begin with the first frame in the AVI clip.

wTo
Zero-based index of the frame where playing ends. The value must be less than 65,536. A value of –1 means end with the last frame in the AVI clip.

Return Values

Returns nonzero if successful, or zero otherwise.

Remarks

You can use **Animate_Seek** to direct the animation control to display a particular frame of the AVI clip.

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 commctrl.h.

ACM_STOP

Stops playing an AVI clip in an animation control. You can send this message either explicitly or by using the **Animate_Stop** macro.

```
ACM_STOP
    wParam = 0;
    lParam = 0;
```

Return Values

Returns nonzero if successful, 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 commctrl.h.

Animation Control Macros

Animate_Close

Closes an AVI clip and displays its first frame in an animation control. You can use this macro or send the **ACM_OPEN** message explicitly.

```
BOOL Animate_Close(
  HWND hwnd
);
```

Parameters

hwnd
Handle to the animation control.

Return Values

Always returns FALSE.

Remarks

You can use **Animate_Close** to close an AVI file or AVI resource that was opened previously for the specified animation control.

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 commctrl.h.

See Also

Animate_Open, MAKEINTRESOURCE

Animate_Create

Creates an animation control. **Animate_Create** calls the **CreateWindow** function to create the animation control.

```
HWND Animate_Create(
  HWND hwndP,
  UINT id,
  DWORD dwStyle,
  HINSTANCE hInstance
);
```

Parameters

hwndP
Handle to the parent window.

id
Child window identifier of the animation control.

dwStyle
Window styles. For a list of the animation control style values, see *Animation Control Styles*.

hInstance
Handle to the instance of the module that is creating the animation control.

Return Values

Returns the handle to the animation control.

Remarks

The **Animate_Create** macro sets the width and height of the animation control to zero if the **ACS_CENTER** style is specified. If the **ACS_CENTER** style is not specified, **Animate_Create** sets the width and height based on the dimensions of a frame in the AVI clip.

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 commctrl.h.

Animate_Open

Opens an AVI clip and displays its first frame in an animation control. You can use this macro or send the **ACM_OPEN** message explicitly.

```
BOOL Animate_Open(
  HWND hwnd,
  LPSTR lpszName
);
```

Parameters

hwnd
Handle to the animation control.

lpszName
Address of a buffer that contains the path of the AVI file or the name of an AVI resource. Alternatively, this parameter can consist of the AVI resource identifier in

the low-order word and zero in the high-order word. To create this value, use the **MAKEINTRESOURCE** macro. The control loads an AVI resource from the module specified by the instance handle passed to the **CreateWindow** function, the **Animate_Create** macro, or the dialog-box creation function that created the control.

The AVI file or resource specified by *lpszName* must not contain audio.

If this parameter is NULL, the system closes the AVI file that was opened previously for the specified animation control, if any.

Return Values

Returns nonzero if successful, or zero otherwise.

Remarks

You can only open silent AVI clips. **ACM_OPEN** and **Animate_Open** will fail if *lpszSource* specifies an AVI clip that contains sound.

You can use **Animate_Close** to close an AVI file or AVI resource that was opened previously for the specified animation control.

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 commctrl.h.

Animate_OpenEx

Opens an AVI clip from a resource in a specified module and displays its first frame in an animation control. You can use this macro or send the **ACM_OPEN** message explicitly.

```
BOOL Animate_OpenEx(
  HWND hwnd,
  HINSTANCE hinst,
  LPSTR lpszName
);
```

Parameters

hwnd

Handle to the animation control.

hinst

Instance handle to the module from which the resource should be loaded. If this value is NULL, the resource is loaded from the module that created the animation control.

lpszName

Address of a buffer that contains the path of the AVI file or the name of an AVI resource. Alternatively, this parameter can consist of the AVI resource identifier in the low-order word and zero in the high-order word. To create this value, use the **MAKEINTRESOURCE** macro. The control loads the AVI resource from the module specified by *hinst*.

The AVI file or resource specified by *lpszName* must not contain audio.

If this parameter is NULL, the system closes the AVI file that was opened previously for the specified animation control, if any.

Return Values

Returns nonzero if successful, or zero otherwise.

Remarks

You can only open silent AVI clips. **ACM_OPEN** and **Animate_Open** will fail if *lpszSource* specifies an AVI clip that contains sound.

You can use **Animate_Close** to close an AVI file or AVI resource that was previously opened for the specified animation control.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

Animate_Play

Plays an AVI clip in an animation control. The control plays the clip in the background while the thread continues executing. You can use this macro or send the **ACM_PLAY** message explicitly.

```
BOOL Animate_Play(
  HWND hwndAnim,
  UINT wFrom,
  UINT wTo,
  UINT cRepeat
);
```

Parameters

hwndAnim
Handle to the animation control in which to play the AVI clip.

wFrom
Zero-based index of the frame where playing begins. The value must be less than 65,536. A value of zero means begin with the first frame in the AVI clip.

wTo
Zero-based index of the frame where playing ends. The value must be less than 65,536. A value of –1 means end with the last frame in the AVI clip.

cRepeat
Number of times to replay the AVI clip. A value of –1 means replay the clip indefinitely.

Return Values

Returns nonzero if successful, or zero otherwise.

Remarks

You can use **Animate_Seek** to direct the animation control to display a particular frame of the AVI clip.

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 commctrl.h.

Animate_Seek

Directs an animation control to display a particular frame of an AVI clip. The control displays the clip in the background while the thread continues executing. You can use this macro or send the **ACM_PLAY** message explicitly.

```
BOOL Animate_Seek(
  HWND hwndAnim,
  UINT wFrame
);
```

Parameters

hwndAnim
Handle to the animation control in which to display the AVI frame.

wFrame
Zero-based index of the frame to display.

Return Values

Returns nonzero if successful, or zero otherwise.

Windows NT/2000: Requires Windows NT 3.51 or later
Windows 95/98: Requires Windows 95 or later
Windows CE: Unsupported.
Header: Declared in commctrl.h.

Animate_Play

Animate_Stop

Stops playing an AVI clip in an animation control. You can use this macro or send the **ACM_STOP** message explicitly.

```
BOOL Animate_Stop(
  HWND hwnd
);
```

Parameters

hwnd
Handle to the animation control.

Return Values

Returns nonzero if successful, or zero otherwise.

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in commctrl.h.

Animation Control Notifications

ACN_START

Notifies an animation control's parent window that the associated AVI clip has started playing. This notification message is sent in the form of a **WM_COMMAND** message.

```
ACN_START
```

Return Values

The return value is ignored.

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 commctrl.h.

ACN_STOP

Notifies an animation control's parent window that the associated AVI clip has stopped playing. This notification message is sent in the form of a **WM_COMMAND** message.

```
ACN_STOP
```

Return Values

The return value is ignored.

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 commctrl.h.

CHAPTER 10

ComboBoxEx Controls

A *ComboBoxEx control* is an extension of the combo box control that provides native support for item images. To make item images easily accessible, the control provides image list support. By using this control, you can provide the functionality of a combo box without having to manually draw item graphics.

About ComboBoxEx Controls

Effectively, a ComboBoxEx control creates a child combo box and performs owner draw tasks for you based on an assigned image list. Therefore, the CBS_OWNERDRAWFIXED style is implied, and you do not need to use it when creating the control. Because image lists are used to provide item graphics, the CBS_OWNERDRAWVARIABLE style cannot be used.

A ComboBoxEx control must be initialized by calling the **InitCommonControlsEx** function, specifying ICC_USEREX_CLASSES in the accompanying **INITCOMMONCONTROLSEX** structure.

You can create a ComboBoxEx control by using the **CreateWindowEx** function and specifying WC_COMBOBOXEX as the window class. The class is registered when the **InitCommonControlsEx** function is called as explained above.

ComboBoxEx controls are created without a default image list. To use item images, you must create an image list for the ComboBoxEx control and assign it to the control using the **CBEM_SETIMAGELIST** message. If you do not assign an image list to the ComboBoxEx control, the control displays item text only.

ComboBoxEx Control Styles

ComboBoxEx controls only support the following ComboBox styles:

- CBS_SIMPLE
- CBS_DROPDOWN
- CBS_DROPDOWNLIST
- WS_CHILD

There are also several extended styles that are used only by ComboBoxEx.

Note The CBS_SIMPLE style may not work properly in some cases.

Because the ComboBoxEx control performs owner draw tasks for you based on an assigned image list, the CBS_OWNERDRAWFIXED style is implied; you need not use it when creating the control. Because image lists are used to provide item graphics, the CBS_OWNERDRAWVARIABLE style cannot be used. The ComboBoxEx control also supports extended styles that provide additional features.

ComboBoxEx Control Items

ComboBoxEx controls maintain item information using a **COMBOBOXEXITEM** structure. This structure includes members for item indexes, image indexes (normal, selection state, and overlay), indentation values, text strings, and item-specific values.

The ComboBoxEx control provides easy access to and manipulation of items through messaging. To add or delete an item, send the **CBEM_INSERTITEM** or **CBEM_DELETEITEM** message. You can modify items currently in the control using the **CBEM_SETITEM** message.

Callback Items

ComboBoxEx controls support callback item attributes. You can specify an item as a callback item when you add it to the control using **CBEM_INSERTITEM**. When you assign values to an item's **COMBOBOXEXITEM** structure, you must specify the appropriate callback flag values. The following are **COMBOBOXEXITEM** structure members and their corresponding callback flag values:

Member	Callback value
pszText	LPSTR_TEXTCALLBACK
iImage	I_IMAGECALLBACK
iSelectedImage	I_IMAGECALLBACK
iOverlay	I_IMAGECALLBACK
iIndent	I_INDENTCALLBACK

The control will request information about callback items by sending **CBEN_GETDISPINFO** notification messages. This notification is sent in the form of a **WM_NOTIFY** message. When your application processes this message, it must provide the requested information for the control. If you set the **mask** member of the accompanying **COMBOBOXEXITEM** structure to CBEIF_DI_SETITEM, the control will store the item data and will not request it again.

ComboBoxEx Control Image Lists

If you want a ComboBoxEx control to display icons with items, you must provide an image list. ComboBoxEx controls support up to three images for an item—one for its selected state, one for its nonselected state, and one for an overlay image. Assign an

existing image list to a ComboBoxEx control using the **CBEM_SETIMAGELIST** message.

The **COMBOBOXEXITEM** structure contains members that represent the image indexes for each image list (selected, unselected, and overlay). For each item, set these members to display the desired images. It is not necessary to specify image indexes for each type of image. You can mix and match image types as you like, but always set the **mask** member of the **COMBOBOXEXITEM** structure to indicate which members are being used. The control ignores members that have not been flagged as valid.

Note If you use the CBS_SIMPLE style, icons are not displayed.

About ComboBoxEx Control Notification Messages

A ComboBoxEx control sends notification messages to report changes within itself or to request callback item information. The parent of the control receives all **WM_COMMAND** messages from the combo box contained within the ComboBoxEx control. The ComboBoxEx control sends its own notifications using **WM_NOTIFY** messages. As a result, the control's owner must be prepared to process both forms of notification messages.

Following are the ComboBoxEx-specific notification messages:

Notification	Description
CBEN_BEGINEDIT	Signals that the user has activated the drop-down list or clicked in the control's edit box.
CBEN_DELETEITEM	Reports that an item was deleted.
CBEN_ENDEDIT	Signals that the user has selected an item from the drop-down list or has concluded an edit operation within the edit box.
CBEN_GETDISPINFO	Requests information about an item's attributes.
CBEN_INSERTITEM	Signals that an item was inserted in the control.

ComboBoxEx Control Message Forwarding

The following are the standard combo-box messages that a ComboBoxEx control forwards to its child combo box. Some of these messages may be processed by the ComboBoxEx control either before or after the message has been forwarded:

- CB_DELETESTRING
- CB_FINDSTRINGEXACT
- CB_GETCOUNT
- CB_GETCURSEL
- CB_GETDROPPEDCONTROLRECT
- CB_GETDROPPEDSTATE

- CB_GETEXTENDEDUI
- CB_GETITEMDATA
- CB_GETITEMHEIGHT
- CB_GETLBTEXT
- CB_GETLBTEXTLEN
- CB_LIMITTEXT
- CB_RESETCONTENT
- CB_SETCURSEL
- CB_SETDROPPEDWIDTH
- CB_SETEXTENDEDUI
- CB_SETITEMDATA
- CB_SETITEMHEIGHT
- CB_SHOWDROPDOWN

Following are the windows messages that a ComboBoxEx control forwards to its parent window:

- **WM_COMMAND** (This includes all of the CBN_ notifications.)
- **WM_NOTIFY**

Using ComboBoxEx Controls

Creating a ComboBoxEx Control

To create a ComboBoxEx control, call the **CreateWindowEx** function, using WC_COMBOBOXEX as the window class. You first must register the window class by calling the **InitCommonControlsEx** function, while specifying the ICC_USEREX_CLASSES bit in the accompanying **INITCOMMONCONTROLSEX** structure.

The following application-defined function creates a ComboBoxEx control with the CBS_DROPDOWN style in the main window:

```
// CreateComboEx - Registers the ComboBoxEx window class
// and creates a ComboBoxEx control in the client area of
// the main window.
//
// g_hwndMain - A handle to the main window.
// g_hinst    - A handle to the program instance.

HWND WINAPI CreateComboEx(void)
{
    HWND hwnd;
```

```
    INITCOMMONCONTROLSEX icex;

    icex.dwSize = sizeof(INITCOMMONCONTROLSEX);
    icex.dwICC = ICC_USEREX_CLASSES;

    InitCommonControlsEx(&icex);

    hwnd = CreateWindowEx(0, WC_COMBOBOXEX, NULL,
                          WS_BORDER | WS_VISIBLE |
                          WS_CHILD | CBS_DROPDOWN,
                          // No size yet-resize
                          // after setting image list.
                          0,      // Vertical position of Combobox
                          0,      // Horizontal position of Combobox
                          0,      // Sets the width of Combobox
                          100,    // Sets the height of Combobox
                          g_hwndMain,
                          NULL,
                          g_hinst,
                          NULL);

    return (hwnd);
}
```

Preparing ComboBoxEx Items and Images

To add an item to a ComboBoxEx control, first define a **COMBOBOXEXITEM** structure. Set the **mask** member of the structure to indicate which members you want the control to use. Set the specified members of the structure to the values you want, and then send the **CBEM_INSERTITEM** message to add the item to the control.

The following application-defined function adds 15 items to an existing ComboBoxEx control. Note that the **mask** member of the **COMBOBOXEXITEM** structure includes flag values that tell the control to display images for each item:

```
#define MAX_ITEMS 15

// AddItems - Uses the CBEM_INSERTITEM message to add items
//to an existing ComboBoxEx control.

BOOL WINAPI AddItems(HWND hwndCB)
{
    //  Declare and init locals.
    COMBOBOXEXITEM cbei;
    int iCnt;

    typedef struct {
```

(continued)

(continued)

```
    int iImage;
    int iSelectedImage;
    int iIndent;
    LPTSTR pszText;
} ITEMINFO, *PITEMINFO;

ITEMINFO IInf[] = {
    { 0, 3,  0, "first"},
    { 1, 4,  1, "second"},
    { 2, 5,  2, "third"},
    { 0, 3,  0, "fourth"},
    { 1, 4,  1, "fifth"},
    { 2, 5,  2, "sixth"},
    { 0, 3,  0, "seventh"},
    { 1, 4,  1, "eighth"},
    { 2, 5,  2, "ninth"},
    { 0, 3,  0, "tenth"},
    { 1, 4,  1, "eleventh"},
    { 2, 5,  2, "twelfth"},
    { 0, 3,  0, "thirteenth"},
    { 1, 4,  1, "fourteenth"},
    { 2, 5,  2, "fifteenth"}
};

// Set the mask common to all items.
cbei.mask = CBEIF_TEXT | CBEIF_INDENT |
            CBEIF_IMAGE| CBEIF_SELECTEDIMAGE;

for(iCnt=0;iCnt<MAX_ITEMS;iCnt++){
    // Initialize the COMBOBOXEXITEM struct.
    cbei.iItem          = iCnt;
    cbei.pszText        = IInf[iCnt].pszText;
    cbei.cchTextMax     = sizeof(IInf[iCnt].pszText);
    cbei.iImage         = IInf[iCnt].iImage;
    cbei.iSelectedImage = IInf[iCnt].iSelectedImage;
    cbei.iIndent        = IInf[iCnt].iIndent;

    // Tell the ComboBoxEx to add the item. Return
    // FALSE if this fails.
    if(SendMessage(hwndCB,CBEM_INSERTITEM,0,(LPARAM)&cbei) == -1)
        return FALSE;
}

// Assign the existing image list to the ComboBoxEx
```

```
    // control and return TRUE.
    SendMessage(hwndCB,CBEM_SETIMAGELIST,0,(LPARAM)g_himl);

    // Set size of control to make sure it's displayed
    // correctly now that the image list is set.
    SetWindowPos(hwndCB,NULL,20,20,250,120,SWP_NOACTIVATE);

    return TRUE;
}
```

Supporting Callback Items

If your application is going to use callback items in a ComboBoxEx control, it must be prepared to handle the **CBEN_GETDISPINFO** notification message. A ComboBoxEx control sends this notification whenever it needs the owner to provide specific item information. For more information about callback items, see *Callback Items*.

The following application-defined function processes **CBEN_GETDISPINFO** by providing attributes for a given item. Note that it sets the **mask** member of the incoming **COMBOBOXEXITEM** structure to CBEIF_DI_SETITEM. Setting **mask** to this value makes the control retain the item information, so it will not need to request the information again:

```
// DoItemCallback - Processes CBEN_GETDISPINFO by
// providing item attributes for a given callback item.

void WINAPI DoItemCallback(PNMCOMBOBOXEX pNMCBex)
{
    DWORD dwMask = pNMCBex->ceItem.mask;

    if(dwMask & CBEIF_TEXT)
        ;// Provide item text.

    if(dwMask & CBEIF_IMAGE)
        ;// Provide an item image index.

    /*
     * Provide other callback information as desired.
     */

    // Make the ComboBoxEx control hold onto the item
    // information.
    pNMCBex->ceItem.mask = CBEIF_DI_SETITEM;
}
```

Processing ComboBoxEx Notifications

A ComboBoxEx control notifies its parent window of events by sending **WM_NOTIFY** messages. Because the control uses a child combo box, it forwards all **WM_COMMAND** notification messages it receives to the parent window to be processed. Therefore, your application must be prepared to process **WM_NOTIFY** messages from the ComboBoxEx and **WM_COMMAND** messages forwarded from the ComboBoxEx's child combo box control.

The example in this section handles the **WM_NOTIFY** and **WM_COMMAND** messages from a ComboBoxEx control by calling a corresponding application-defined function to process them.

```
LRESULT CALLBACK WndProc (HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
   switch(msg){
      case WM_COMMAND:
         // If this is an "old style" combo box
         // notification, handle it.
         if((HWND)lParam == g_hwndCB)
            DoOldNotify(hwnd, wParam);
         break;

      case WM_NOTIFY:
         return (DoCBEXNotify(hwnd, lParam));

      case WM_PAINT:
         hdc = BeginPaint(hwnd, &ps);
         EndPaint(hwnd, &ps);
         break;

      case WM_DESTROY:
         PostQuitMessage(0);
         break;

      default:
         return DefWindowProc(hwnd, msg, wParam, lParam);
         break;
   }
   return FALSE;
}
```

ComboBoxEx Control Extended Styles

ComboBoxEx controls support most standard combo-box control styles. Additionally, ComboBoxEx controls support the following extended styles, which you can set and

retrieve by using **CBEM_SETEXTENDEDSTYLE** and **CBEM_GETEXTENDEDSTYLE** messages:

CBES_EX_CASESENSITIVE
String searches in the list will be case-sensitive. This includes searches as a result of text being typed in the edit box and the CB_FINDSTRINGEXACT message.

CBES_EX_NOEDITIMAGE
The edit box and the drop-down list will not display item images.

CBES_EX_NOEDITIMAGEINDENT
The edit box and the drop-down list will not display item images.

CBES_EX_NOSIZELIMIT
Allows the ComboBoxEx control to be sized vertically smaller than its contained combo-box control. If the ComboBoxEx is sized smaller than the combo box, the combo box will be clipped.

CBES_EX_PATHWORDBREAKPROC
Windows NT only. The edit box will use the slash (/), backslash (\), and period (.) characters as word delimiters. This makes keyboard shortcuts for word-by-word cursor movement (CTRL+ARROW) effective in path names and URLs.

Note If you try to set an extended style for a ComboBoxEx control created with the CBS_SIMPLE style, it might not repaint properly. The CBS_SIMPLE style also does not work properly with the CBES_EX_PATHWORDBREAKPROC extended style.

ComboBoxEx Control Reference

ComboBoxEx Control Messages

CBEM_DELETEITEM

Removes an item from a ComboBoxEx control.

```
CBEM_DELETEITEM
    wParam = (WPARAM)(int) iIndex;
    lParam = 0;
```

Parameters

iIndex
Zero-based index of the item to be removed.

Return Values

Returns an INT value that represents the number of items remaining in the control. If *iIndex* is invalid, the message returns CB_ERR.

Remarks

This message maps to the combo-box control message **CB_DELETESTRING**.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

CBEM_GETCOMBOCONTROL

Retrieves the handle to the child combo-box control.

```
CBEM_GETCOMBOCONTROL
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the handle to the combo-box control within the ComboBoxEx control.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

CBEM_GETEDITCONTROL

Retrieves the handle to the edit control portion of a ComboBoxEx control. A ComboBoxEx control uses an edit box when it is set to the CBS_DROPDOWN style.

```
CBEM_GETEDITCONTROL
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the handle to the edit control within the ComboBoxEx control if it uses the CBS_DROPDOWN style. Otherwise, the message returns NULL.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

CBEM_GETEXTENDEDSTYLE

Retrieves the extended styles that are in use for a ComboBoxEx control.

```
CBEM_GETEXTENDEDSTYLE
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a DWORD value that contains the ComboBoxEx control extended styles in use for the control.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

CBEM_GETIMAGELIST

Retrieves the handle to an image list assigned to a ComboBoxEx control.

```
CBEM_GETIMAGELIST
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the handle to the image list assigned to the control if successful, or NULL otherwise.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

CBEM_GETITEM

Retrieves item information for a given ComboBoxEx item.

```
CBEM_GETITEM
    wParam = 0;
    lParam = (LPARAM)(PCOMBOBOXEXITEM) pCBItem;
```

Parameters

pCBItem
Address of a **COMBOBOXEXITEM** structure that will receive the item information.

Return Values

Returns nonzero if successful, or zero otherwise.

Remarks

When the message is sent, the **iItem** and **mask** members of the structure must be set to indicate the index of the target item and the type of information to be retrieved. Other members are set as needed. For example, to get text, you must set the CBEIF_TEXT flag in **mask**, and assign a value to **cchTextMax**. Setting the **iItem** member to –1 will retrieve the item displayed in the edit control.

If the CBEIF_TEXT flag is set in the **mask** member of the **COMBOBOXEXITEM** structure, the control may change the **pszText** member of the structure to point to the new text instead of filling the buffer with the requested text. Applications should not assume that the text always will be placed in the requested buffer.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

CBEM_GETUNICODEFORMAT

Retrieves the UNICODE character format flag for the control.

```
CBEM_GETUNICODEFORMAT
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the UNICODE format flag for the control. If this value is nonzero, the control is using UNICODE characters. If this value is zero, the control is using ANSI characters.

Remarks

See the remarks for **CCM_GETUNICODEFORMAT** for a discussion of this message.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

See Also

CBEM_SETUNICODEFORMAT

CBEM_HASEDITCHANGED

Determines whether or not the user has changed the text of a ComboBoxEx edit control.

```
CBEM_HASEDITCHANGED
    wParam = 0;
    lParam = 0;
```

Return Values

Returns TRUE if the text in the control's edit box has been changed, or FALSE otherwise.

Remarks

A ComboBoxEx control uses an edit box control when it is set to the CBS_DROPDOWN style. You can get the edit box control's window handle by sending a **CBEM_GETEDITCONTROL** message.

When the user begins editing, you will receive a **CBEN_BEGINEDIT** notification. When editing is complete, or the focus changes, you will receive a **CBEN_ENDEDIT** notification. The CBEM_HASEDITCHANGED message is only useful for determining whether or not the text has been changed if it is sent before the CBEN_ENDEDIT notification. If is sent afterwards, it will return FALSE. For example, suppose the user starts to edit the text in the edit box but changes focus, generating a CBEN_ENDEDIT notification. If you then send a CBEM_HASEDITCHANGED message, it will return FALSE, even though the text has been changed.

The CBS_SIMPLE style does not work correctly with CBEM_HASEDITCHANGED.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

CBEM_INSERTITEM

Inserts a new item in a ComboBoxEx control.

```
CBEM_INSERTITEM
    wParam = 0;
    lParam = (LPARAM)(const COMBOBOXEXITEM FAR *) lpcCBItem;
```

Parameters

lpcCBItem

Address of a **COMBOBOXEXITEM** structure that contains information about the item to be inserted. When the message is sent, the **iItem** member must be set to indicate the zero-based index at which to insert the item. To insert an item at the end of the list, set the **iItem** member to –1.

Return Values

Returns the index at which the new item was inserted if successful, or –1 otherwise.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

CBEM_SETEXTENDEDSTYLE

Sets extended styles within a ComboBoxEx control.

```
CBEM_SETEXTENDEDSTYLE
    wParam = (WPARAM)(DWORD) dwExMask;
    lParam = (LPARAM)(DWORD) dwExStyle;
```

Parameters

dwExMask
A DWORD value that indicates which styles in *dwExStyle* are to be affected. Only the extended styles in *dwExMask* will be changed. If this parameter is zero, then all of the styles in *dwExStyle* will be affected.

dwExStyle
A DWORD value that contains the ComboBoxEx control extended styles to set for the control.

Return Values

Returns a DWORD value that contains the extended styles previously used for the control.

Remarks

dwExMask allows you to modify one or more extended styles without having to retrieve the existing styles first. For example, if you pass CBES_EX_NOEDITIMAGE for *dwExMask* and 0 for *dwExStyle*, the CBES_EX_NOEDITIMAGE style will be cleared, but all other styles will remain the same.

If you try to set an extended style for a ComboBoxEx control created with the CBS_SIMPLE style, it might not repaint properly.

Version 4.00 and later of Comctl32.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 commctrl.h.

CBEM_SETIMAGELIST

Sets an image list for a ComboBoxEx control.

```
CBEM_SETIMAGELIST
    wParam = 0;
    lParam = (LPARAM)(HIMAGELIST) himl;
```

Parameters

himl

Handle to the image list to be set for the control.

Return Values

Returns the handle to the image list previously associated with the control, or returns NULL if no image list was previously set.

Remarks

The height of images within your image list might change the size requirements of the ComboBoxEx control. It is recommended that you resize the control after sending this message to ensure that it is displayed properly.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

CBEM_SETITEM

Sets the attributes for an item in a ComboBoxEx control.

```
CBEM_SETITEM
    wParam = 0;
    lParam = (LPARAM) (const COMBOBOXEXITEM FAR *) lpcCBItem;
```

Parameters

lpcCBItem

Address of a **COMBOBOXEXITEM** structure that contains the item information to be set. When the message is sent, the **mask** member of the structure must be set to indicate which attributes are valid and the **iItem** member must specify the zero-based index of the item to be modified. Setting the **iItem** member to –1 will modify the item displayed in the edit control.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

CBEM_SETUNICODEFORMAT

Sets the UNICODE character format flag for the control. This message allows you to change the character set used by the control at run time, instead of having to re-create the control.

```
CBEM_SETUNICODEFORMAT
    wParam = (WPARAM)(BOOL)fUnicode;
    lParam = 0;
```

Parameters

fUnicode
Determines the character set that is used by the control. If this value is nonzero, the control will use UNICODE characters. If this value is zero, the control will use ANSI characters.

Return Values

Returns the previous UNICODE format flag for the control.

Remarks

See the remarks for **CCM_SETUNICODEFORMAT** for a discussion of this message.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

See Also

CBEM_GETUNICODEFORMAT

ComboBoxEx Control Notification Messages

CBEN_BEGINEDIT

Sent when the user activates the drop-down list or clicks in the control's edit box. This notification is sent in the form of a **WM_NOTIFY** message.

```
CBEN_BEGINEDIT
    lpnmhdr = (LPNMHDR) lParam;
```

Parameters

lpnmhdr
Address of an **NMHDR** structure that contains information about the notification.

Return Values

The application processing this notification must return zero.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

CBEN_DELETEITEM

Sent when an item has been deleted. This notification is sent in the form of a **WM_NOTIFY** message.

```
CBEN_DELETEITEM
    pCBEx= (PNMCOMBOBOXEX) lParam;
```

Parameters

pCBEx
Address of an **NMCOMBOBOXEX** structure that contains information about the notification and the deleted item.

Return Values

The application processing this notification must return zero.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

CBEN_DRAGBEGIN

Sent when the user begins dragging the image of the item displayed in the edit portion of the control. This notification is sent in the form of a **WM_NOTIFY** message.

```
CBEN_DRAGBEGIN
    lpnmdb = (LPNMCBEDRAGBEGIN) lParam;
```

Parameters

lpnmdb
Address of an **NMCBEDRAGBEGIN** structure that contains information about the notification.

Return Values

The return value is ignored.

Remarks

If the receiving application implements drag-and-drop functionality from the control, the application will begin the drag-and-drop operation in response to this notification.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

CBEN_ENDEDIT

Sent when the user has concluded an operation within the edit box or has selected an item from the control's drop-down list. This notification is sent in the form of a **WM_NOTIFY** message.

```
CBEN_ENDEDIT
    pnmEditInfo = (PNMCBEENDEDIT) lParam;
```

Parameters

pnmEditInfo

Address of an **NMCBEENDEDIT** structure that contains information about how the user concluded the edit operation.

Return Values

To accept the notification and allow the control to display the selected item, return FALSE. To abort the edit selection, return TRUE.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

CBEN_GETDISPINFO

Sent to retrieve display information about a callback item. This notification is sent in the form of a **WM_NOTIFY** message.

```
CBEN_GETDISPINFO
    pDispInfo = (PNMCOMBOBOXEX) lParam;
```

Parameters

pDispInfo

Address of an **NMCOMBOBOXEX** structure that contains information about the notification.

Return Values

The application processing this notification must return zero.

Remarks

The **NMCOMBOBOXEX** structure contains a **COMBOBOXEXITEM** structure. The **mask** member specifies the information being requested by the control.

Fill the appropriate members of the structure to return the requested information to the control. If your message handler sets the **mask** member of the **COMBOBOXEXITEM** structure to CBEIF_DI_SETITEM, the header control stores the information and will not request it again.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

CBEN_INSERTITEM

Sent when a new item has been inserted in the control. This notification is sent in the form of a **WM_NOTIFY** message.

```
CBEN_INSERTITEM
    pNMInfo = (PNMCOMBOBOXEX) lParam;
```

Parameters

pNMInfo
Address of an **NMCOMBOBOXEX** structure containing information about the notification and the item that was inserted.

Return Values

The application processing this notification must return zero.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

NM_SETCURSOR (ComboBoxEx)

Notifies a ComboBoxEx control's parent window that the control is setting the cursor in response to a WM_SETCURSOR message. This notification is sent in the form of a **WM_NOTIFY** message.

```
NM_SETCURSOR
    lpnmm = (LPNMMOUSE) lParam;
```

Parameters

lpnmm

Address of an **NMMOUSE** structure that contains additional information about this notification message.

Return Values

Return nonzero to allow the control to set the cursor, or zero to prevent the control from setting the cursor.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

ComboBoxEx Control Structures

COMBOBOXEXITEM

Contains information about an item in a ComboBoxEx control.

```
typedef struct {
    UINT    mask;
    int     iItem;
    LPTSTR  pszText;
    int     cchTextMax;
    int     iImage;
    int     iSelectedImage;
    int     iOverlay;
    int     iIndent;
    LPARAM  lParam;
} COMBOBOXEXITEM, *PCOMBOBOXEXITEM;
```

Members

mask

Set of bit flags that specify attributes of this structure or of an operation that is using this structure. The flags specify members that are valid or must be filled in. This member can be a combination of the following values:

Flag	Description
CBEIF_DI_SETITEM	Set this flag when processing **CBEN_GETDISPINFO** and the ComboBoxEx control will retain the supplied information and not request it again.
CBEIF_IMAGE	The iImage member is valid or must be filled in.
CBEIF_INDENT	The iIndent member is valid or must be filled in.
CBEIF_LPARAM	The lParam member is valid or must be filled in.
CBEIF_OVERLAY	The iOverlay member is valid or must be filled in.
CBEIF_SELECTEDIMAGE	The iSelectedImage member is valid or must be filled in.
CBEIF_TEXT	The pszText member is valid or must be filled in.

iItem
Zero-based index of the item.

pszText
Address of a character buffer that contains or receives the item's text. If text information is being retrieved, this member must be set to the address of a character buffer that will receive the text. The size of this buffer must also be indicated in **cchTextMax**. If this member is set to LPSTR_TEXTCALLBACK, the control will request the information by using the **CBEN_GETDISPINFO** notification messages.

cchTextMax
Length of **pszText**, in characters. If text information is being set, this member is ignored.

iImage
Zero-based index of an image within the image list. The specified image will be displayed for the item when it is not selected. If this member is set to I_IMAGECALLBACK, the control will request the information by using **CBEN_GETDISPINFO** notification messages.

iSelectedImage
Zero-based index of an image within the image list. The specified image will be displayed for the item when it is selected. If this member is set to I_IMAGECALLBACK, the control will request the information by using **CBEN_GETDISPINFO** notification messages.

iOverlay
One-based index of an overlay image within the image list. If this member is set to I_IMAGECALLBACK, the control will request the information by using **CBEN_GETDISPINFO** notification messages.

iIndent
Number of indent spaces to display for the item. Each indentation equals 10 pixels. If this member is set to I_INDENTCALLBACK, the control will request the information by using **CBEN_GETDISPINFO** notification messages.

lParam
32-bit value specific to the item.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

NMCBEENDEDIT

Contains information about the conclusion of an edit operation within a ComboBoxEx control. This structure is used with the **CBEN_ENDEDIT** notification message.

```
typedef struct {
    NMHDR  hdr;
    BOOL   fChanged;
    int    iNewSelection;
    TCHAR  szText[CBEMAXSTRLEN]; // CBEMAXSTRLEN is 260
    int    iWhy;
} NMCBEENDEDIT, *PNMCBEENDEDIT;
```

Members

hdr
NMHDR structure that contains information about the notification message.

fChanged
Value indicating whether the contents of the control's edit box have changed. This value is nonzero if the contents have been modified, or zero otherwise.

iNewSelection
Zero-based index of the item that will be selected after completing the edit operation. This value can be CB_ERR if no item will be selected.

szText
Zero-terminated string that contains the text from within the control's edit box.

iWhy
Value that specifies the action that generated the **CBEN_ENDEDIT** notification message. This value can be one of the following:

CBENF_DROPDOWN	The user activated the drop-down list.
CBENF_ESCAPE	The user pressed ESC.
CBENF_KILLFOCUS	The edit box lost the keyboard focus.
CBENF_RETURN	The user completed the edit operation by pressing ENTER.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

NMCBEDRAGBEGIN

Contains information used with the **CBEN_DRAGBEGIN** notification message.

```
typedef struct {
    NMHDR hdr;
    int iItemid;
    TCHAR szText[CBEMAXSTRLEN];
}NMCBEDRAGBEGIN, *LPNMCBEDRAGBEGIN;
```

Members

hdr
NMHDR structure that contains information about the notification message.

iItemid
Zero-based index of the item being dragged. This value always will be –1, indicating that the item being dragged is the item displayed in the edit portion of the control.

szText
Character buffer that contains the text of the item being dragged.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

NMCOMBOBOXEX

Holds information specific to ComboBoxEx items for use with notification messages.

```
typedef struct {
    NMHDR hdr;
    COMBOBOXEXITEM ceItem;
} NMCOMBOBOXEX, *PNMCOMBOBOXEX;
```

Members

hdr

NMHDR structure that contains information about the notification message.

celtem

COMBOBOXEXITEM structure that holds item information specific to the current notification. Upon receiving a notification message, the **COMBOBOXEXITEM** structure holds information required for the owner to respond. The members of this structure are often used as fields for the owner to return values in response to the notification.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

CHAPTER 11

Creating Wizards

Wizards are a type of property sheet that provide a simple and powerful way to guide users through complex procedures.

For a general discussion of property sheets, see *Property Sheets*. For links to the Property Sheet API elements, see *Property Sheet Reference*.

Introduction

Wizards are one of the keys to simplifying the user experience. They allow you to take a complex operation, such as configuration of an application, and break it into a series of simple steps. At each point in the process, you can provide an explanation of what is needed, and display controls that allow the user to make selections and enter text.

In terms of implementation, a wizard is actually a type of property sheet. A property sheet is essentially a container for a collection of *pages*, where each page is a separate dialog box. Much of what you need to do to implement a wizard actually involves familiar dialog box programming techniques.

A standard property sheet displays the pages as if they were stacked one on top of the other. Each page has a tab at the top, and users select a page by clicking its tab. They then interact with the page as they would with a regular dialog box. Figure 11-1 shows the Microsoft Windows 2000 Date/Time properties sheet in the Control Panel.

Wizards present pages one at a time. Instead of tabs, there are Next and Back buttons located at the bottom of the wizard. Users click these buttons to navigate forward or backward through a sequence of pages. The order in which pages are displayed is controlled by the application and can be modified based on user input. Figure 11-2 shows the welcome page of the Windows 2000 Hardware Wizard in the Control Panel.

This chapter outlines the basic process of creating a Wizard97 style wizard. The next section is a general discussion of wizard implementation. The final section of the chapter is a detailed walkthrough of Wiz97, a simple wizard application.

Note The discussion in most of this document assumes that you are implementing a Wizard97-style wizard for a system with version 5.80 or later of the Common Controls. You can use this style on Windows 2000 or later systems, or any Windows system with Internet Explorer 5 or later. However, you must use a **#define** to set the value of _WIN32_IE to 0x0500 or higher before including the common controls header file (commctrl.h). If you attempt to use the Wizard97 style with earlier versions of the common controls, your application may compile but it will not display properly. For a discussion about how to create a Wizard97-compatible wizard on earlier systems, see the *Backward Compatible Wizards* section.

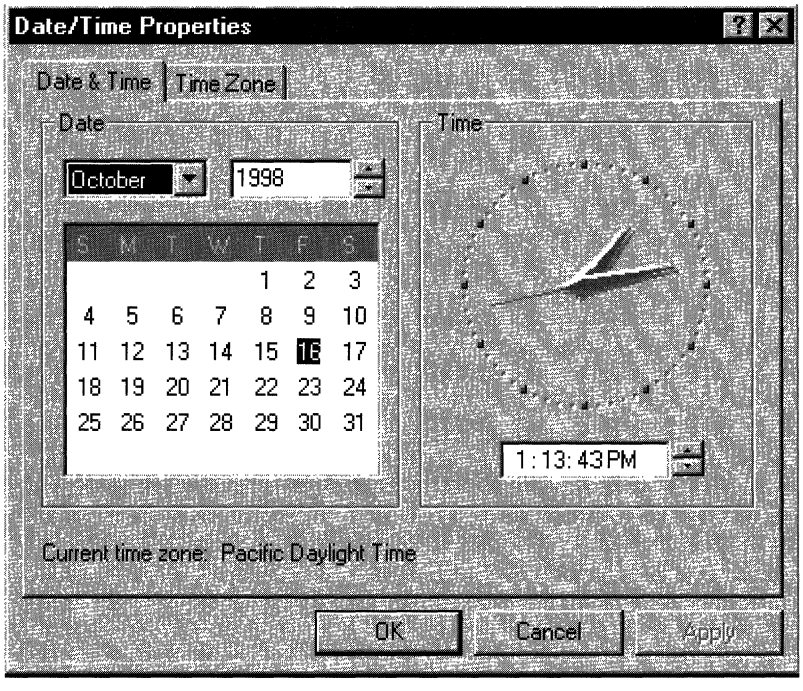


Figure 11-1: Date/Time properties sheet.

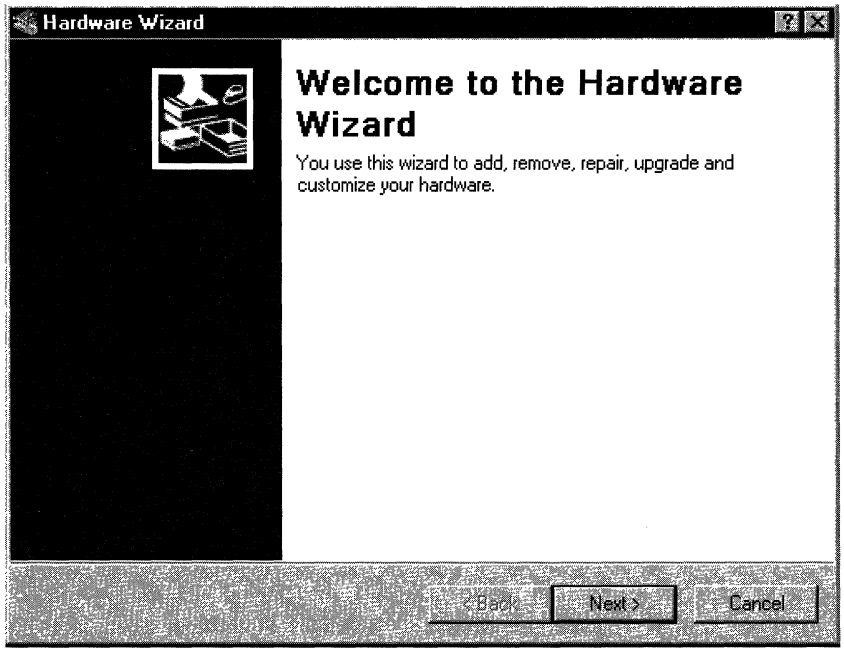


Figure 11-2: Windows 2000 Hardware Wizard welcome page.

How to Implement a Wizard

Implementing a wizard is similar to implementing a regular property sheet. At the most basic level, it is simply a matter of setting the appropriate style flag. To implement the individual pages, use the same dialog box programming techniques that you use for property sheets. However, there are a number of wizard-specific property sheet features that are used by nearly all wizards.

The primary focus of this chapter is on those aspects of property sheets that are specific to wizards. It provides a detailed discussion of how to create a wizard's property sheet and pages, and how to handle navigation from one page to the next. It also goes into a few of the related design issues. For a complete discussion of design issues, you should see the *Wizard97 Specification*, elsewhere in the Platform SDK. Familiarity with standard dialog box programming techniques is assumed. For detailed information, see the discussion of dialog boxes, elsewhere in the Platform SDK. For a more general discussion of property sheets, see the *Property Sheets* chapter.

Once you have defined your task and designed a user interface for each page, the basic procedure for implementing a wizard is relatively straightforward:

1. Create a dialog box template for each page.
2. Define the pages by creating a **PROPSHEETPAGE** structure for each page. This structure defines the page, and contains pointers to the dialog box template and any bitmaps or other resources.
3. Pass the **PROPSHEETPAGE** structure created in the previous step to **CreatePropertySheetPage** to create the page's HPROPSHEETPAGE handle.
4. Define the wizard by creating a **PROPSHEETHEADER** structure for it.
5. Pass the **PROPSHEETHEADER** structure to **PropertySheet** to display the wizard.
6. Implement dialog box procedures for each page to handle notification messages from the page's controls and the wizard's buttons and to process other Windows messaging.

Creating the Dialog Box Templates

This chapter discusses how to create a Wizard97-style wizard. This wizard style is used by a wide variety of Microsoft applications. To be compatible with the Wizard97 style, your templates need to adhere to the *Wizard97 Specification*. This document has guidelines for such things as the dimensions for the dialog boxes, bitmap dimensions and colors, and the placement of controls.

There are two basic types of wizard page: *exterior* and *interior*. The exterior pages are the *welcome* and *completion* pages. The other pages in the wizard are interior.

Exterior Page Dialog Box Templates

All wizards have two exterior pages: welcome and completion. They come at the beginning and end of the sequence, respectively. Their basic layout is identical. For example, Figure 11-3 shows a sample Wizard97 welcome page.

Figure 11-3: Sample welcome page.

For exterior pages, the dialog box is 317x193 dialog units (dlus). It fills all of the wizard, except for the caption and the band at the bottom that contains the Back, Next, and Cancel buttons. The left side of the template, which is reserved for a *watermark* bitmap, should not contain any controls. The watermark is specified in the wizard's **PROPSHEETHEADER** structure and is added to the page automatically.

When you create the watermark bitmap, keep in mind that the dialog box may increase in size if, for example, the user chooses a large system font. When the page grows, the area reserved for the watermark gets bigger proportionately. The watermark is not stretched to fill the larger area. It is left in its original size in the upper-left portion of the reserved area. The part of the larger reserved area that is not covered by the watermark is automatically filled with the color of the bitmap's upper-left pixel.

You can place controls in the area to the right of the watermark as you would for a regular dialog box. The background color of this area is determined by the system, and requires no action on your part. You typically put two static controls in this area. The

upper one holds the title and uses a large bold font (12 point Verdana Bold). The other one, which is for explanatory text, uses the standard dialog box font.

The main difference between the two types of exterior page is the wizard buttons and the text in the static controls. Welcome pages normally have a Next and a Back button, with only the Next button enabled. Completion pages have the Back button enabled, and the Next button is replaced by a Finish button. The wizard buttons are handled in the dialog procedure, and don't affect the template. As far as the template is concerned, all you need to do is provide appropriate text in the static controls for each exterior page.

Interior Page Dialog Box Templates

Interior pages have a somewhat different appearance than exterior pages. Figure 11-4 shows what a typical Wizard97 interior page looks like.

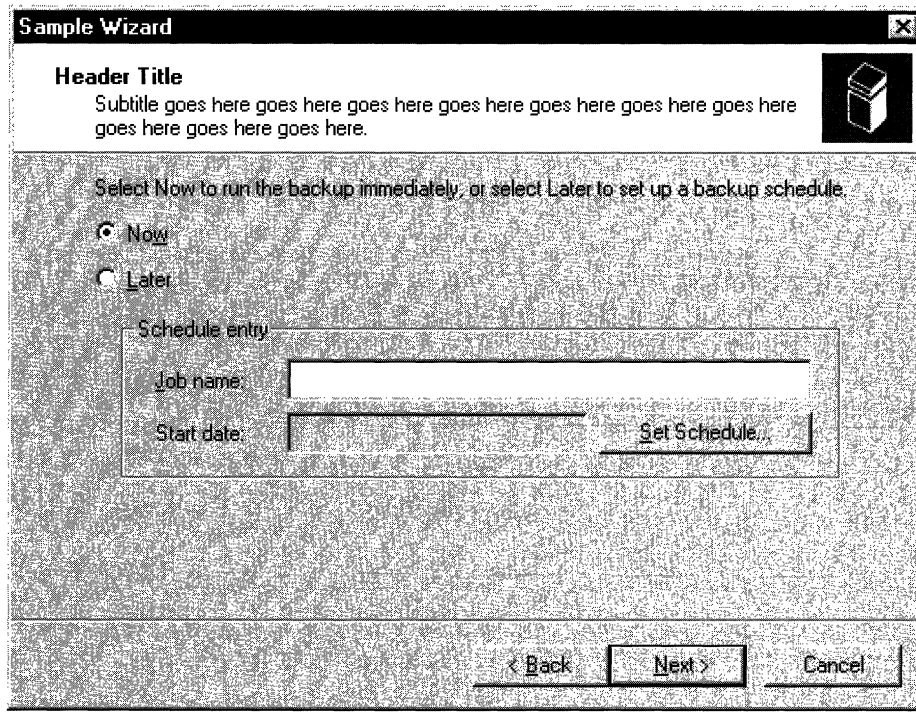


Figure 11-4: Sample Wizard97 interior page.

The header area at the top of the page is handled by the property sheet, so it is not included in the template. The contents of the header are specified in the page's **PROPSHEETPAGE** structure and the wizard's **PROPSHEETHEADER** structure. Because the interior pages' template needs to fit between the header and the buttons, it is 317x143 dlus, somewhat smaller than the template for exterior pages. None of the template area is reserved, but you should consult the *Wizard97 Specification* for guidelines on control placement.

Defining the Wizard Pages

Once you have created the dialog box templates and related resources such as bitmaps and string tables, you can create the property sheet pages. The procedure is similar to that for standard property sheets. First, fill in the appropriate members of a **PROPSHEETPAGE** structure and then call the **CreatePropertySheetPage** function to create the page's HPROPSHEETPAGE handle. However, there are a few features of this structure that are specific to wizards in general, and the Wizard97 style in particular.

There are a number of wizard-related flags that can be set in the **dwFlags** member of the **PROPSHEETPAGE** structure:

Flag	Description
PSP_HIDEHEADER	Set this flag for exterior pages. Do not set it for interior pages
PSP_USEHEADERTITLE	Set this flag for interior pages to put a title in the header area.
PSP_USEHEADERSUBTITLE	Set this flag for interior pages to put a subtitle in the header area.

Defining the Wizard Property Sheet

As with ordinary property sheets, you define the wizard's property sheet by filling in members of a **PROPSHEETHEADER** structure. This structure allows you to specify the pages that will make up the wizard and the default order in which they will be displayed, along with several related parameters. You then launch the wizard by calling the **PropertySheet** function.

There are a number of wizard-related flags that can be set in the structure's **dwFlags**.

Flag	Description
PSH_USEHBMHEADER	Set this flag instead of PSH_USEHEADER to use an HBITMAP handle to identify the bitmap that will be placed at the right side of an interior page's header area. Assign the handle to the **hbmHeader** member.
PSH_USEHBMWATERMARK	Set this flag instead of PSH_WATERMARK to use an HBITMAP handle to identify the watermark bitmap. Assign the handle to the **hbmWatermark** member.
PSH_USEHEADER	Set this flag to use a resource ID to identify the header bitmap. Use MAKEINTRESOURCE to convert the resource ID into a string, and assign the string to the **pszbmHeader** member.

Flag	Description
PSH_USEPLWATERMARK	Set this flag to define your own palette for the watermark bitmap. Assign the palette's HPALETTE handle to the **hplWatermark** member.
PSH_WATERMARK	Set this flag to use a resource ID to identify the watermark bitmap. Use MAKEINTRESOURCE to convert the resource ID into a string, and assign the string to the **pszbmWatermark** member.
PSH_WIZARD97	Set this flag to specify a Wizard97 style wizard.
PSH_WIZARDCONTEXTHELP	Set this flag to display a context-sensitive help button on the wizards caption bar.
PSH_WIZARDHASFINISH	Set this flag to display a Finish button on all pages. Typically, wizards only display Back, Next or Finish, and Cancel, and replace Next with Finish on the completion page. If this flag is set, every page will display all four buttons.

The **pszCaption** member of the **PROPSHEETHEADER** structure is ignored. Instead, the wizard displays the caption that is specified in the current page's dialog box template.

If you have created an array of HPROPSHEETPAGE handles for your pages, assign the array to the **phpage** member. If you have instead created an array of **PROPSHEETPAGE** structures, assign the array to the **ppsp** member and set the PSH_PROPSHEETPAGE flag in the **dwFlags** member.

The following example assigns values to psh, a **PROPSHEETHEADER** structure and calls the **PropertySheet** function to launch the wizard. The Wizard97-style wizard has both watermark and header graphics, specified by their resource IDs. The *ahpsp* array contains all the HPROPSHEETPAGE handles and defines the default order in which they will be displayed.

```
psh.dwSize =          sizeof(psh);
psh.hInstance =       hInstance;
psh.hwndParent =      NULL;
psh.phpage =          ahpsp;
psh.dwFlags =         PSH_WIZARD97|PSH_WATERMARK|PSH_HEADER;
psh.pszbmWatermark = MAKEINTRESOURCE(IDB_WATERMARK);
psh.pszbmHeader =     MAKEINTRESOURCE(IDB_BANNER);
psh.nStartPage =      0;
psh.nPages =          4;

PropertySheet(&psh);
```

The Dialog Box Procedure

Once the wizard has been launched, each page will need a dialog box procedure to process Windows messaging, particularly notifications from its controls and the wizard. The dialog box procedure will receive the usual WM_XXX Windows messages. The three that virtually all wizards will need to handle are WM_INITDIALOG, WM_DESTROY, and WM_NOTIFY.

Handling WM_INITDIALOG and WM_DESTROY

When a page is about to be displayed for the first time, its dialog box procedure receives a WM_INITDIALOG message. Handling this message allows the wizard to do any needed initialization tasks. For example, the sample code discussed later handles this message to set the font for the welcome and completion page titles.

The WM_INITDIALOG message's *lParam* value points to a copy of the page's **PROPSHEETPAGE** structure. The **lParam** member of this structure is ignored by the system. It typically points to an application-defined shared data structure that is used to store persistent data and pass data from one page to another.

To preserve access to the shared data structure for later use, load it's pointer into the pages' user data by calling the **GetWindowLong** function with an index of GWL_USERDATA. You can then use **SetWindowLong** to retrieve the pointer when needed. The **Wiz97 sample** has a simple example of how to use a shared data structure.

Note Do not attempt to modify any members of the **PROPSHEETPAGE** structure other than **lParam**. Doing so will have unpredictable consequences.

When the property sheet is destroyed, you will receive a WM_DESTROY message. The wizard is automatically destroyed by the system, but handling this message allows you to do any needed cleanup.

Handling WM_NOTIFY

Notifications from the wizard come in the form of a WM_NOTIFY message. You will receive this message before the page is displayed and when any of the wizard's buttons are clicked. The *lParam* parameter of the message is a pointer to a **NMHDR** header structure. The notification's ID is contained in the structure's **code** member. The four that most wizards will need to handle are:

Code	Description
PSN_SETACTIVE	Sent before the page is displayed.
PSN_WIZBACK	Sent when the Back button is clicked.
PSN_WIZNEXT	Sent when the Next button is clicked.
PSN_WIZFINISH	Sent when the Finish button is clicked.

Handling PSN_SETACTIVE

The PSN_SETACTIVE notification message is sent each time a page is about to be made visible. The first time a page is visited, PSN_SETACTIVE follows the WM_INITDIALOG message. If the page is subsequently revisited, it will receive only a PSN_SETACTIVE notification. This notification is usually handled to initialize data for the page and enable the appropriate buttons.

By default, the wizard displays Back, Next, and Cancel buttons, with all buttons enabled. To disable a button or display Finish instead of Next, you must send a **PSM_SETWIZBUTTONS** message. Once this message has been sent, the state of the buttons will be preserved until it is modified by another PSM_SETWIZBUTTONS message, even if a new page is selected. Typically, all PSN_SETACTIVE handlers send this message to ensure that each page has the correct button state.

You can change the button state with this message at any time. For example, you may want the Next button to be initially disabled. Once a user has entered all the necessary information, you can send another PSM_SETWIZBUTTONS message to enable the Next button and let the user proceed to the next page.

The following code fragment uses the **PropSheet_SetWizButtons** macro to enable the Back and Next buttons on an interior page before it is displayed:

```
case WM_NOTIFY :
{
    LPNMHDR pnmh = (LPNMHDR)lParam;
    switch(pnmh->code)
    {
    ...
    case PSN_SETACTIVE :
        ...
        //this is an interior page
        PropSheet_SetWizButtons(hwnd, PSWIZB_NEXT | PSWIZB_BACK);
        ...
    }
...
}
```

Handling PSN_WIZNEXT, PSNWIZBACK, and PSN_WIZFINISH

When a Next or Back button is clicked, you will receive a PSN_WIZNEXT or PSN_WIZBACK notification message. By default, the wizard will automatically go to either the next or previous page in the order that is defined when the property sheet is created. A common reason to handle these notifications is to prevent the user from switching pages, or to override the default page order.

To prevent the user from switching pages, handle the button notification, call the **SetWindowLong** function with the DWL_MSGRESULT value set to –1, and return TRUE. For example:

```
case PSN_WIZNEXT :
    ...
    //Don't go to next page yet.
    SetWindowLong(hwnd, DWL_MSGRESULT, -1);
    return TRUE;
    ...
```

To override the standard order and go to a particular page, call **SetWindowLong** with the DWL_MSGRESULT value set to the page's dialog box resource ID, and return TRUE. For example:

```
case PSN_WIZNEXT :
    ...
    //Go straight to the completion page.
    SetWindowLong(hwnd, DWL_MSGRESULT, IDD_FINISH);
    return TRUE;
    ...
```

When the Finish or Cancel buttons are clicked, you will receive a PSN_WIZFINISH or PSN_RESET notification message, respectively. When either of these buttons is clicked, the wizard will be automatically destroyed by the system. However, you can handle these notifications if you need to perform cleanup tasks before the wizard is destroyed. To prevent the wizard from being destroyed when you receive a PSN_WIZFINISH notification, call **SetWindowLong** with the DWL_MSGRESULT value set to TRUE, and return TRUE. For example:

```
case PSN_WIZFINISH :
    ...
    //Not finished yet.
    SetWindowLong(hwnd, DWL_MSGRESULT, TRUE);
    return TRUE;
    ...
```

Backward Compatible Wizards

The preceding section assumes that you are implementing a wizard for a system with version 5 or later of the Common Controls. You will find this version on systems with Windows 2000 or later or any Windows system with Internet Explorer 5 or later.

If you are writing a wizard for systems with earlier versions of the Common Controls, many of the features discussed in the preceding section will not be available. A number of the members of the **PROPSHEETHEADER** and **PROPSHEETPAGE** structures that are used by the Wizard97 style are only supported by Common Controls version 5 and later. However, it is still possible to implement a *backward compatible* wizard with much the same look and feel as the Wizard97 style. To do so, you must explicitly implement the following:

- Add the watermark graphic to the dialog box template for your welcome and completion pages.
- Make all your templates same size. There is no separate system-defined header area for interior pages.
- Create the interior page's header area explicitly on your templates.
- Do not use a header graphic because it may conflict with the title or subtitle if the wizard changes size.

For further discussion of backward-compatible wizards, see the *Backward-Compatible Wizards Specification* elsewhere in the Platform SDK.

A Sample Wizard Application

This section discusses Wiz97, a simple stand-alone wizard application that demonstrates the essentials of how to implement a Wizard97-style wizard. The focus is on how to create exterior and interior pages and navigate between them. There is only a limited discussion about how to design and implement individual dialog boxes. Although the sample dialog boxes have controls, only a few control notification messages are handled, largely to illustrate how to use certain features of wizards. For links to the source code, see the Wiz97 source files.

The Wiz97 wizard has a welcome page, a completion page, and two interior pages. The welcome page has a simple watermark and two static controls to hold the title and explanatory text.

The first interior page has a group box with three radio buttons and a check box. Note that the Next button is initially disabled. It is only enabled after one of the radio buttons or the check box is selected. If the check box is selected, clicking Next skips directly to the completion page instead of the second interior page.

The second interior page has three edit boxes, provided for illustration purposes only. None of the boxes have handlers.

The completion page is very similar to the welcome page. The text in the static controls is different, and the Next button is replaced by a Finish button. If the check box on the first interior page was selected, clicking Back will return to that page. If the check box was not selected at all, or was cleared, clicking Back will return to the second interior page.

The remainder of this section describes the implementation of the Wiz97 application. The complete sample code can be found under the Common Controls Samples topic or by clicking Wiz97 sample code. To make the code as readable as possible, all of the source code other than the dialog box procedures is in the **WinMain** function. For the same reason, error checking has been omitted. You should use whatever error checking is appropriate for your wizard application.

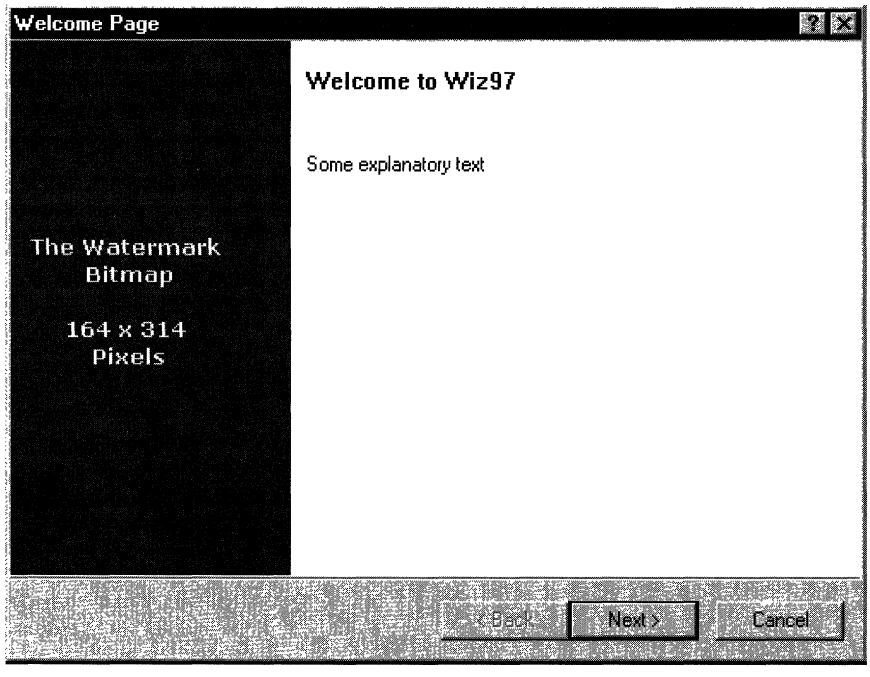


Figure 11-5: The Wiz97 welcome page.

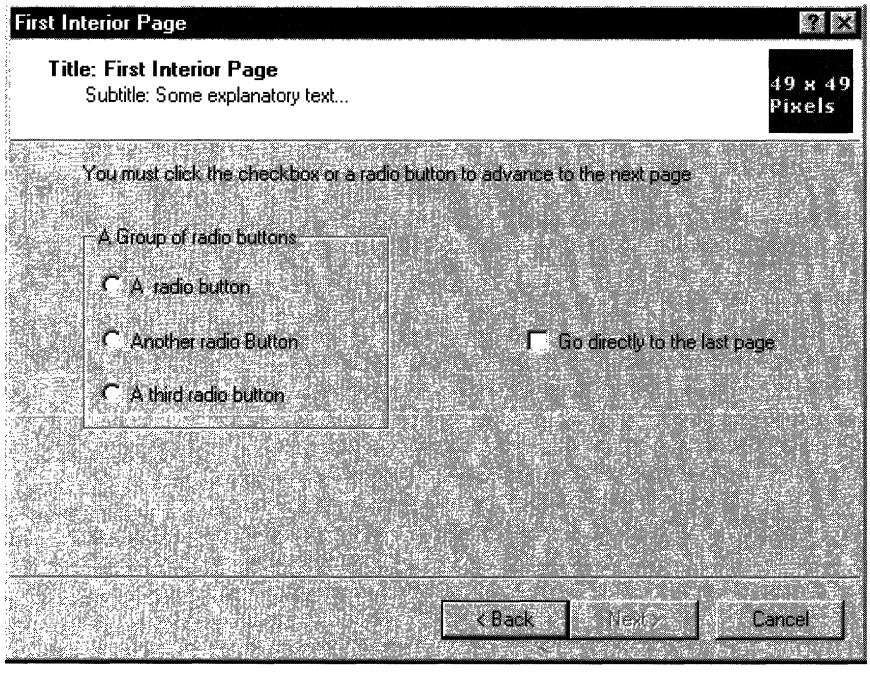


Figure 11-6: The first Wiz97 interior page.

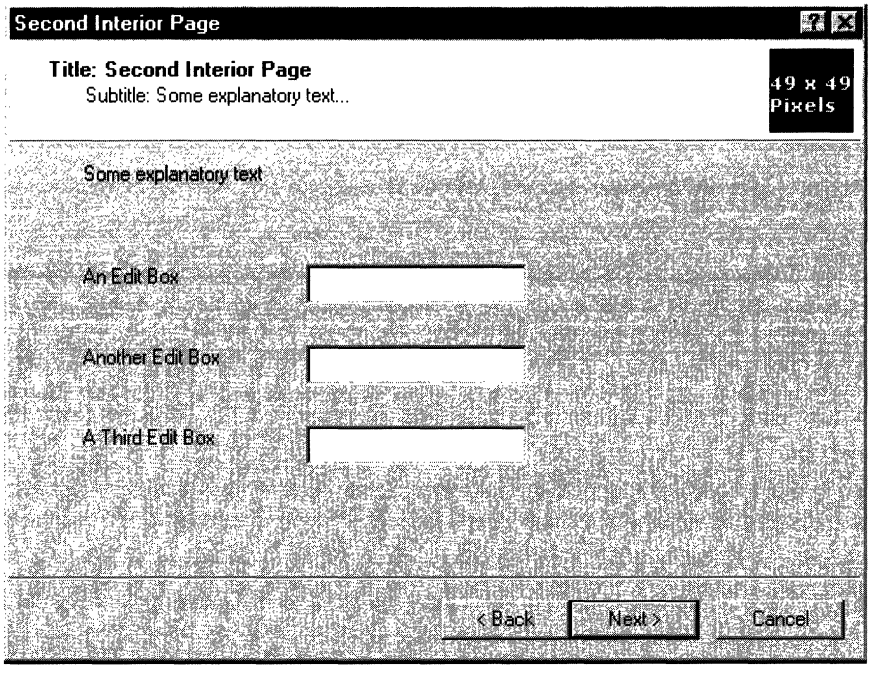


Figure 11-7: The second Wiz97 interior page.

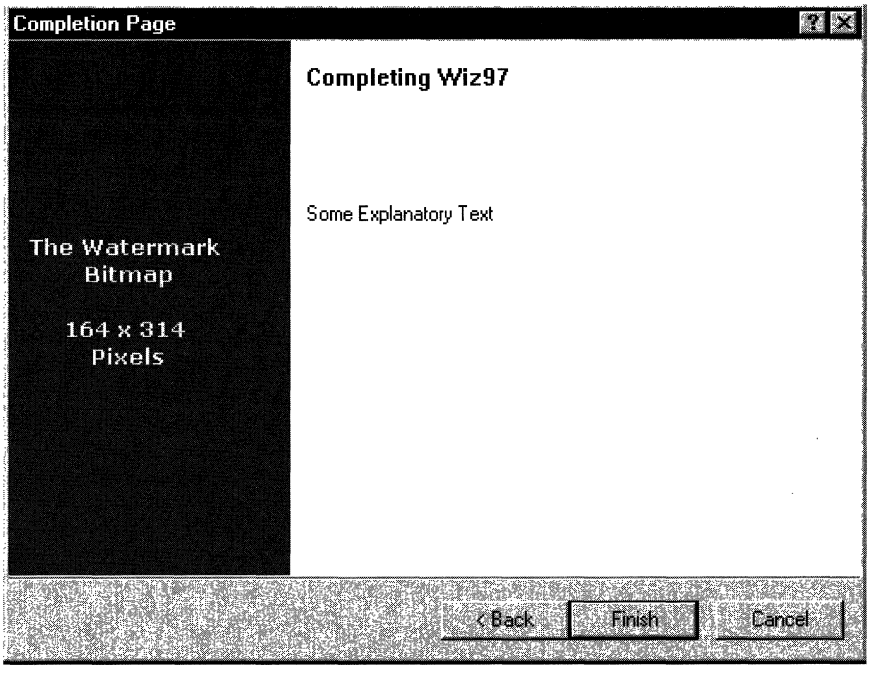


Figure 11-8: The Wiz97 completion page.

Note that this sample must be run on a system with version 5.80 of the Common Controls. Attempting to run it on earlier versions will have unpredictable results. For information (and sample code) about how to determine which version is present, see *Shell and Common Control Versions* .

Designing the Templates

The mechanics of creating a dialog box template is largely outside the scope of this document. For a general discussion of how to place your controls, specify bitmap colors and dimensions, and handle a number of other design issues, see the *Wizard97 Specification*. Here are some specific points to keep in mind when planning your template design:

- The template doesn't include the band at the bottom of the wizard that holds the buttons.
- For interior pages, the template also doesn't include the header area. Interior page templates are thus smaller than those for welcome and completion pages.
- The template does not include the watermark bitmap. Just leave the area reserved for the watermark empty. The watermark bitmap is specified when you fill the **PROPSHEETHEADER** structure.
- Remember that the color of the watermark's upper-left pixel will be used for a fill color if the dialog box is enlarged.

Creating the Wizard Pages

Creating the wizard pages primarily involves assigning values to the members of a **PROPSHEETPAGE** structure and calling **CreatePropertySheetPage** to create the page's HPROPSHEETPAGE handle. The following code defines the welcome page by assigning values to a **PROPSHEETPAGE** structure named *psp*.

```
psp.dwSize =      sizeof(psp);
psp.dwFlags =     PSP_DEFAULT|PSP_HIDEHEADER;
psp.hInstance =   hInstance;
psp.lParam =      (LPARAM) &wizdata; //shared data
                                     //structure
psp.pfnDlgProc =  IntroDlgProc;
psp.pszTemplate = MAKEINTRESOURCE(IDD_INTRO);

ahpsp[0] =        CreatePropertySheetPage(&psp);
```

Setting the PSP_HIDEHEADER flag defines this page as exterior. The structure's **lParam** member points to an application-defined SHAREDWIZDATA structure. Because the Wiz97 wizard application assigns a pointer to this structure to the **lParam** member of the **PROPSHEETPAGE** structure used to create all its pages, the SHAREDWIZDATA structure can be used to store shared data. The SHAREDWIZDATA structure is used to

store the HFONT handle used by the welcome and completion pages, and the state of the radio buttons and check box on the first interior page. It is declared as:

```
typedef struct SHAREDWIZDATA {
    HFONT hTitleFont;
    BOOL IsBoxChecked;
    BOOL IsButtonClicked;
} SHAREDWIZDATA, *LPSHAREDWIZDATA;
```

The welcome page's box dialog procedure is assigned to IntroDlgProc. It is a generally a good practice for each page to have its own dialog box procedure. Even a page with only static controls may still need to handle the wizard button notification messages.

When **CreatePropertySheetPage** is called, the HPROPSHEETPAGE handle is assigned to *ahpsp [0]*. The *ahpsp* array, which is an array of HPROPSHEETPAGE handles, is used later to create the property sheet. The array also determines the default order that the pages will be displayed.

The following example shows how to create the first interior page.

```
    psp.dwFlags =          PSP_DEFAULT|PSP_
USEHEADERTITLE|PSP_USEHEADERSUBTITLE;
    psp.pszHeaderTitle =   MAKEINTRESOURCE(IDS_TITLE1);
    psp.pszHeaderSubTitle = MAKEINTRESOURCE(IDS_SUBTITLE1);
    psp.pszTemplate =      MAKEINTRESOURCE(IDD_INTERIOR1);
    psp.pfnDlgProc =       IntPage1DlgProc;

    ahpsp[1] =             CreatePropertySheetPage(&psp);
```

The **dwSize**, **hInstance**, and **lParam** members are the same for all pages, and are left unchanged. The PSP_HIDEHEADER flag is not set, so the page is interior. Setting the PSP_USEHEADERTITLE and PSP_USEHEADERSUBTITLE flags enables the system to place a title and subtitle in the header area. These titles are specified by assigning text strings, from a string table in this instance, to **pszHeaderTitle** and **pszHeaderSubTitle**. Assigning the HPROPSHEETPAGE handle to *ahpsp[1]* places the page second in the default display order.

The code used to create the second interior and completion pages is very similar to the first interior and welcome pages, respectively. The only difference is that different titles, dialog box procedures, and so on, are assigned to the members of the **PROPSHEETPAGE** structure, and the page's HPROPSHEETPAGE handle is assigned to a different element of *ahpsp* array.

Creating the Property Sheet

Creating the property sheet is similar to creating the individual pages. Assign values to members of a **PROPSHEETHEADER** structure, and then pass the structure to **PropertySheet** to launch the wizard. The following code defines the Wiz97 property sheet by assigning values to a **PROPSHEETHEADER** structure named *psh*.

```
psh.dwSize =          sizeof(psh);
psh.hInstance =       hInstance;
psh.hwndParent =      NULL;
psh.phpage =          ahpsp;
psh.dwFlags =         PSH_WIZARD97|PSH_WATERMARK|PSH_HEADER;
psh.pszbmWatermark = MAKEINTRESOURCE(IDB_WATERMARK);
psh.pszbmHeader =    MAKEINTRESOURCE(IDB_BANNER);
psh.nStartPage =      0;
psh.nPages =          4;
```

The PSH_WIZARD97 flag gives this property sheet the Wizard97 style. Setting PSH_WATERMARK and PSH_HEADER enables the system to add a watermark bitmap to the exterior pages, and a smaller bitmap to the header area of the interior pages. The watermark and header bitmaps, which are identified by their resource IDs, are assigned to the **pszbmWatermark** and **pszbmHeader** members, respectively. The handles to the individual pages are passed to the property sheet by assigning the *ahpsp* array to the **phpage** member.

Creating a Title Font

The *Wizard97 Specification* uses a 12 point Verdana Bold font for the titles of the welcome and completion pages. The Wiz97 application creates the font, and then assigns its HFONT handle to the hTitleFont member of the shared SHAREDWIZDATA structure.

The welcome and completion page's dialog box procedures extract a pointer to this structure when initializing the page. They then use the hTitleFont member to set the font for the static control that contains the title. The font is destroyed when the wizard shuts down, so it is created and destroyed only once. The following example shows how to create the font.

```
NONCLIENTMETRICS ncm = {0};
ncm.cbSize = sizeof(ncm);
SystemParametersInfo(SPI_GETNONCLIENTMETRICS, 0, &ncm, 0);
LOGFONT TitleLogFont = ncm.lfMessageFont;
TitleLogFont.lfWeight = FW_BOLD;
lstrcpy(TitleLogFont.lfFaceName, TEXT("Verdana Bold"));

HDC hdc = GetDC(NULL);
INT FontSize = 12;
TitleLogFont.lfHeight = 0 - GetDeviceCaps(hdc, LOGPIXELSY) * FontSize /
72;
wizdata.hTitleFont = CreateFontIndirect(&TitleLogFont);
ReleaseDC(NULL, hdc);
```

The Dialog Box Procedures

Once the wizard is launched, most of the remainder of the application is handled by the dialog box procedures. Even with no active controls, pages still need to handle messages such as WM_INITDIALOG, and notifications from the Back, Next, Cancel, and Finish buttons. To simplify this task, each Wiz97 page has its own dialog box procedure.

The Welcome Page

The welcome page only has static controls, so its dialog box procedure is quite simple. The following example shows the code for the procedure:

```
BOOL CALLBACK IntroDlgProc (
                                        HWND hwndDlg,
                                        UINT uMsg,
                                        WPARAM wParam,
                                        LPARAM lParam
                                        )
{
    LPSHAREDWIZDATA pdata = (LPSHAREDWIZDATA) GetWindowLong(hwndDlg,
GWL_USERDATA);

    switch (uMsg)
    {
    case WM_INITDIALOG :
        {
            pdata = (LPSHAREDWIZDATA) ((LPPROPSHEETPAGE) lParam) -> lParam;

            SetWindowLong(hwndDlg, GWL_USERDATA, (DWORD_PTR) pdata);

            HWND hwndControl = GetDlgItem(hwndDlg, IDC_TITLE);
            SetWindowFont(hwndControl,pdata->hTitleFont, TRUE);
            break;
        }

    case WM_NOTIFY :
        {
            LPNMHDR lpnm = (LPNMHDR) lParam;

            switch (lpnm->code)
            {
            case PSN_SETACTIVE : //Enable the Next button
                PropSheet_SetWizButtons(GetParent(hwndDlg), PSWIZB_NEXT);
                break;
            default :
                break;
```

(continued)

(continued)

```
            }
        }
        break;

    default:
        break;
    }
    return 0;
}
```

The first time a page's dialog box procedure is called, it will receive a WM_INITDIALOG message. The message handler extracts the pointer to the shared SHAREDWIZDATA structure from the **lParam** member of the page's **PROPSHEETPAGE** structure. It then uses **SetWindowLong** to assign the pointer to the page's user data. Each time the dialog box procedure is subsequently called, it uses **GetWindowLong** to extract the SHAREDWIZDATA pointer for later use. After storing the pointer, the message handler uses the hTitleFont member of the SHAREDWIZDATA structure to set the font for the static control that contains the title.

Following the WM_INITDIALOG message and each time a page is subsequently selected, its dialog box procedure receives a PSN_SETACTIVE notification message. This notification is sent before the page is displayed and is used for initialization. The welcome page's PSN_SETACTIVE notification handler uses the **PropSheet_SetWizButtons** macro to send a PSM_SETWIZBUTTONS message to enable the Next button. Because the wizard automatically takes care of displaying the next page, the Next button notification message is not handled. Although not shown here, the sample code has token button handlers as placeholders.

The Interior Pages

The first interior page is initially displayed with the Next button disabled. It is only enabled after a radio button or the check box is selected. If a user selects the check box, clicking Next takes the user directly to the completion page. The following example shows the dialog box procedure for the interior page:

```
BOOL CALLBACK IntPage1DlgProc (
                            HWND hwndDlg,
                            UINT uMsg,
                            WPARAM wParam,
                            LPARAM lParam
                            )
{
    LPSHAREDWIZDATA pdata = (LPSHAREDWIZDATA) GetWindowLong
(hwndDlg, GWL_USERDATA);

    switch (uMsg)
```

```
    {
    case WM_INITDIALOG :
        {
            pdata = (LPSHAREDWIZDATA) ((LPPROPSHEETPAGE) lParam) -> lParam;
            SetWindowLong(hwndDlg, GWL_USERDATA, (DWORD_PTR) pdata);
            break;
        }

    case WM_COMMAND :
        switch(LOWORD (wParam))
        {
        case IDC_CHECK1 :
            pdata->IsBoxChecked = !(pdata->IsBoxChecked);
            break;

        case IDC_RADIO1 :

        case IDC_RADIO2 :

        case IDC_RADIO3 :
            pdata->IsButtonClicked = TRUE;
            break;

        default :
            break;
        }

        if(pdata->IsBoxChecked || pdata->IsButtonClicked)
        {
            PropSheet_SetWizButtons(GetParent(hwndDlg),
PSWIZB_BACK | PSWIZB_NEXT);
        }

        else
        {
            PropSheet_SetWizButtons(GetParent(hwndDlg), PSWIZB_BACK);
        }
        break;

    case WM_NOTIFY :
        {
        LPNMHDR lpnm = (LPNMHDR) lParam;

        switch (lpnm->code)
            {
```

(continued)

(continued)

```
            case PSN_SETACTIVE : //Enable the appropriate buttons.
                if(pdata->IsBoxChecked || pdata->IsButtonClicked)
                {
                    PropSheet_SetWizButtons(GetParent
(hwndDlg), PSWIZB_BACK | PSWIZB_NEXT);
                }

                else //Otherwise, only enable Back.
                {
                    PropSheet_SetWizButtons(GetParent(hwndDlg), PSWIZB_BACK);
                }
                break;

            case PSN_WIZNEXT :

                if(pdata->IsBoxChecked)
                {
                    SetWindowLong(hwndDlg, DWL_MSGRESULT, IDD_END);
                    return TRUE;
                }
                break;

            default :
                break;
        }
    }
    break;

  default:
    break;
  }
  return 0;
}
```

The WM_INITDIALOG handler simply extracts the pointer to the shared SHAREDWIZDATA structure and assigns it to the page's user data. Initially, the PSN_SETACTIVE handler enables just the Back button. The Next button is only enabled after the user selects a radio button or the check box. If the check box has been selected, the PSN_WIZNEXT handler uses **SetWindowLong** to jump directly to the final page.

The **IsBoxChecked** member of the shared SHAREDWIZDATA structure is used to store the state of the check box. The value of the **IsButtonClicked** member indicates whether one of the radio buttons has been selected. Storing these state parameters in the SHAREDWIZDATA structure serves two purposes. One is to make this state information

persistent, so that it is available if the page is revisited. The other is to make the state information available to other pages.

The second interior page does not implement any of the three edit boxes, so its dialog box procedure is minimal. As with the other dialog box procedures, its WM_INITDIALOG message handler assigns the pointer to the shared SHAREDWIZDATA structure to its user data. Its PSN_ACTIVE notification handler enables the Next and Back buttons. For more information, see the Wiz97 source code.

The Completion Page

The dialog box procedure for the completion page is similar to that for the welcome page. The following example shows the procedure code:

```
BOOL CALLBACK EndDlgProc (
    HWND hwndDlg,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam
    )
{
    LPSHAREDWIZDATA pdata = (LPSHAREDWIZDATA) GetWindowLong
(hwndDlg, GWL_USERDATA);

    switch (uMsg)
    {
    case WM_INITDIALOG :
        {
            pdata = (LPSHAREDWIZDATA) ((LPPROPSHEETPAGE) lParam) -> lParam;
            SetWindowLong(hwndDlg, GWL_USERDATA, (DWORD_PTR) pdata);

            HWND hwndControl = GetDlgItem(hwndDlg, IDC_TITLE);
            SetWindowFont(hwndControl,pdata->hTitleFont, TRUE);
            break;
        }

    case WM_NOTIFY :
        {
        LPNMHDR lpnm = (LPNMHDR) lParam;

        switch (lpnm->code)
            {
            case PSN_SETACTIVE :
```

(continued)

(continued)

```
                PropSheet_SetWizButtons(GetParent(hwndDlg),
PSWIZB_BACK | PSWIZB_FINISH);
                break;

            case PSN_WIZBACK :

                if(pdata->IsBoxChecked)
                {
                    SetWindowLong(hwndDlg, DWL_MSGRESULT, IDD_INTERIOR1);
                    return TRUE;
                }
                break;
            default :
                break;
            }
        }
        break;

    default:
        break;
    }
    return 0;
}
```

As with the welcome page, the WM_INITDIALOG handler extracts the shared SHAREDWIZDATA structure and stores the pointer in the page's user data. It then uses the hTitleFont member to set the font for the static control holding the title. The PSN_SETACTIVE handler enables the Back and Finish buttons.

The PSN_WIZBACK handler checks the IsBoxChecked member of the shared SHAREDWIZDATA structure to see if the check box on the second interior page is selected. If it is, the handler takes the user back to the first interior page, instead of the second interior page. As long as that check box is selected, the user will never see the second interior page.

Because there is no need for any cleanup, there is no PSN_WIZFINISH handler. The wizard is automatically destroyed when the Finish button is clicked. After the wizard is destroyed, control is returned to the **WinMain** function, which destroys the title font by using the **DeleteObject** function and then returns.

CHAPTER 12

Date and Time Picker Controls

A *date and time picker (DTP) control* provides a simple and intuitive interface through which to exchange date and time information with a user. For example, with a DTP control you can ask the user to enter a date and then retrieve his or her selection with ease.

About Date and Time Picker Controls

Date and time picker controls are implemented in version 4.70 and later of Comctl32.dll. To create a DTP control call **CreateWindowEx** and specify DATETIMEPICK_CLASS as the window class. The class is registered when the date and time picker class is loaded from the common control dynamic-link library (DLL). Register this class by calling the **InitCommonControlsEx** function, while specifying the ICC_DATE_CLASSES bit flag in the accompanying **INITCOMMONCONTROLSEX** structure.

Note Windows does not support dates prior to 1601. See ***FILETIME*** for details.
The DTP control is based on the Gregorian calendar, which was introduced in 1753. It will not calculate dates that are consistent with the Julian calendar that was in use prior to 1753.

Date and Time Picker User Interface

The client area of a date and time picker control displays date and time information and acts as the interface through which users modify the information. The control's display consists of fields that are defined by the control's format string. Additionally, the control will display a check box when the **DTS_SHOWNONE** style is in use.

Each field in the control displays a portion of the date and time information that the control stores internally. The user can click a field to set keyboard focus and then provide keyboard input to change the information represented by that field. The DTP control automatically updates internal information based on the user's input. The control recognizes the following as valid input.

Input Category	Description
Arrow Keys	The control accepts arrow keys to navigate the fields in the control and change values. The user can press the LEFT ARROW or RIGHT ARROW key to move through the control. If the user attempts to move past the last field in a given direction, the keyboard focus "wraps around" to the field on the opposite side of the control. The UP ARROW and DOWN ARROW keys change values in the current field incrementally.

(continued)

(continued)

Input Category	Description
End and Home	The control accepts the VK_END and VK_HOME virtual keys to change the value within the current field to its upper and lower limits, respectively.
Function Keys	The F2 key activates edit mode. The F4 key causes the control to display a drop-down month calendar control (pressing ALT+DOWN ARROW does this, too).
Numbers	The control accepts numeric input in two-character segments. If the value entered by the user is invalid (like setting the month to 14), the control rejects it and resets the display to the previous value.
Plus and Minus	The control accepts the VK_ADD and VK_SUBTRACT virtual keys from the numeric keypad to increment and decrement the value in the current field.

DTP controls that do not use the **DTS_UPDOWN** style display an arrow button. If the user clicks this button, a month calendar control drops down. The user can select a specific date by clicking an area of the calendar.

Date and Time Picker Control Styles and Formats

Date and time picker controls have several styles that determine a control's appearance and behavior. Specify the style when creating the control with the *dwStyle* parameter of **CreateWindowEx**. To retrieve or change the window style after you have created the control, use **GetWindowLong** and **SetWindowLong**.

Preset Formats

There are three preset formats available for displaying the date and one for displaying time. Set these formats by choosing one of the following window styles:

DTS_LONGDATEFORMAT
The display will look like: "Friday, April 19, 1996".

DTS_SHORTDATEFORMAT
The display will look like: "4/19/96".

DTS_SHORTDATECENTURYFORMAT
Version 5.80. The display will look like: "4/19/1996".

DTS_TIMEFORMAT
The display will look like: "5:31:42 PM".

Custom Formats

A DTP control relies on a format string to determine how it will display fields of information. If the preset formats are not sufficient, you can create a custom format

by defining your own format string. Custom formats provide greater flexibility for an application. They allow you to specify the order in which the control will display fields of information. You can include body text as well as callback fields for requesting information from the user. Once the string is created, you assign it to the DTP control with a **DTM_SETFORMAT** message.

Format Strings

A DTP format string consists of a series of elements that represent a particular piece of information and define its display format. The elements will be displayed in the order they appear in the format string.

Date and time format elements will be replaced by the actual date and time. They are defined by the following groups of characters:

Element	Description
"d"	The one- or two-digit day.
"dd"	The two-digit day. Single-digit day values are preceded by a zero.
"ddd"	The three-character weekday abbreviation.
"dddd"	The full weekday name.
"h"	The one- or two-digit hour in 12-hour format.
"hh"	The two-digit hour in 12-hour format. Single-digit values are preceded by a zero.
"H"	The one- or two-digit hour in 24-hour format.
"HH"	The two-digit hour in 24-hour format. Single-digit values are preceded by a zero.
"m"	The one- or two-digit minute.
"mm"	The two-digit minute. Single-digit values are preceded by a zero.
"M"	The one- or two-digit month number.
"MM"	The two-digit month number. Single-digit values are preceded by a zero.
"MMM"	The three-character month abbreviation.
"MMMM"	The full month name.
"t"	The one-letter AM/PM abbreviation (that is, AM is displayed as "A").
"tt"	The two-letter AM/PM abbreviation (that is, AM is displayed as "AM").
"yy"	The last two digits of the year (that is, 1996 would be displayed as 96).
"yyyy"	The full year (that is, 1996 would be displayed as "1996").

To make the information more readable, you can add body text to the format string by enclosing it in single quotation marks. Spaces and punctuation marks do not need to be enclosed within quotation marks.

Note Nonformat characters that are not delimited by single quotation marks will result in unpredictable display by the DTP control.

For example, to display the current date with the format "'Today is: 04:22:31 Tuesday Mar 23, 1996", the format string is "'Today is: 'hh':'m':'s dddd MMM dd', 'yyyy". To include a single quotation mark in your body text, use two consecutive single quotation marks. For example, "'Don't forget' MMM dd',' yyyy" produces output that looks like: Don't forget Mar 23, 1996. It is not necessary to use quotation marks with the comma, so "'Don't forget' MMM dd, yyyy" is also valid, and produces the same output.

Callback Fields

In addition to the standard format characters and body text, you also can define certain parts of the display as callback fields. These fields can be used to query the user for information. To declare a callback field, include one or more "X" characters (ASCII Code 88) anywhere in the format string. You can create callback fields that have a unique identity by repeating the "X" character. Thus, the format string "XX dddd MMM dd', 'yyy XXX" contains two unique callback fields, "XX" and "XXX". Like other DTP control fields, callback fields are displayed in left-to-right order based on their location in the format string.

When the DTP control parses the format string and encounters a callback field, it sends **DTN_FORMAT** and **DTN_FORMATQUERY** notification messages. The format string element corresponding to the callback field is included with the notifications to allow the receiving application to determine which callback field is being queried. The owner of the control must respond to these notifications to ensure that the custom information is properly displayed.

Date and Time Picker Control Notification Messages

A date and time picker control sends notification messages when it receives user input or processes and reacts to callback fields. The parent of the control receives these notification messages in the form of **WM_NOTIFY** messages.

The following notification messages are used with DTP controls:

Notification	Description
DTN_CLOSEUP	Indicates that the drop-down month calendar is about to be removed.
DTN_DATETIMECHANGE	Signals a change within the DTP control.
DTN_DROPDOWN	Indicates that the drop-down month calendar is about to be displayed.
DTN_FORMAT	Requests text to display in a portion of the format string described as a callback field.

Notification	Description
DTN_FORMATQUERY	Requests information about the maximum allowable size of the text to be displayed in a callback field.
DTN_USERSTRING	Signals the end of a user's edit operation within the control. This notification is sent only by DTP controls that use the **DTS_APPCANPARSE** style.
DTN_WMKEYDOWN	Signals that the user has pressed a key in a callback field of the DTP control.

Using Date and Time Picker Controls

This section provides information and sample code for implementing a date and time picker control.

Creating a Date and Time Picker Control

To create a date and time picker control, use the **CreateWindowEx** function, specifying DATETIMEPICK_CLASS as the window class. You first must register the window class by calling the **InitCommonControlsEx** function, while specifying the ICC_DATE_CLASSES bit in the accompanying **INITCOMMONCONTROLSEX** structure.

The following example creates a DTP control in an existing modeless dialog box. It uses the **DTS_SHOWNONE** style to allow the user to simulate deactivation of the date within the control.

```
//  CreateDatePick creates a DTP control within a dialog box.
//  Returns the handle to the new DTP control if successful,
//  or NULL otherwise.
//
//    hwndMain - The handle to the main window.
//    g_hinst - global handle to the program instance.
HWND WINAPI CreateDatePick(hwndMain)
{
    HWND hwndDP = NULL;
    INITCOMMONCONTROLSEX icex;

    icex.dwSize = sizeof(icex);
    icex.dwICC = ICC_DATE_CLASSES;

    InitCommonControlsEx(&icex);

    hwndDlg = CreateDialog  (g_hinst,
        MAKEINTRESOURCE(IDD_DIALOG1),
          hwndMain,
```

(continued)

(continued)

```
        DlgProc);

    if(hwndDlg)
        hwndDP = CreateWindowEx(0,
            DATETIMEPICK_CLASS,
            "DateTime",
            WS_BORDER|WS_CHILD|WS_VISIBLE|DTS_SHOWNONE,
    20,50,220,20,
    hwndDlg,
    NULL,
    g_hinst,
    NULL);
    return (hwndDP);
}
```

Processing Date and Time Picker Notifications

The following example processes the **DTN_DATETIMECHANGE**, **DTN_FORMAT**, **DTN_FORMATQUERY**, and **DTN_WMKEYDOWN** notifications by calling the **DoDateTimeChange**, **DoFormatQuery**, **DoFormat**, and **DoWMKeydown** application-defined functions, respectively.

Other topics in this chapter provide additional information on these notifications. See *Supporting Callback Fields in a DTP Control* and *Processing the DTN_DATETIMECHANGE Notification Message* for additional information.

```
BOOL WINAPI DoNotify(HWND hwnd, WPARAM wParam, LPARAM lParam)
{
    LPNMHDR hdr = (LPNMHDR)lParam;

    switch(hdr->code){
        case DTN_DATETIMECHANGE:{
            LPNMDATETIMECHANGE lpChange =(LPNMDATETIMECHANGE)lParam;
            DoDateTimeChange(lpChange);
        }
        break;

        case DTN_FORMATQUERY:{
            LPNMDATETIMEFORMATQUERY lpDTFQuery =(LPNMDATETIMEFORMATQUERY)lParam;

            // Process DTN_FORMATQUERY to ensure that the
            // control displays callback information
            // properly.
            DoFormatQuery(hdr->hwndFrom, lpDTFQuery);
        }
        break;
```

```
        case DTN_FORMAT:{
            LPNMDATETIMEFORMAT lpNMFormat =(LPNMDATETIMEFORMAT) lParam;

            // Process DTN_FORMAT to supply information
            // about callback fields (fields) in the DTP
            // control.
            DoFormat(hdr->hwndFrom, lpNMFormat);
        }
        break;

        case DTN_WMKEYDOWN:{
            LPNMDATETIMEWMKEYDOWN lpDTKeystroke =
                (LPNMDATETIMEWMKEYDOWN)lParam;

            // Process DTN_WMKEYDOWN to respond to a user's
            // keystroke in a callback field.
            DoWMKeydown(hdr->hwndFrom, lpDTKeystroke);
        }
        break;
    }
    // All of the above notifications require the owner to
    // return zero.
    return FALSE;
}
```

Processing the DTN_DATETIMECHANGE Notification

A date and time picker control sends the **DTN_DATETIMECHANGE** message whenever a change occurs. This message will be generated if the user changes one of the fields in the control or changes the state of the control's check box (**DTS_SHOWNONE** only).

The following example is an application-defined function designed to update a static control within a dialog box. The text within the static control is changed to reflect the current state of the DTP control:

```
void WINAPI DoDateTimeChange(LPNMDATETIMECHANGE lpChange)
{
    // If the user has unchecked the DTP's check box,
    // change the text in a static control to show the
    // appropriate message.
    //
    // g_hwndDlg - a program-global address of a dialog box.

    if(lpChange->dwFlags == GDT_NONE)
        SetDlgItemText(g_hwndDlg, IDC_STATUS, "Disabled");
    else
        SetDlgItemText(g_hwndDlg, IDC_STATUS, "Active");
}
```

Supporting Callback Fields in a DTP control

If you plan to use callback fields with the date and time picker controls in your application, you must be prepared to handle **DTN_FORMATQUERY**, **DTN_FORMAT**, and **DTN_WMKEYDOWN** notification messages. For additional information about callback fields, see ***Callback Fields***.

This section contains information about how your application can process these notification messages. There are several examples of source code for application-defined functions. The following list shows notification messages with sample functions that process them:

Notification message	Sample function
DTN_FORMAT	**DoFormat**
DTN_FORMATQUERY	**DoFormatQuery**
DTN_WMKEYDOWN	**DoWMKeydown**

The DoFormatQuery Application-Defined Function

A date and time picker control sends a **DTN_FORMATQUERY** notification message to request information about the maximum possible size of a callback field within the control. Handling this message ensures that all fields are displayed properly. The following **DoFormatQuery** application-defined function processes the **DTN_FORMATQUERY** notification message by calculating the width of the widest possible string for a given callback field:

```
//
//  DoFormatQuery processes DTN_FORMATQUERY messages to
//  ensure that the DTP control displays callback fields
//  properly.
//
void WINAPI DoFormatQuery(
 HWND hwndDP,
 LPNMDATETIMEFORMATQUERY lpDTFQuery)
{
    HDC hdc;
    HFONT hFont, hOrigFont;

    //  Prepare the device context for GetTextExtentPoint32call.
    hdc = GetDC(hwndDP);

    hFont = FORWARD_WM_GETFONT(hwndDP, SendMessage);

    if(!hFont)
        hFont = (HFONT)GetStockObject(DEFAULT_GUI_FONT);
```

```
    hOrigFont = SelectObject(hdc, hFont);

    // Check to see if this is the callback segment
    // desired. If so, use the longest text segment to
    // determine the maximum width of the callback field,
    // and then place the information into the
    // NMDATETIMEFORMATQUERY structure.
    if(!lstrcmp("XX",lpDTFQuery->pszFormat))
    GetTextExtentPoint32 (hdc,
        "366",  // widest date string
        3,
        &lpDTFQuery->szMax);

    // Reset the font in the device context; then release
    // the context.
    SelectObject(hdc,hOrigFont);
    ReleaseDC(hwndDP, hdc);
}
```

The DoFormat Application-Defined Function

A date and time picker control sends the **DTN_FORMAT** notification to request text that will be displayed in a callback field of the control. By handling this notification message, you allow the DTP control to display information that it does not natively support.

The following **DoFormat** application-defined function processes **DTN_FORMAT** notification messages by providing a text string for a callback field. **DoFormat** calls the **GetDayNum** application-defined function to request the day number to be used in the callback string.

```
//  DoFormat processes DTN_FORMAT to provide the text for a
//  callback field in a DTP control. In this example, the
//  callback field contains a value for the day of year.
//  The function calls the application-defined function
//  GetDayNum (below) to retrieve the correct value.
//
void WINAPI DoFormat(HWND hwndDP, LPNMDATETIMEFORMATlpDTFormat)
{
    wsprintf(lpDTFormat->szDisplay,"%d",GetDayNum(&lpDTFormat->st));
}
```

The GetDayNum Application-Defined Function

The application-defined sample function **DoFormat** calls the following **GetDayNum** application-defined function to request the day number based on the current date. **GetDayNum** returns an INT value that represents the current day of the year, from 0 to 366. This function calls another application-defined function, **IsLeapYr**, during processing.

```
//
//  GetDayNum is an application-defined function that
//  retrieves the correct day of the year value based on
//  the contents of a given SYSTEMTIME structure. This
//  function calls the IsLeapYr function to check if the
//  current year is a leap year. The function returns an
//  integer value that represents the day of year.
//
int WINAPI GetDayNum(SYSTEMTIME *st)
{
    int iNormYearAccum[] = {31,59,90,120,151,181,
                            212,243,273,304,334,365};

    int iLeapYearAccum[] = {31,60,91,121,152,182,
                            213,244,274,305,335,366};
    int iDayNum;

    if(IsLeapYr(st->wYear))
        iDayNum=iLeapYearAccum[st->wMonth-2]+st->wDay;
    else
        iDayNum=iNormYearAccum[st->wMonth-2]+st->wDay;

    return (iDayNum);
}
```

The IsLeapYr Application-Defined Function

The application-defined sample function **GetDayNum** calls the **IsLeapYr** function to determine whether the current year is a leap year. **IsLeapYr** returns a BOOL value that is TRUE if it is a leap year and FALSE otherwise.

```
//
//  IsLeapYr determines if a given year value is a leap
//  year. The function returns TRUE if the current year is
//  a leap year, and FALSE otherwise.
//
BOOL WINAPI IsLeapYr(int iYear)
{
    int iQuotient;
    BOOL fLeapYr = FALSE;

    // If the year is evenly divisible by 4 and not by 100,
    // then this is a leap year.
    if(!(iYear%4) && (iYear%100))
        fLeapYr = TRUE;
    else{
        // If the year is evenly divisible by 4 and 100,
```

```
        // then check to see if the quotient of the year
        // divided by 100 is also evenly divisible by 4.
        // If it is, then this is a leap year.
        if(!(iYear%4) && !(iYear%100)){
            iQuotient = iYear/100;
            if(!(iQuotient%4)) fLeapYr = TRUE;
        }
    }
    return (fLeapYr);
}
```

The DoWMKeydown Application-Defined Function

Date and time picker controls send the **DTN_WMKEYDOWN** message to report that the user has typed input in a callback field. Handling this message allows you to emulate the same keyboard responses supported for standard DTP fields or provide custom responses. The following **DoWMKeydown** application-defined function provides an example of how **DTN_WMKEYDOWN** notifications can be handled:

```
//
//  DoWMKeydown increments or decrements the day of month
//  according to user keyboard input.
//
void WINAPI DoWMKeydown(
 HWND hwndDP,
 LPNMDATETIMEWMKEYDOWN lpDTKeystroke)
{
    int delta =1;

    if(!lstrcmp(lpDTKeystroke->pszFormat,"XX")){
        switch(lpDTKeystroke->nVirtKey){
            case VK_DOWN:
            case VK_SUBTRACT:
                delta = -1;  // fall through

            case VK_UP:
            case VK_ADD:
                lpDTKeystroke->st.wDay += delta;
                break;
        }
    }
}
```

Date and Time Picker Control Styles

The window styles listed here are specific to date and time picker controls. The DTS_XXXFORMAT styles that define the display format cannot be combined. If none of

the format styles are suitable, use a **DTM_SETFORMAT** message to define a custom format.

DTS_APPCANPARSE	Allows the owner to parse user input and take necessary action. It enables users to edit within the client area of the control when they press the F2 key. The control sends **DTN_USERSTRING** notification messages when users are finished.
DTS_LONGDATEFORMAT	Displays the date in long format. The default format string for this style is defined by LOCALE_SLONGDATEFORMAT, which produces output like "Friday, April 19, 1996".
DTS_RIGHTALIGN	The drop-down month calendar will be right-aligned with the control instead of left-aligned, which is the default.
DTS_SHORTDATEFORMAT	Displays the date in short format. The default format string for this style is defined by LOCALE_SSHORTDATE, which produces output like "4/19/96".
DTS_SHORTDATECENTURYFORMAT	**Version 5.80.** Similar to the DTS_SHORTDATEFORMAT style, except the year is a four-digit field. The default format string for this style is based on LOCALE_SSHORTDATE. The output looks like: "4/19/1996".
DTS_TIMEFORMAT	Displays the time. The default format string for this style is defined by LOCALE_STIMEFORMAT, which produces output like "5:31:42 PM".
DTS_SHOWNONE	It is possible to have no date currently selected in the control. With this style, the control displays a check box that users can check once they have entered or selected a date. Until this check box is checked, the application will not be able to retrieve the date from the control because, in essence, the control has no date. This state can be set with the **DTM_SETSYSTEMTIME** message or queried with the **DTM_GETSYSTEMTIME** message.
DTS_UPDOWN	Places an up-down control to the right of the DTP control to modify date-time values. This style can be used in place of the drop-down month calendar, which is the default style.

Date and Time Picker Reference

Date and Time Picker Control Messages

DTM_GETMCCOLOR

Retrieves the color for a given portion of the month calendar within a date and time picker control. You can send this message explicitly or use the **DateTime_GetMonthCalColor** macro.

```
DTM_GETMCCOLOR
    wParam = (WPARAM)(INT)iColor;
    lParam = 0;
```

Parameters

iColor

INT value specifying which month calendar color to retrieve. This value can be one of the following:

MCSC_BACKGROUND	Retrieve the background color displayed between months.
MCSC_MONTHBK	Retrieve the background color displayed within the month.
MCSC_TEXT	Retrieve the color used to display text within a month.
MCSC_TITLEBK	Retrieve the background color displayed in the calendar's title.
MCSC_TITLETEXT	Retrieve the color used to display text within the calendar's title.
MCSC_TRAILINGTEXT	Retrieve the color used to display header day and trailing day text. Header and trailing days are the days from the previous and following months that appear on the current month calendar.

Return Values

Returns a **COLORREF** value that represents the color setting for the specified portion of the month calendar control if successful. The message returns –1 if unsuccessful.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).

Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DTM_GETMCFONT

Retrieves the font that the date and time picker control's child month calendar control is currently using. You can send this message explicitly or use the **DateTime_GetMonthCalFont** macro.

```
DTM_GETMCFONT
    wParam = 0;
    lParam = 0;
```

Return Values

Returns an HFONT value that is the handle to the current font.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DTM_GETMONTHCAL

Retrieves the handle to a date and time picker's child month calendar control. You can send this message explicitly or use the **DateTime_GetMonthCal** macro.

```
DTM_GETMONTHCAL
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the handle to a DTP control's child month calendar control if successful, or NULL otherwise.

Remarks

DTP controls create a child month calendar control when the user clicks the drop-down arrow. When the month calendar is no longer needed, it is destroyed. So your application must not rely on a static handle to the DTP control's child month calendar.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

See Also

DTN_CLOSEUP, **DTN_DROPDOWN**

DTM_GETRANGE

Retrieves the current minimum and maximum allowable system times for a date and time picker control. You can send this message explicitly or use the **DateTime_GetRange** macro.

```
DTM_GETRANGE
    wParam = 0;
    lParam = (LPARAM) lpSysTimeArray;
```

Parameters

lpSysTimeArray
Address of a two-element array of **SYSTEMTIME** structures.

Return Values

Returns a DWORD value that is a combination of GDTR_MIN or GDTR_MAX. The first element of the **SYSTEMTIME** array contains the minimum allowable time if GDTR_MIN is set. The second element of the **SYSTEMTIME** array contains the maximum allowable time if GDTR_MAX is set.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DTM_GETSYSTEMTIME

Retrieves the currently selected time from a date and time picker control and places it in a specified **SYSTEMTIME** structure. You can send this message explicitly or use the **DateTime_GetSystemtime** macro.

```
DTM_GETSYSTEMTIME
    wParam = 0;
    lParam = (LPARAM) lpSysTime;
```

Parameters

lpSysTime

Pointer to a **SYSTEMTIME** structure. If **DTM_GETSYSTEMTIME** returns GDT_VALID, this structure will contain the system time. Otherwise, it will not contain valid information. This parameter must be a valid pointer; it cannot be NULL.

Return Values

Returns GDT_VALID if the time information was successfully placed in *lpSysTime*. Returns GDT_NONE if the control was set to the **DTS_SHOWNONE** style and the control check box was not selected. Returns GDT_ERROR if an error occurs.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DTM_SETFORMAT

Sets the display of a date and time picker control based on a given format string. You can send this message explicitly or use the **DateTime_SetFormat** macro.

```
DTM_SETFORMAT
    wParam = 0;
    lParam = (LPARAM) lpszFormat;
```

Parameters

lpszFormat

Address of a zero-terminated **format string** that defines the desired display. Setting this parameter to NULL will reset the control to the default format string for the current style.

Return Values

Returns nonzero if successful, or zero otherwise.

Remarks

It is acceptable to include extra characters within the format string to produce a more rich display. However, any nonformat characters must be enclosed within single quotation marks. For example, the format string "'Today is: 'hh':'m':'s ddddMMMdd', 'yyy" would produce output like "Today is: 04:22:31 Tuesday Mar 23, 1996".

Note A DTP control tracks locale changes when it is using the default format string. If you set a custom format string, it will not be updated in response to locale changes.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DTM_SETMCCOLOR

Sets the color for a given portion of the month calendar within a date and time picker control. You can send this message explicitly or use the **DateTime_SetMonthCalColor** macro.

```
DTM_SETMCCOLOR
    wParam = (WPARAM)(INT) iColor;
    lParam = (LPARAM)(COLORREF) clr;
```

Parameters

iColor

INT value specifying which month calendar color to set. This value can be one of the following:

MCSC_BACKGROUND	Set the background color displayed between months.
MCSC_MONTHBK	Set the background color displayed within the month.
MCSC_TEXT	Set the color used to display text within a month.
MCSC_TITLEBK	Set the background color displayed in the calendar's title.

(continued)

(continued)

MCSC_TITLETEXT	Set the color used to display text within the calendar's title.
MCSC_TRAILINGTEXT	Set the color used to display header day and trailing day text. Header and trailing days are the days from the previous and following months that appear on the current month calendar.

clr
COLORREF value representing the color that will be set for the specified area of the month calendar.

Return Values

Returns a **COLORREF** value that represents the previous color setting for the specified portion of the month calendar control if successful. Otherwise, the message returns –1.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DTM_SETMCFONT

Sets the font to be used by the date and time picker control's child month calendar control. You can send this message explicitly or use the **DateTime_SetMonthCalFont** macro.

```
DTM_SETMCFONT
    wParam = (WPARAM)(HFONT) hFont;
    lParam = (LPARAM) MAKELONG(fRedraw, 0);
```

Parameters

hFont
Handle to the font that will be set.

fRedraw
Specifies whether the control should be redrawn immediately upon setting the font. Setting this parameter to TRUE causes the control to redraw itself.

Return Values

The return value for this message is not used.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DTM_SETRANGE

Sets the minimum and maximum allowable system times for a date and time picker control. You can send this message explicitly or use the **DateTime_SetRange** macro.

```
DTM_SETRANGE
    wParam = (WPARAM) flags;
    lParam = (LPARAM) lpSysTimeArray;
```

Parameters

flags

Value specifying which range values are valid. This parameter can be a combination of the following values:

GDTR_MAX	The second element in the **SYSTEMTIME** structure array is valid and will be used to set the maximum allowable system time.
GDTR_MIN	The first element in the **SYSTEMTIME** structure array is valid and will be used to set the minimum allowable system time.

lpSysTimeArray

Address of a two-element array of **SYSTEMTIME** structures. The first element of the **SYSTEMTIME** array contains the minimum allowable time. The second element of the **SYSTEMTIME** array contains the maximum allowable time. It is not necessary to fill an array element that is not specified in the *flags* parameter.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DTM_SETSYSTEMTIME

Sets the time in a date and time picker control. You can send this message explicitly or use the **DateTime_SetSystemtime** macro.

```
DTM_SETSYSTEMTIME
    wParam = (WPARAM) flag;
    lParam = (LPARAM) lpSysTime;
```

Parameters

flag
Value specifying the action that should be performed. This value must be set to one of the following:

GDT_NONE	Set the DTP control to "no date" and clear its check box. When this flag is specified, *lpSysTime* is ignored. This flag applies only to DTP controls that are set to the **DTS_SHOWNONE** style.
GDT_VALID	Set the DTP control according to the data within the structure that *lpSysTime* points to.

lpSysTime
Address of a **SYSTEMTIME** structure containing the system time used to set the DTP control.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

Date and Time Picker Control Macros

DateTime_GetMonthCal

Retrieves the handle to a date and time picker's child month calendar control. You can use this macro or send the **DTM_GETMONTHCAL** message explicitly.

```
HWND DateTime_GetMonthCal(
    HWND hwndDP);
```

Parameters

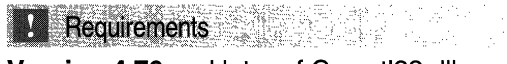
hwndDP
Handle to a DTP control.

Return Values

Returns the handle to a DTP control's child month calendar control.

Remarks

DTP controls create a child month calendar control when the user clicks the drop-down arrow. When the month calendar is no longer needed, it is destroyed. So your application must not rely on a static handle to the DTP's child month calendar.



Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DTN_CLOSEUP, **DTN_DROPDOWN**

DateTime_GetMonthCalColor

Retrieves the color for a given portion of the month calendar within a date and time picker control. You can use this macro or send the **DTM_GETMCCOLOR** message explicitly.

```
COLORREF DateTime_GetMonthCalColor(
    HWND hwndDP,
    int iColor);
```

Parameters

hwndDP

Handle to a DTP control.

iColor

INT value specifying which month calendar color to retrieve. This value can be one of the following:

MCSC_BACKGROUND	Retrieve the background color displayed between months.
MCSC_MONTHBK	Retrieve the background color displayed within the month.
MCSC_TEXT	Retrieve the color used to display text within a month.
MCSC_TITLEBK	Retrieve the background color displayed in the calendar's title.
MCSC_TITLETEXT	Retrieve the color used to display text within the calendar's title.
MCSC_TRAILINGTEXT	Retrieve the color used to display header day and trailing day text. Header and trailing days are the days from the previous and following months that appear on the current month calendar.

Return Values

Returns a **COLORREF** value that represents the color setting for the specified portion of the month calendar control if successful. Otherwise, the macro returns –1.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DateTime_GetMonthCalFont

Retrieves the font that the date and time picker control's child month calendar control is currently using. You can use this macro or send the **DTM_GETMCFONT** message explicitly.

```
HFONT DateTime_GetMonthCalFont(
    HWND hwndDP);
```

Parameters

hwndDP
Handle to a DTP control.

Return Values

Returns an HFONT value that is the handle to the current font.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DateTime_GetRange

Retrieves the current minimum and maximum allowable system times for a date and time picker control. You can use this macro, or send the **DTM_GETRANGE** message explicitly.

```
DWORD DateTime_GetRange(
    HWND hwndDT,
    LPSYSTEMTIME lpSysTimeArray);
```

Parameters

hwndDT
Address of a DTP control.

lpSysTimeArray
Address of a two-element array of **SYSTEMTIME** structures.

Return Values

Returns a DWORD value that is a combination of GDTR_MIN or GDTR_MAX. The first element of the **SYSTEMTIME** array contains the minimum allowable time. The second element of the **SYSTEMTIME** array contains the maximum allowable time.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DateTime_GetSystemtime

Retrieves the currently selected time from a date and time picker control and places it in a specified **SYSTEMTIME** structure. You can use this macro, or send the **DTM_GETSYSTEMTIME** message explicitly.

```
DWORD DateTime_GetSystemtime(
    HWND hwndDP,
    LPSYSTEMTIME lpSysTime);
```

Parameters

hwndDP
Handle to a DTP control.

lpSysTime
Pointer to a **SYSTEMTIME** structure. If **DTM_GETSYSTEMTIME** returns GDT_VALID, this structure will contain the system time. Otherwise, it will not contain valid information. This parameter must be a valid pointer; it cannot be NULL.

Return Values

Returns GDT_VALID if the time information was successfully placed in *lpSysTime*. Returns GDT_NONE if the control was set to the **DTS_SHOWNONE** style and the control check box was not selected. Returns GDT_ERROR if an error occurs.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DateTime_SetFormat

Sets the display of a date and time picker control based on a given format string. You can use this macro or send the **DTM_SETFORMAT** message explicitly.

```
void DateTime_SetFormat(
    HWND hwndDT,
    LPCTSTR lpszFormat);
```

Parameters

hwndDT
Handle to a DTP control.

lpszFormat
Address of a zero-terminated **format string** that defines the desired display. Setting this parameter to NULL will reset the control to the default format string for the current style.

Return Values

Returns nonzero if successful, or zero otherwise.

Remarks

It is acceptable to include extra characters within the format string to produce a more rich display. However, any nonformat characters must be enclosed within single quotation marks. For example, the format string "'Today is: 'hh':'m':'s ddddMMMdd', 'yyy" would produce output like "Today is: 04:22:31 Tuesday Mar 23, 1996".

Note A DTP control tracks locale changes when it is using the default format string. If you set a custom format string, it will not be updated in response to locale changes.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later)
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DateTime_SetMonthCalColor

Sets the color for a given portion of the month calendar within a date and time picker control. You can use this macro or send the **DTM_SETMCCOLOR** message explicitly.

```
COLORREF DateTime_SetMonthCalColor(
    HWND     hwndDP,
    int      iColor,
    COLORREF clr);
```

Parameters

hwndDP

Handle to a DTP control.

iColor

INT value specifying which month calendar color to set. This value can be one of the following:

Value	Meaning
MCSC_BACKGROUND	Set the background color displayed between months.
MCSC_MONTHBK	Set the background color displayed within the month.
MCSC_TEXT	Set the color used to display text within a month.
MCSC_TITLEBK	Set the background color displayed in the calendar's title.
MCSC_TITLETEXT	Set the color used to display text within the calendar's title.
MCSC_TRAILINGTEXT	Set the color used to display header day and trailing day text. Header and trailing days are the days from the previous and following months that appear on the current month calendar.

clr

COLORREF value that represents the color that will be set for the specified area of the month calendar.

Return Values

Returns a **COLORREF** value that represents the previous color setting for the specified portion of the month calendar control if successful. Otherwise, this message returns –1.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).

Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DateTime_SetMonthCalFont

Sets the font to be used by the date and time picker control's child month calendar control. You can use this macro or explicitly send the **DTM_SETMCFONT** message.

```
void DateTime_SetMonthCalFont(
    HWND  hwndDP,
    HFONT  hFont,
    LPARAM MAKELONG(fRedraw,0));
```

Parameters

hwndDP
Handle to a DTP control.

hFont
Handle to the font that will be set.

fRedraw
Specifies whether the control should be redrawn immediately upon setting the font. Setting this parameter to TRUE causes the control to redraw itself.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DateTime_SetRange

Sets the minimum and maximum allowable system times for a date and time picker control. You can use this macro or send the **DTM_SETRANGE** message explicitly.

```
BOOL DateTime_SetRange(
    HWND hwndDT,
    DWORD flags,
    LPSYSTEMTIME lpSysTimeArray);
```

Parameters

hwndDT
Handle to a DTP control.

flags
Value that specifies which range values are valid. This value can be a combination of the following:

GDTR_MAX	The second element in the **SYSTEMTIME** structure array is valid and will be used to set the maximum allowable system time.
GDTR_MIN	The first element in the **SYSTEMTIME** structure array is valid and will be used to set the minimum allowable system time.

lpSysTimeArray
Address of a two-element array of **SYSTEMTIME** structures. The first element of the **SYSTEMTIME** array contains the minimum allowable time. The second element of the **SYSTEMTIME** array contains the maximum allowable time. It is not necessary to fill an array element that is not specified in the *flags* parameter.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DateTime_SetSystemtime

Sets a date and time picker control to a given date and time. You can use this macro or send the **DTM_SETSYSTEMTIME** message explicitly.

```
BOOL DateTime_SetSystemtime(
    HWND hwndDT,
    DWORD flag,
    LPSYSTEMTIME lpSysTime);
```

Parameters

hwndDT
Handle to a DTP control.

flag
Value that specifies the action that should be performed. This should be set to one of the following values:

GDT_NONE	Set the DTP control to "no date" and clear its check box. When this flag is specified, *lpSysTime* is ignored. This flag applies only to DTP controls that are set to the **DTS_SHOWNONE** style.
GDT_VALID	Set the DTP control according to the data within the **SYSTEMTIME** structure pointed to by *lpSysTime*.

lpSysTime
Address of a **SYSTEMTIME** structure that contains the system time information by which to set the DTP control.

Return Values
Returns nonzero if successful, or zero otherwise.

Requirements
Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

Date and Time Picker Control Notification Messages

DTN_CLOSEUP

Sent by a date and time picker control when the user closes the drop-down month calendar. The month calendar is closed when the user chooses a date from the month calendar or clicks the drop-down arrow while the calendar is open.

```
DTN_CLOSEUP
    lpNmhdr = (LPMNHDR)lParam;
```

Parameters
lpNmhdr
Address of an **NMHDR** structure that contains information about the notification message.

Return Values

The return value for this notification is not used.

Remarks

DTP controls do not maintain a static child month calendar control. The DTP control destroys the child month calendar control prior to sending this notification. So your application must not rely on a static window handle for the control's child month calendar.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.
Import Library: comctl32.lib.

See Also

DTN_DROPDOWN, **DTM_GETMONTHCAL**

DTN_DATETIMECHANGE

Sent by a date and time picker control whenever a change occurs. This notification message is sent in the form of a **WM_NOTIFY** message.

```
DTN_DATETIMECHANGE
    lpChange = (LPNMDATETIMECHANGE) lParam;
```

Parameters

lpChange
Address of an **NMDATETIMECHANGE** structure containing information about the change that took place in the control.

Return Values

The owner of the control must return zero.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DTN_DROPDOWN

Sent by a date and time picker control when the user activates the drop-down month calendar.

```
DTN_DROPDOWN
    lpNmhdr = (LPNMHDR)lParam;
```

Parameters

lpNmhdr
Address of an **NMHDR** structure that contains information about the notification message.

Return Values

The return value for this notification is not used.

Remarks

One task that your notification handler may need to perform is to customize the dropdown month-calendar control. For instance, if you do not want "Go To Today", you need to set the control's **MCS_NOTODAY** style. To get a handle to the month-calendar control, send the DTP control a **DTM_GETMONTHCAL** message. You can then use this handle and **SetWindowLong** to set the desired month-calendar style.

DTP controls do not maintain a static child month calendar control. The DTP control creates a new month calendar control before sending this notification message. Additionally, the DTP control destroys the child control when it is not active (visible). So your application must not rely on a static window handle for the control's child month calendar.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

See Also

DTN_CLOSEUP, DTM_GETMONTHCAL

DTN_FORMAT

Sent by a date and time picker control to request text to be displayed in a callback field. This notification message is sent in the form of a **WM_NOTIFY** message.

```
DTN_FORMAT
    lpNMFormat = (LPNMDATETIMEFORMAT) lParam;
```

Parameters

lpNMFormat

Address of an **NMDATETIMEFORMAT** structure containing information regarding this instance of the notification message. The structure contains the substring that defines the callback field and receives the formatted string that the control will display.

Return Values

The owner of the control must return zero.

Remarks

Handling this message allows the owner of the control to provide a custom string that the control will display. (For additional information about callback fields, see *Callback Fields*.)

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DTN_FORMATQUERY

Sent by a date and time picker control to retrieve the maximum allowable size of the string that will be displayed in a callback field. This notification message is sent in the form of a **WM_NOTIFY** message.

```
DTN_FORMATQUERY
    lpDTFormatQuery = (LPNMDATETIMEFORMATQUERY) lParam;
```

Parameters

lpDTFormatQuery

Address of an **NMDATETIMEFORMATQUERY** structure containing information about the callback field. The structure contains a substring that defines a callback field and receives the maximum allowable size of the string that will be displayed in the callback field.

Return Values

The owner of the control must calculate the maximum possible width of the text that will be displayed in the callback field, set the **szMax** member of the **NMDATETIMEFORMATQUERY** structure, and return zero.

Remarks

Handling this message prepares the control to adjust for the maximum size of the string that will be displayed in a particular callback field. This enables the control to properly display output at all times, reducing flicker within the control's display. (For additional information about callback fields, see *Callback Fields*.)

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DTN_USERSTRING

Sent by a date and time picker control when a user finishes editing a string in the control. This notification message is only sent by DTP controls that are set to the **DTS_APPCANPARSE** style. This message is sent in the form of a **WM_NOTIFY** message.

```
DTN_USERSTRING
    lpDTstring = (LPNMDATETIMESTRING) lParam;
```

Parameters

lpDTstring

Address of an **NMDATETIMESTRING** structure that contains information about the instance of the notification message.

Return Values

The owner of the control must return zero.

Remarks

Handling this message allows the owner to provide custom responses to strings entered into the control by the user. The owner must be prepared to parse the input string and take action if necessary.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

DTN_WMKEYDOWN

Sent by a date and time picker control when the user types in a callback field. This message is sent in the form of a **WM_NOTIFY** message.

```
DTN_WMKEYDOWN
    lpDTKeystroke = (LPNMDATETIMEWMKEYDOWN)lParam;
```

Parameters

lpDTKeystroke

Address of an **NMDATETIMEWMKEYDOWN** structure containing information about this instance of the notification. The structure includes information about the key that the user typed, the substring that defines the callback field, and the current system date and time.

Return Values

The owner of the control must return zero.

Remarks

Handling this message allows the owner of the control to provide specific responses to keystrokes within the callback fields of the control.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

NM_KILLFOCUS (date time)

Notifies a date and time picker control's parent window that the control has lost the input focus. NM_KILLFOCUS is sent in the form of a **WM_NOTIFY** message.

```
NM_KILLFOCUS
    lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh
Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The return value is ignored.

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 commctrl.h.

NM_SETFOCUS (date time)

Notifies a date and time picker control's parent window that the control has received the input focus. NM_SETFOCUS is sent in the form of a **WM_NOTIFY** message.

```
NM_SETFOCUS
    lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh

Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The return value is ignored.

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 commctrl.h.

Date and Time Picker Control Structures

NMDATETIMECHANGE

Contains information about a change that has taken place in a date and time picker control. This structure is used with the **DTN_DATETIMECHANGE** notification message.

```
typedef struct tagNMDATETIMECHANGE {
    NMHDR      nmhdr;
    DWORD      dwFlags;
    SYSTEMTIME st;
} NMDATETIMECHANGE, FAR * LPNMDATETIMECHANGE;
```

Members

nmhdr

NMHDR structure that contains information about the notification message.

dwFlags

Value that indicates if the control was set to "no date" status (for DTS_SHOWNONE only). This flag also specifies whether the contents of the **st** member are valid and contain current time information. This value can be one of the following:

GDT_NONE	The control is set to "no date" status. The "no date" status applies only to controls that are set to the **DTS_SHOWNONE** style.
GDT_VALID	The control is not set to the "no date" status. The **st** member contains the current date and time.

st
SYSTEMTIME structure that contains information about the current system date and time.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

NMDATETIMEFORMAT

Contains information about a portion of the format string that defines a callback field within a date and time picker control. It carries the substring that defines the callback field and contains a buffer to receive the string that will be displayed in the callback field. This structure is used with the **DTN_FORMAT** notification message.

```
typedef struct tagNMDATETIMEFORMAT{
    NMHDR nmhdr;
    LPCTSTR pszFormat;
    SYSTEMTIME st;
    LPCTSTR pszDisplay;
    TCHAR szDisplay[64];
} NMDATETIMEFORMAT, FAR * LPNMDATETIMEFORMAT;
```

Members

nmhdr
NMHDR structure that contains information about the notification message.

pszFormat
Address of the substring that defines a DTP control callback field. The substring comprises one or more "X" characters followed by a NULL character. (For more information about callback fields, see *Callback Fields*.)

st
SYSTEMTIME structure that contains the date and time to be formatted.

pszDisplay
Address of a null-terminated string that contains the display text of the control. By default, this is the address of the **szDisplay** member of this structure.

It is acceptable to have **pszDisplay** point to an existing string. In this case, you don't need to assign a value to **szDisplay**. However, the string that **pszDisplay** points to

must remain valid until another DTN_FORMAT notification is sent, or until the control is destroyed.

szDisplay
64-character buffer that is to receive the zero-terminated string that the DTP control will display. It is not necessary to fill the entire buffer.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

NMDATETIMEFORMATQUERY

Contains information about a date and time picker control callback field. It contains a substring (taken from the control's format string) that defines a callback field. The structure receives the maximum allowable size of the text that will be displayed in the callback field. This structure is used with the **DTN_FORMATQUERY** notification message.

```
typedef struct tagNMDATETIMEFORMATQUERY{
    NMHDR nmhdr;
    LPCTSTR pszFormat;
    SIZE szMax;
} NMDATETIMEFORMATQUERY, FAR * LPNMDATETIMEFORMATQUERY;
```

Members

nmhdr
NMHDR structure that contains information about this notification message.

pszFormat
Address of a substring that defines a DTP control callback field. The substring is one or more "X" characters followed by a NULL. (For additional information about callback fields, see *Callback Fields*.)

szMax
SIZE structure that must be filled with the maximum size of the text that will be displayed in the callback field.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

NMDATETIMESTRING

Contains information specific to an edit operation that has taken place in a date and time picker control. This message is used with the **DTN_USERSTRING** notification message.

```
typedef struct tagNMDATETIMESTRING{
    NMHDR      nmhdr;
    LPCTSTR    pszUserString;
    SYSTEMTIME st;
    DWORD      dwFlags;
} NMDATETIMESTRING, FAR * LPNMDATETIMESTRING;
```

Members

nmhdr
NMHDR structure that contains information about this notification message.

pszUserString
Address of the zero-terminated string that the user entered.

st
SYSTEMTIME structure that must be filled in by the owner when handling the **DTN_USERSTRING** notification message.

dwFlags
Return field. Set this member to GDT_VALID to indicate that the **st** member is valid or to GDT_NONE to set the control to "no date" status (**DTS_SHOWNONE** style only).

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

NMDATETIMEWMKEYDOWN

Carries information used to describe and handle a **DTN_WMKEYDOWN** notification message.

```
typedef struct tagNMDATETIMEWMKEYDOWN{
    NMHDR     nmhdr;
    int       nVirtKey;
    LPCTSTR   pszFormat;
    SYSTEMTIME st;
} NMDATETIMEWMKEYDOWN, FAR * LPNMDATETIMEWMKEYDOWN;
```

Members

nmhdr
NMHDR structure that contains information about the notification message.

nVirtKey
Virtual key code that represents the key that the user pressed.

pszFormat
Zero-terminated substring, taken from the format string, that defines the callback field. The substring is one or more "X" characters, followed by a NULL.

st
SYSTEMTIME structure containing the current date and time from the DTP control. The owner of the control must modify the time information based on the user's keystroke.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

CHAPTER 13

Drag List Boxes

A *drag list box* is a special type of list box that enables the user to drag items from one position to another. An application can use a drag list box to display strings in a particular sequence and allow the user to change the sequence by dragging the items into position.

Using Drag List Boxes

Drag list boxes have the same window styles and process the same messages as standard list boxes. To create a drag list box, first create a standard list box and then call the **MakeDragList** function. To convert a list box in a dialog box to a drag list box, you can call **MakeDragList** when the WM_INITDIALOG message is processed.

Drag List Box Messages

A drag list box notifies the parent window of drag events by sending it a drag list message. The parent window must process the drag list message.

The drag list box registers this message when the **MakeDragList** function is called. To get the message identifier (numeric value) of the drag list message, call the **RegisterWindowMessage** function and specify the DRAGLISTMSGSTRING value.

The *wParam* parameter of the drag list message is the control identifier for the drag list box. The *lParam* parameter is the address of a **DRAGLISTINFO** structure, which contains the notification code for the drag event and other information. The return value of the message depends on the notification.

Drag List Box Notification Messages

The drag list notification message, which is identified by the **uNotification** member of the **DRAGLISTINFO** structure included with the drag list message, can be **DL_BEGINDRAG**, **DL_DRAGGING**, **DL_CANCELDRAG**, or **DL_DROPPED**.

The **DL_BEGINDRAG** notification message is sent when the cursor is on a list item and the user clicks the left mouse button. The parent window can return TRUE to begin the drag operation or FALSE to disallow dragging. In this way, the parent window can enable dragging for some list items and disable it for others. You can determine which list item is at the specified location by using the **LBItemFromPt** function.

If dragging is in effect, the **DL_DRAGGING** notification message is sent whenever the mouse is moved, or at regular intervals if the mouse is not being moved. The parent window should first determine the list item under the cursor by using **LBItemFromPt** and

then draw the insert icon by using the **DrawInsert** function. By specifying TRUE for the *bAutoScroll* parameter of **LBItemFromPt**, you can cause the list box to scroll by one line if the cursor is above or below its client area. The value you return for this notification specifies the type of mouse cursor that the drag list box should set.

The **DL_CANCELDRAG** notification message is sent if the user cancels a drag operation by clicking the right mouse button or pressing the ESC key. The **DL_DROPPED** notification message is sent if the user completes a drag operation by releasing the left mouse button, even if the cursor is not over a list item. The drag list box releases the mouse capture before sending either notification. The return value of these two notifications is ignored.

Drag List Box Reference

Drag List Box Functions

DrawInsert

Draws the insert icon in the parent window of the specified drag list box.

```
void DrawInsert(
  HWND handParent,
  HWND hLB,
  int nItem
);
```

Parameters

handParent
Handle to the parent window of the drag list box.

hLB
Handle to the drag list box.

nItem
Identifier of the icon item to be drawn.

Return Values

No return value.

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 commctrl.h.
Import Library: comctl32.lib.

LBItemFromPt

Retrieves the index of the item at the specified point in a list box.

```
int LBItemFromPt(
  HWND hLB,
  POINT pt,
  BOOL bAutoScroll
);
```

Parameters

hLB
Handle to the list box to check.

pt
POINT structure that contains the screen coordinates to check.

bAutoScroll
Scroll flag. If this parameter is TRUE and the point is directly above or below the list box, the function scrolls the list box by one line and returns –1. Otherwise, the function does not scroll the list box.

Return Values

Returns the item identifier if the point is over a list item, or –1 otherwise.

Remarks

The **LBItemFromPt** function only scrolls the list box if a minimum amount of time has passed since it last did so. Timing prevents the list box from scrolling too quickly if the function is called repeatedly in rapid succession—for example, when **DL_DRAGGING** notification messages or **WM_MOUSEMOVE** messages are processed.

If the specified point is outside the client area of the list box and *bAutoScroll* is TRUE, the function scrolls the list box instead of returning an item identifier.

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 commctrl.h.
Import Library: comctl32.lib.

MakeDragList

Changes the specified single-selection list box to a drag list box.

```
BOOL MakeDragList(
  HWND hLB
);
```

Parameters

hLB
Handle to the single-selection list box.

Return Values

Returns nonzero if successful, 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 commctrl.h.
Import Library: comctl32.lib.

Drag List Box Notifications

DL_BEGINDRAG

Notifies the drag list box's parent window that the user has clicked the left mouse button on an item. A drag list box sends DL_BEGINDRAG in the form of a drag list message.

```
DL_BEGINDRAG
    idCtl = (WPARAM)(int) wParam;
    pDragInfo = (LPARAM)(LPDRAGLISTINFO) lParam;
```

Parameters

idCtl
Control identifier of the drag list box.

pDragInfo
Address of a **DRAGLISTINFO** structure that contains the DL_BEGINDRAG notification code, the handle to the drag list box, and the cursor position.

Return Values

Return TRUE to begin the drag operation, or FALSE to prevent the drag operation.

Remarks

When processing this notification message, a window procedure typically determines the list item at the specified cursor position by using the **LBItemFromPt** function. It then returns TRUE or FALSE, depending on whether the item should be dragged. Before returning TRUE, the window procedure should save the index of the list item so the application knows which item to move or copy when the drag operation is completed.

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 commctrl.h.

DL_CANCELDRAG

Signals that the user has canceled a drag operation by clicking the right mouse button or pressing the ESC key. A drag list box sends DL_CANCELDRAG to its parent window in the form of a drag list message.

```
DL_CANCELDRAG
    idCtl = (WPARAM)(int) wParam;
    pDragInfo = (LPARAM)(LPDRAGLISTINFO) lParam;
```

Parameters

idCtl
Control identifier of the drag list box.

pDragInfo
Address of a **DRAGLISTINFO** structure that contains the DL_CANCELDRAG notification code, the handle to the drag list box, and the cursor position.

Return Values

No return value.

Remarks

By processing the DL_CANCELDRAG notification message, an application can reset its internal state to indicate that dragging is not in effect.

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 commctrl.h.

DL_DRAGGING

Signals that the user has moved the mouse while dragging an item. DL_DRAGGING is also sent periodically during dragging even if the mouse is not moved. A drag list box sends this notification to its parent window in the form of a drag list message.

```
DL_DRAGGING
    idCtl = (WPARAM)(int) wParam;
    pDragInfo = (LPARAM)(LPDRAGLISTINFO) lParam;
```

Parameters

idCtl

Control identifier of the drag list box.

pDragInfo

Address of a **DRAGLISTINFO** structure that contains the DL_DRAGGING notification code, the handle to the drag list box, and the cursor position.

Return Values

The return value determines the type of mouse cursor that the drag list should set; it can be the DL_STOPCURSOR, DL_COPYCURSOR, or DL_MOVECURSOR value. If any other value is returned, the cursor does not change.

Remarks

A window procedure typically processes the DL_DRAGGING notification message by determining the item under the cursor and then drawing an insert icon. To get the item under the cursor, use the **LBItemFromPt** function, specifying TRUE for the *bAutoScroll* parameter. This option causes the drag list box to scroll periodically if the cursor is above or below its client area. To draw the insert icon, use the **DrawInsert** function.

! 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 commctrl.h.

DL_DROPPED

Signals that the user has completed a drag operation by releasing the left mouse button. A drag list box sends DL_DROPPED to its parent window in the form of a drag list message.

```
DL_BEGINDRAG
    idCtl = (WPARAM)(int) wParam;
    pDragInfo = (LPARAM)(LPDRAGLISTINFO) lParam;
```

Parameters

idCtl
Control identifier of the drag list box.

pDragInfo
Address of a **DRAGLISTINFO** structure that contains the DL_DROPPED notification code, the handle to the drag list box, and the cursor position.

Return Values

No return value.

Remarks

This notification is normally processed by inserting the item being dragged into the list in front of the item under the cursor. To retrieve the index of the item at the cursor position, use the **LBItemFromPt** function. Note that the DL_DROPPED notification message is sent even if the cursor is not on a list item. In that case, **LBItemFromPt** returns –1.

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 commctrl.h.

Drag List Box Structures

DRAGLISTINFO

Contains information about a drag event. The pointer to **DRAGLISTINFO** is passed as the *lParam* parameter of the drag list message.

```
Typedef struct {
    UINT uNotification;
    HWND hWnd;
    POINT ptCursor;
} DRAGLISTINFO, FAR *LPDRAGLISTINFO;
```

Members

uNotification

Notification code that specifies the type of drag event. This member can be one of the following values:

DL_BEGINDRAG	The user has clicked the left mouse button on a list item.
DL_CANCELDRAG	The user has canceled the drag operation by clicking the right mouse button or pressing the ESC key.
DL_DRAGGING	The user has moved the mouse while dragging an item.
DL_DROPPED	The user has released the left mouse button, completing a drag operation.

hWnd

Handle to the drag list box.

ptCursor

POINT structure that contains the current x- and y-coordinates of the mouse cursor.

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 commctrl.h.

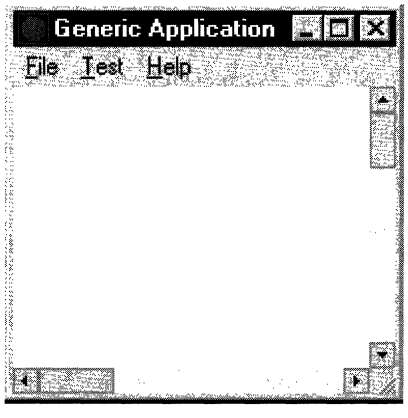
CHAPTER 14

Flat Scroll Bars

Flat Scroll Bars

Microsoft Internet Explorer Version 4.0 introduces a new visual technology called *flat scroll bars*. Functionally, flat scroll bars behave just like normal scroll bars. The only difference is they are not displayed three-dimensionally.

The following illustration shows a window that contains flat scroll bars.



Note Flat scroll bar APIs are implemented in version 4.71 and later of Comctl32.dll.

Using Flat Scroll Bars

This section describes how to implement flat scroll bars in your application.

Before You Begin

To use the flat scroll bar APIs, you must include Commctrl.h in your source files and link with Comctl32.lib.

Adding Flat Scroll Bars to a Window

To add flat scroll bars to a window, call **InitializeFlatSB**, passing the handle to the window. Instead of using the standard scroll bar APIs to manipulate your scroll bars, you must use the **FlatSB_xxxx** versions. There are flat scroll bar APIs for setting and retrieving the scroll information, range, and position. If flat scroll bars haven't been initialized for your window, the flat scroll bar APIs will defer to the corresponding

standard APIs, if any exist. This allows you to turn flat scroll bars on and off without having to write conditional code.

Because an application may have set custom metrics for its flat scroll bars, they are not automatically updated when system metrics change. When the system scroll bar metrics change, a **WM_SETTINGCHANGE** message is broadcast, with its *wParam* set to SPI_SETNONCLIENTMETRICS. To update flat scroll bars to the new system metrics, applications must handle this message, and change the flat scroll bar's metric-dependent properties explicitly.

To update your scroll bar properties, use **FlatSB_SetScrollProp**. The following code fragment changes a flat scroll bar's metric dependent properties to the current system values.

```
void FlatSB_UpdateMetrics(HWND hWnd)
{
FlatSB_SetScrollProp(hWnd, WSB_PROP_CXVSCROLL,
GetSystemMetrics(SM_CXVSCROLL), FALSE);
FlatSB_SetScrollProp(hWnd, WSB_PROP_CXHSCROLL,
GetSystemMetrics(SM_CXHSCROLL), FALSE);
FlatSB_SetScrollProp(hWnd, WSB_PROP_CYVSCROLL,
GetSystemMetrics(SM_CYVSCROLL), FALSE);
FlatSB_SetScrollProp(hWnd, WSB_PROP_CYHSCROLL,
GetSystemMetrics(SM_CYHSCROLL), FALSE);
FlatSB_SetScrollProp(hWnd, WSB_PROP_CXHTHUMB,
GetSystemMetrics(SM_CXHTHUMB), FALSE);
FlatSB_SetScrollProp(hWnd, WSB_PROP_CYVTHUMB,
GetSystemMetrics(SM_CYVTHUMB), TRUE);
}
```

Enhancing Flat Scroll Bars

FlatSB_SetScrollProp allows you to modify the flat scroll bars to customize the look of your window. You can set the width of a vertical scroll bar and the height of a horizontal scroll bar. You can also set the width (horizontal scroll bar) and the height (vertical scroll bar) of the scroll bar's direction arrows.

FlatSB_SetScrollProp also allows you to customize how the flat scroll bars are displayed. By changing the WSB_PROP_VSTYLE and WSB_PROP_HSTYLE properties, you can set the type of scroll bar that you wish to use. Three styles are available:

FSB_ENCARTA_MODE	A standard flat scroll bar is displayed. When the mouse moves over a direction button or the thumb, that portion of the scroll bar will be displayed in 3-D.
FSB_FLAT_MODE	A standard flat scroll bar is displayed. When the mouse moves over a direction button or the thumb, that portion of the scroll bar will be displayed in inverted colors.
FSB_REGULAR_MODE	A normal, nonflat scroll bar is displayed. No special visual effects will be applied.

Removing Flat Scroll Bars

If you wish to remove flat scroll bars from your window, call the **UninitializeFlatSB API**, passing the handle to the window. This API only removes flat scroll bars from your window at run time. You do not need to call this API when your window is destroyed.

Flat Scroll Bar Reference

Flat Scroll Bar Functions

InitializeFlatSB

Initializes flat scroll bars for a particular window.

```
BOOL InitializeFlatSB(
  HWND hwnd
);
```

Parameters

hwnd
Handle to the window that will receive flat scroll bars.

Return Values

Returns nonzero if successful, or zero otherwise.

Remarks

This API must be called before any other flat scroll bar APIs are called. The window will receive flat scroll bars by default. The scroll bar style can be changed with the **FlatSB_SetScrollProp API**.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.
Import Library: comctl32.lib.

FlatSB_EnableScrollBar

Enables or disables one or both flat scroll bar direction buttons. If flat scroll bars are not initialized for the window, this function calls the standard **EnableScrollBar** API.

```
BOOL FlatSB_EnableScrollBar(
  HWND hwnd,
  int wSBflags,
  UINT wArrows
);
```

Parameters

hwnd

Handle to the window that contains the flat scroll bar. This window handle must have been passed previously in a call to **InitializeFlatSB**.

wSBflags

Parameter that specifies the scroll bar type. It can be one of the following values:

SB_BOTH	Enables or disables the direction buttons on the horizontal and vertical scroll bars.
SB_HORZ	Enables or disables the direction buttons on the horizontal scroll bar.
SB_VERT	Enables or disables the direction buttons on the vertical scroll bar.

wArrows

Parameter that specifies whether the scroll bar arrows are enabled or disabled and indicates which arrows are enabled or disabled. It can be one of the following values:

ESB_DISABLE_BOTH	Disables both direction buttons on the specified scroll bar.
ESB_DISABLE_DOWN	Disables the down direction button on the vertical scroll bar.
ESB_DISABLE_LEFT	Disables the left direction button on the horizontal scroll bar.
ESB_DISABLE_LTUP	Disables the left direction button on the horizontal scroll bar or the up direction button on the vertical scroll bar.
ESB_DISABLE_RIGHT	Disables the right direction button on the horizontal scroll bar.
ESB_DISABLE_RTDN	Disables the right direction button on the horizontal scroll bar or the down direction button on the vertical scroll bar.
ESB_DISABLE_UP	Disables the up direction button on the vertical scroll bar.
ESB_ENABLE_BOTH	Enables both direction buttons on the specified scroll bar.

Return Values

Returns nonzero if the scroll bar changes, or zero otherwise.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.
Import Library: comctl32.lib.

FlatSB_GetScrollInfo

Retrieves the information for a flat scroll bar. If flat scroll bars are not initialized for the window, this function calls the standard **GetScrollInfo** API.

```
BOOL FlatSB_GetScrollInfo(
  HWND hwnd,
  int fnBar,
  LPSCROLLINFO lpsi
);
```

Parameters

hwnd
Handle to the window that contains the flat scroll bar. This window handle must have been passed previously in a call to **InitializeFlatSB**.

fnBar
Parameter that specifies the scroll bar type. It can be one of the following values:

SB_HORZ	Retrieves the information for the horizontal scroll bar.
SB_VERT	Retrieves the information for the vertical scroll bar.

lpsi
Address of a **SCROLLINFO** structure that will receive the information for the specified scroll bar. The **cbSize** and **fMask** members of the structure must be filled out prior to calling **FlatSB_GetScrollInfo**. The **fMask** member specifies which properties should be retrieved and can be any combination of the following values:

SIF_PAGE	Retrieves the page information for the flat scroll bar. This will be placed in the **nPage** member of the **SCROLLINFO** structure.
SIF_POS	Retrieves the position information for the flat scroll bar. This will be placed in the **nPos** member of the **SCROLLINFO** structure.
SIF_RANGE	Retrieves the range information for the flat scroll bar. This will be placed in the **nMin** and **nMax** members of the **SCROLLINFO** structure.
SIF_ALL	A combination of SIF_PAGE, SIF_POS, and SIF_RANGE.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.
Import Library: comctl32.lib.

FlatSB_GetScrollPos

Retrieves the thumb position in a flat scroll bar. If flat scroll bars are not initialized for the window, this function calls the standard **GetScrollPos** API.

```
int FlatSB_GetScrollPos(
  HWND hwnd,
  int code
);
```

Parameters

hwnd
Handle to the window that contains the flat scroll bar. This window handle must have been passed previously in a call to **InitializeFlatSB**.

code
Parameter that specifies the scroll bar type. It can be one of the following values:

SB_HORZ	Retrieves the thumb position of the horizontal scroll bar.
SB_VERT	Retrieves the thumb position of the vertical scroll bar.

Return Values

Returns the current thumb position of the specified flat scroll bar.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.
Import Library: comctl32.lib.

FlatSB_GetScrollProp

Retrieves the properties for a flat scroll bar. This function can also be used to determine if **InitializeFlatSB** has been called for this window.

```
BOOL FlatSB_GetScrollProp(
  HWND hwnd,
  UINT index,
  LPINT pValue
);
```

Parameters

hwnd

Handle to the window that contains the flat scroll bar. This window handle must have been passed previously in a call to **InitializeFlatSB**.

index

Parameter that determines what *pValue* represents and which property is being retrieved. It can be one of the following values:

WSB_PROP_CXHSCROLL	*pValue* is an address of an INT value that receives the width, in pixels, of the direction buttons in a horizontal scroll bar.
WSB_PROP_CXHTHUMB	*pValue* is an address of an INT value that receives the width, in pixels, of the thumb in a horizontal scroll bar.
WSB_PROP_CXVSCROLL	*pValue* is an address of an INT value that receives the width, in pixels, of a vertical scroll bar.
WSB_PROP_CYHSCROLL	*pValue* is an address of an INT value that receives the height, in pixels, of a horizontal scroll bar.
WSB_PROP_CYVSCROLL	*pValue* is an address of an INT value that receives the height, in pixels, of the direction buttons in a vertical scroll bar.
WSB_PROP_CYVTHUMB	*pValue* is an address of an INT value that receives the height, in pixels, of the thumb in a vertical scroll bar.
WSB_PROP_HBKGCOLOR	*pValue* is an address of a **COLORREF** value that receives the background color in a horizontal scroll bar.

(continued)

(continued)

WSB_PROP_HSTYLE	*pValue* is an address of an INT value that receives one of the following visual effects for the horizontal scroll bar. FSB_ENCARTA_MODE A standard flat scroll bar is displayed. When the mouse moves over a direction button or the thumb, that portion of the scroll bar is displayed in 3-D. FSB_FLAT_MODE A standard flat scroll bar is displayed. When the mouse moves over a direction button or the thumb, that portion of the scroll bar is displayed in inverted colors. FSB_REGULAR_MODE A normal, nonflat scroll bar is displayed. No special visual effects are applied.
WSB_PROP_PALETTE	*pValue* is an address of an **HPALETTE** value that receives the palette that a scroll bar uses when drawing.
WSB_PROP_VBKGCOLOR	*pValue* is an address of a **COLORREF** value that receives the background color in a vertical scroll bar.
WSB_PROP_VSTYLE	*pValue* is an address of an INT value that receives one of the following visual effects for the vertical scroll bar. FSB_ENCARTA_MODE A standard flat scroll bar is displayed. When the mouse moves over a direction button or the thumb, that portion of the scroll bar is displayed in 3-D. FSB_FLAT_MODE A standard flat scroll bar is displayed. When the mouse moves over a direction button or the thumb, that portion of the scroll bar is displayed in inverted colors. FSB_REGULAR_MODE A normal, nonflat scroll bar is displayed. No special visual effects are applied.
WSB_PROP_WINSTYLE	*pValue* is an address of an INT value that receives the WS_HSCROLL and WS_VSCROLL style bits contained by the current window.

pValue
Address that receives the requested data. This parameter depends on the flag passed in *index*.

Return Values

Returns nonzero if successful, or zero otherwise. If *index* is WSB_PROP_HSTYLE, the return is nonzero if **InitializeFlatSB** has been called for this window, or zero otherwise.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.
Import Library: comctl32.lib.

FlatSB_GetScrollRange

Retrieves the scroll range for a flat scroll bar. If flat scroll bars are not initialized for the window, this function calls the standard **GetScrollRange** API.

```
BOOL FlatSB_GetScrollRange(
  HWND hwnd,
  int code,
  LPINT lpMinPos,
  LPINT lpMaxPos
);
```

Parameters

hwnd
Handle to the window that contains the flat scroll bar. This window handle must have been passed previously in a call to **InitializeFlatSB**.

code
Parameter that specifies the scroll bar type. It can be one of the following values:

SB_HORZ	Retrieves the scroll range of the horizontal scroll bar.
SB_VERT	Retrieves the scroll range of the vertical scroll bar.

lpMinPos
Address of an INT value that receives the minimum scroll range value.

lpMaxPos
Address of an INT value that receives the maximum scroll range value.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.
Import Library: comctl32.lib.

FlatSB_SetScrollInfo

Sets the information for a flat scroll bar. If flat scroll bars are not initialized for the window, this function calls the standard **SetScrollInfo** API.

```
int FlatSB_SetScrollInfo(
  HWND hwnd,
  int fnBar,
  LPSCROLLINFO lpsi,
  BOOL fRedraw
);
```

Parameters

hwnd

Handle to the window that contains the flat scroll bar. This window handle must have been passed previously in a call to **InitializeFlatSB**.

fnBar

Parameter that specifies the scroll bar type. It can be one of the following values:

SB_HORZ	Sets the information for the horizontal scroll bar.
SB_VERT	Sets the information for the vertical scroll bar.

lpsi

Address of a **SCROLLINFO** structure that contains the new information for the specified scroll bar. The **cbSize** and **fMask** members of the structure must be filled in prior to calling **FlatSB_SetScrollInfo**. The **fMask** member specifies which members of the structure contain valid information and can be any combination of the following values:

SIF_DISABLENOSCROLL	Disables the scroll bar if the new information would cause the scroll bar to be removed.
SIF_ALL	A combination of SIF_PAGE, SIF_POS, and SIF_RANGE.

SIF_PAGE	Sets the page information for the flat scroll bar. The **nPage** member of the **SCROLLINFO** structure must contain the new page value.
SIF_POS	Sets the position information for the flat scroll bar. The **nPos** member of the **SCROLLINFO** structure must contain the new position value.
SIF_RANGE	Sets the range information for the flat scroll bar. The **nMin** and **nMax** members of the **SCROLLINFO** structure must contain the new range values.

fRedraw
Parameter that specifies whether the scroll bar should be redrawn immediately to reflect the change. If this parameter is TRUE, the scroll bar is redrawn; if it is FALSE, the scroll bar is not redrawn.

Return Values

Returns the current scroll position. If the call to **FlatSB_SetScrollInfo** changes the scroll position, then the previous position is returned.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.
Import Library: comctl32.lib.

FlatSB_SetScrollPos

Sets the current position of the thumb in a flat scroll bar. If flat scroll bars are not initialized for the window, this function calls the standard **SetScrollPos** API.

```
int FlatSB_SetScrollPos(
  HWND hwnd,
  int code,
  int nPos,
  BOOL fRedraw
);
```

Parameters

hwnd

Handle to the window that contains the flat scroll bar. This window handle must have been passed previously in a call to **InitializeFlatSB**.

code

Parameter that specifies the scroll bar type. It can be one of the following values:

SB_HORZ	Sets the thumb position of the horizontal scroll bar.
SB_VERT	Sets the thumb position of the vertical scroll bar.

nPos

Parameter that specifies the new thumb position.

fRedraw

Parameter that specifies whether the scroll bar should be redrawn immediately to reflect the change. If this parameter is TRUE, the scroll bar is redrawn; if it is FALSE, the scroll bar is not redrawn.

Return Values

Returns the previous position of the thumb in the specified flat scroll bar.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.
Import Library: comctl32.lib.

FlatSB_SetScrollProp

Sets the properties for a flat scroll bar.

```
BOOL FlatSB_SetScrollProp(
  HWND hwnd,
  UINT index,
  int newValue,
  BOOL fRedraw
);
```

Parameters

hwnd

Handle to the window that contains the flat scroll bar. This window handle must have been passed previously in a call to **InitializeFlatSB**.

index

Parameter that determines what *newValue* represents and which property is being set. This parameter can be one of the following values:

WSB_PROP_CXHSCROLL	*newValue* is an INT value that represents the width, in pixels, of the direction buttons in a horizontal scroll bar.
WSB_PROP_CXHTHUMB	*newValue* is an INT value that represents the width, in pixels, of the thumb in a horizontal scroll bar.
WSB_PROP_CXVSCROLL	*newValue* is an INT value that represents the width, in pixels, of the vertical scroll bar.
WSB_PROP_CYHSCROLL	*newValue* is an INT value that represents the height, in pixels, of the horizontal scroll bar.
WSB_PROP_CYVSCROLL	*newValue* is an INT value that represents the height, in pixels, of the direction buttons in a vertical scroll bar.
WSB_PROP_CYVTHUMB	*newValue* is an INT value that represents the height, in pixels, of the thumb in a vertical scroll bar.
WSB_PROP_HBKGCOLOR	*newValue* is a **COLORREF** value that represents the background color in a horizontal scroll bar.
WSB_PROP_HSTYLE	*newValue* is one of the following values that changes the visual effects for the horizontal scroll bar. FSB_ENCARTA_MODE A standard flat scroll bar is displayed. When the mouse moves over a direction button or the thumb, that portion of the scroll bar will be displayed in 3-D. FSB_FLAT_MODE A standard flat scroll bar is displayed. When the mouse moves over a direction button or the thumb, that portion of the scroll bar will be displayed in inverted colors. FSB_REGULAR_MODE A normal, nonflat scroll bar is displayed. No special visual effects will be applied.

(continued)

(continued)

WSB_PROP_PALETTE	*newValue* is an **HPALETTE** value that represents the new palette that the scroll bar should use when drawing.
WSB_PROP_VBKGCOLOR	*newValue* is a **COLORREF** value that represents the background color in a vertical scroll bar.
WSB_PROP_VSTYLE	*newValue* is one of the following values that changes the visual effects for the vertical scroll bar: FSB_ENCARTA_MODE A standard flat scroll bar is displayed. When the mouse moves over a direction button or the thumb, that portion of the scroll bar will be displayed in 3-D. FSB_FLAT_MODE A standard flat scroll bar is displayed. When the mouse moves over a direction button or the thumb, that portion of the scroll bar will be displayed in inverted colors. FSB_REGULAR_MODE A normal, nonflat scroll bar is displayed. No special visual effects will be applied.

newValue
New value to set. This parameter depends on the flag passed in *index*.

fRedraw
Parameter that specifies whether the scroll bar should be redrawn immediately to reflect the change. If this parameter is TRUE, the scroll bar is redrawn; if it is FALSE, the scroll bar is not redrawn.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.
Import Library: comctl32.lib.

FlatSB_SetScrollRange

Sets the scroll range of a flat scroll bar. If flat scroll bars are not initialized for the window, this function calls the standard **SetScrollRange** API.

```
int FlatSB_SetScrollRange(
  HWND hwnd,
  int code,
  int nMinPos,
  int nMaxPos,
  BOOL fRedraw
);
```

Parameters

hwnd
Handle to the window that contains the flat scroll bar. This window handle must have been passed previously in a call to **InitializeFlatSB**.

code
Parameter that specifies the scroll bar type. It can be one of the following values:

SB_HORZ	Sets the scroll range of the horizontal scroll bar.
SB_VERT	Sets the scroll range of the vertical scroll bar.

nMinPos
Parameter that specifies the new minimum scroll range value.

nMaxPos
Parameter that specifies the new maximum scroll range value.

fRedraw
Parameter that specifies whether the scroll bar should be redrawn immediately to reflect the change. If this parameter is TRUE, the scroll bar is redrawn; if it is FALSE, the scroll bar is not redrawn.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.
Import Library: comctl32.lib.

FlatSB_ShowScrollBar

Shows or hides a flat scroll bar. If flat scroll bars are not initialized for the window, this function calls the standard **ShowScrollBar** API.

```
BOOL FlatSB_ShowScrollBar(
  HWND hwnd,
  int code,
  BOOL fShow
);
```

Parameters

hwnd

Handle to the window that contains the flat scroll bar. This window handle must have been passed previously in a call to InitializeFlatSB.

code

Parameter that specifies the scroll bar type. It can be one of the following values:

SB_BOTH	Shows or hides the horizontal and vertical scroll bars.
SB_HORZ	Shows or hides the horizontal scroll bar.
SB_VERT	Shows or hides the vertical scroll bar.

fShow

Parameter that specifies whether the scroll bar should be shown or hidden. If this parameter is nonzero, the scroll bar will be shown; if it is zero, the scroll bar will be hidden.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.
Import Library: comctl32.lib.

UninitializeFlatSB

Uninitializes flat scroll bars for a particular window. The specified window will revert to having standard scroll bars.

```
HRESULT UninitializeFlatSB(
  HWND hwnd
);
```

Parameters

hwnd
Handle to the window with the flat scroll bars that will be uninitialized.

Return Values

Returns one of the following values:

E_FAIL	One of the window's scroll bars is currently in use. The operation cannot be completed at this time.
S_FALSE	The window doesn't have flat scroll bars initialized.
S_OK	The operation was successful.

Requirements

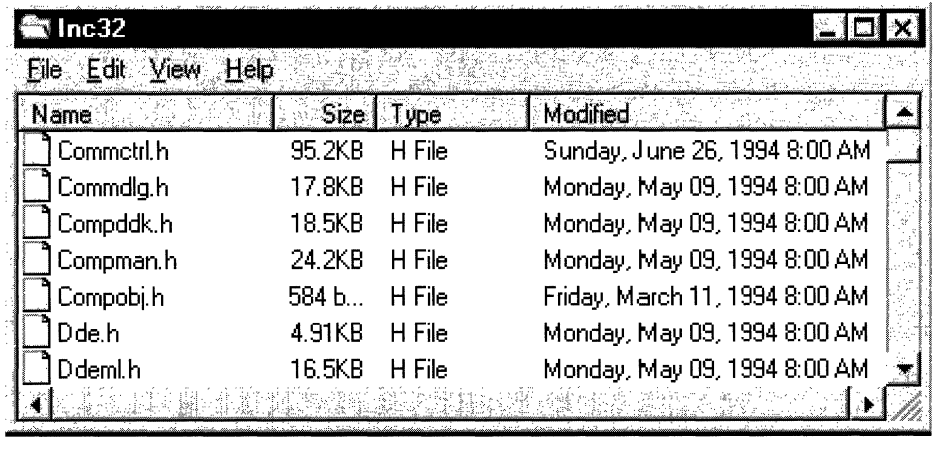
Version 4.71 and later of Comctl32.dll.

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 commctrl.h.
Import Library: comctl32.lib.

CHAPTER 15

Header Controls

A *header control* is a window that is usually positioned above columns of text or numbers. It contains a title for each column, and it can be divided into parts. The user can drag the dividers that separate the parts to set the width of each column. The following illustration shows a header control that has labeled columns that give detailed information about files in a directory.



Using Header Controls

You can create a header control by using the **CreateWindowEx** function, specifying the WC_HEADER window class and the appropriate header styles. This window class is registered when the common control dynamic link library (DLL) is loaded. To ensure that this DLL is loaded, use the **InitCommonControlsEx** function. After you create a header control, you can divide it into parts, set the text in each part, and control the appearance of the window by using header window messages.

Header Control Size and Position

Typically, you must set the size and position of a header control to fit within the boundaries of a particular rectangle, such as the client area of a window. By using the **HDM_LAYOUT** message, you can retrieve the appropriate size and position values from the header control.

When sending **HDM_LAYOUT**, you specify the address of an **HDLAYOUT** structure that contains the coordinates of the rectangle that the header control is to occupy and provides a pointer to a **WINDOWPOS** structure. The control fills the **WINDOWPOS**

structure with size and position values appropriate for positioning the control along the top of the specified rectangle. The height value is the sum of the heights of the control's horizontal borders and the average height of characters in the font currently selected into the control's device context.

If you want to use **HDM_LAYOUT** to set the initial size and position of a header control, set the initial visibility state of the control so that it is hidden. After sending **HDM_LAYOUT** to retrieve the size and position values, you can use the **SetWindowPos** function to set the new size, position, and visibility state.

Items

A header control typically has several header items that define the columns of the control. You add an item to a header control by sending the **HDM_INSERTITEM** message to the control. The message includes the address of an **HDITEM** structure. This structure defines the properties of the header item, which can include a string, a bitmapped image, an initial size, and an application-defined 32-bit value.

The **fmt** member of an item's **HDITEM** structure can include either the HDF_STRING or HDF_BITMAP flag to indicate whether the control displays the item's string or bitmap. If you want to display both a string and a bitmap, create an owner-drawn item by setting the **fmt** member to include the HDF_OWNERDRAW flag. The **HDITEM** structure also specifies formatting flags that tell the control whether to center, left-align, or right-align the string or bitmap in the item's rectangle.

HDM_INSERTITEM returns the index of the newly added item. You can use the index in other messages to set properties or retrieve information about the item. You can delete an item by using the **HDM_DELETEITEM** message, specifying the index of the item to delete.

You can use the **HDM_SETITEM** message to set the properties of an existing header item and the **HDM_GETITEM** message to retrieve the current properties of an item. To retrieve a count of the items in a header control, use the **HDM_GETITEMCOUNT** message.

Owner-Drawn Header Controls

You can define individual items of a header control to be owner-drawn items. Using this technique gives you more control than you would otherwise have over the appearance of a header item.

You can use the **HDM_INSERTITEM** message to insert a new owner-drawn item into a header control or the **HDM_SETITEM** message to change an existing item to an owner-drawn item. Both messages include the address of an **HDITEM** structure, which should have the **fmt** member set to the HDF_OWNERDRAW value.

When a header control must draw an owner-drawn item, it sends the **WM_DRAWITEM** message to the parent window. The *wParam* parameter of the message is the child window identifier of the header control, and the *lParam* parameter is an address of a

DRAWITEMSTRUCT structure. The parent window uses the information in the structure to draw the item. For an owner-drawn item in a header control, the **DRAWITEMSTRUCT** structure contains the following information.

Member	Description
CtlType	ODT_HEADER owner-drawn control type.
CtlID	Child-window identifier of the header control.
itemID	Index of the item to be drawn.
itemAction	ODA_DRAWENTIRE drawing-action flag.
itemState	ODS_SELECTED drawing-action flag if the cursor is on the item and the mouse button is down. Otherwise, this member is zero.
hwndItem	Handle to the header control.
hDC	Handle to the device context of the header control.
rcItem	Coordinates of the header item to be drawn. The coordinates are relative to the upper-left corner of the header control.
itemData	Application-defined 32-bit value associated with the item.

Header Control Notification Messages

A header control sends notification messages to its parent window when the user clicks or double-clicks an item, when the user drags an item divider, and when the attributes of an item change. The parent window receives the notifications in the form of **WM_NOTIFY** messages. The following notifications are used with header controls.

Notification	Description
HDN_BEGINTRACK	Signals the start of divider dragging.
HDN_DIVIDERDBLCLICK	Indicates that the user double-clicked a divider.
HDN_ENDTRACK	Signals the end of divider dragging.
HDN_ITEMCHANGED	Indicates a change in the attributes of an item.
HDN_ITEMCHANGING	Indicates that the attributes of an item are about to change.
HDN_ITEMCLICK	Indicates that the user clicked an item.
HDN_ITEMDBLCLICK	Indicates that the user double-clicked an item.
HDN_TRACK	Indicates that the user dragged a divider.

Default Header Control Message Processing

This section describes the window messages handled by the window procedure for the WC_HEADER window class.

Message	Processing performed
WM_CREATE	Initializes the header control.
WM_DESTROY	Uninitializes the header control.
WM_ERASEBKGND	Fills the background of the header control using the control's current background color.
WM_GETDLGCODE	Returns a combination of the DLGC_WANTTAB and DLGC_WANTARROWS values.
WM_GETFONT	Returns the handle to the current font, which is used by the header control to draw its text.
WM_LBUTTONDBLCLK	Captures mouse input. If the mouse cursor is on a divider, the control sends the **HDN_BEGINTRACK** notification message and begins dragging the divider. If the cursor is on an item, the item is displayed in the pressed state.
WM_LBUTTONDOWN	Same as the WM_LBUTTONDBLCLK message.
WM_LBUTTONUP	Releases the mouse capture. If the control was tracking mouse movement, it sends the **HDN_ENDTRACK** notification message and redraws the header control. Otherwise, the control sends the **HDN_ITEMCLICK** notification message and redraws the header item that was clicked.
WM_MOUSEMOVE	If a divider is being dragged, the control sends the **HDN_TRACK** notification message and displays the item at the new position. If the left mouse button is down and the cursor is on an item, the item is displayed in the pressed state.
WM_NCCREATE	Allocates and initializes an internal data structure.
WM_NCDESTROY	Frees the resources allocated by the header control after the header control is uninitialized.
WM_PAINT	Paints the invalid region of the header control. If the *wParam* parameter is non-NULL, the control assumes that the value is an HDC and paints using that device context.
WM_SETCURSOR	Sets the cursor shape, depending on whether the cursor is on a divider or in a header item.
WM_SETFONT	Selects a new font handle into the device context for the header control.

Creating a Header Control

The following example demonstrates how to create a header control and position it along the top of the parent window's client area. The control is initially hidden. The **HDM_LAYOUT** message is used to calculate the size and position of the control within the parent window. The control is then repositioned and made visible.

```
// DoCreateHeader - creates a header control that is
//     positioned along the top of the parent window's
//     client area.
// Returns the handle to the header control.
// hwndParent - handle to the parent window.
//
// Global variable
//    g_hinst - handle to the application instance
extern HINSTANCE g_hinst;

HWND DoCreateHeader(HWND hwndParent)
{
        HWND hwndHeader;
        RECT rcParent;
        HDLAYOUT hdl;
        WINDOWPOS wp;

        // Ensure that the common control DLL is loaded,
        // and then create the header control.
        InitCommonControlsEx();

        if ((hwndHeader = CreateWindowEx(0, WC_HEADER, (LPCTSTR) NULL,
                WS_CHILD | WS_BORDER | HDS_BUTTONS | HDS_HORZ,
                0, 0, 0, 0, hwndParent, (HMENU) ID_HEADER, g_hinst,
                (LPVOID) NULL)) == NULL)
            return (HWND) NULL;

        // Retrieve the bounding rectangle of the parent
        // window's client area, and then request size and
        // position values from the header control.
        GetClientRect(hwndParent, &rcParent);

        hdl.prc = &rcParent;
        hdl.pwpos = &wp;
        if (!SendMessage(hwndHeader, HDM_LAYOUT, 0, (LPARAM) &hdl))
            return (HWND) NULL;

        // Set the size, position, and visibility of the
        // header control.
        SetWindowPos(hwndHeader, wp.hwndInsertAfter, wp.x, wp.y,
            wp.cx, wp.cy, wp.flags | SWP_SHOWWINDOW);

        return hwndHeader;
}
```

Adding an Item to a Header Control

The following example demonstrates how to use the **HDM_INSERTITEM** message and the **HDITEM** structure to add an item to a header control. The new item consists of a string that is left-justified within the item rectangle.

```
// DoInsertItem - inserts an item into a header control.
// Returns the index of the new item.
// hwndHeader - handle to the header control.
// iInsertAfter - index of the previous item.
// nWidth - width of the new item.
// lpsz - address of the item string.
int DoInsertItem(HWND hwndHeader, int iInsertAfter,
    int nWidth, LPSTR lpsz)
{
    HDITEM hdi;
    int index;

    hdi.mask = HDI_TEXT | HDI_FORMAT | HDI_WIDTH;
    hdi.pszText = lpsz;
    hdi.cxy = nWidth;
    hdi.cchTextMax = lstrlen(hdi.pszText);
    hdi.fmt = HDF_LEFT | HDF_STRING;

    index = SendMessage(hwndHeader, HDM_INSERTITEM,
        (WPARAM) iInsertAfter, (LPARAM) &hdi);

    return index;
}
```

Header Control Updates in Internet Explorer

Header controls in Microsoft Internet Explorer support the following new features.

Image Lists

New messages for this feature include **HDM_GETIMAGELIST** and **HDM_SETIMAGELIST**. The header control structure **HDITEM** has been updated to support image lists. Image lists can be used with current bitmap support.

Callback Items

Currently, callback support includes header item text and images. Setting a header item's text to the LPSTR_TEXTCALLBACK value or its image to the I_IMAGECALLBACK value will cause the control to send an **HDN_GETDISPINFO** message to request callback information as needed. **HDN_GETDISPINFO** is supported by the new **NMHDDISPINFO** structure.

Custom Item Ordering
The new messages that support this feature are: **HDM_GETORDERARRAY**, **HDM_SETORDERARRAY**, and **HDM_ORDERTOINDEX**.

Drag-and-Drop Manipulation
Drag-and-drop reordering of header items is now available. Implement drag-and-drop support by including the **HDS_DRAGDROP** style when creating the control. By default, the header control automatically handles drag-and-drop overhead and graphic effects. However, the owner of the control can manually support drag-and-drop manipulation by handling the **HDN_BEGINDRAG** and **HDN_ENDDRAG** notification messages. While handling these notifications, the owner might need to send **HDM_CREATEDRAGIMAGE** and **HDM_SETHOTDIVIDER** messages.

Hot Tracking
When this feature is enabled, the item that is under the pointer will be highlighted. You can check whether or not hot tracking is enabled by calling **SystemParametersInfo**. To enable hot tracking in a header control, create it with the **HDS_HOTTRACK** style.

Text, Bitmaps, and Images
Items can simultaneously display item text, a bitmap, and an image.

Header Control Styles

Header controls have a number of styles, described below, that determine the control's appearance and behavior. You set the initial styles when you create the header control. To retrieve and change the styles after creating the control, use the **GetWindowLong** and **SetWindowLong** functions.

HDS_BUTTONS
Each item in the control looks and behaves like a push button. This style is useful if an application carries out a task when the user clicks an item in the header control. For example, an application could sort information in the columns differently depending on which item the user clicks.

HDS_DRAGDROP
Version 4.70. Allows drag-and-drop reordering of header items.

HDS_FILTERBAR
Version 5.80. Include a filter bar as part of the standard header control. This bar allows users to conveniently apply a filter to the display. Calls to **HDM_LAYOUT** will yield a new size for the control and cause the list view to update.

HDS_FULLDRAG
Version 4.70. Causes the header control to display column contents even while the user resizes a column.

HDS_HIDDEN
Indicates a header control that is intended to be hidden. This style does not hide the control. Instead, when you send the **HDM_LAYOUT** message to a header control with the HDS_HIDDEN style, the control returns zero in the **cy** member of the **WINDOWPOS** structure. You would then hide the control by setting its height to zero.

This can be useful when you want to use the control as an information container instead of a visual control.

HDS_HORZ
Creates a header control with a horizontal orientation.

HDS_HOTTRACK
Version 4.70. Enables hot tracking.

Header Control Reference

Header Control Messages

HDM_CLEARFILTER

Clears the filter for a given header control. You can send this message explicitly or use the **Header_ClearFilter** macro.

```
HDM_CLEARFILTER
    wParam = (WPARAM) (HWND) hwnd;
    lParam = (LPARAM) (int) i;
```

Parameters

hwnd
Handle to the header control.

i
Column value indicating which filter to clear.

Return Values

Returns the index of the filter control being cleared.

Remarks

If the column value is specified as –1, all the filters will be cleared, and the **HDN_FILTERCHANGE** notification will be sent only once.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.

See Also

Header_ClearAllFilters

HDM_CREATEDRAGIMAGE

Creates a semi-transparent version of an item's image for use as a dragging image. You can send this message explicitly or use the **Header_CreateDragImage** macro.

```
HDM_CREATEDRAGIMAGE
    wParam = (WPARAM) (int) iIndex;
    lParam = 0;
```

Parameters

iIndex

Zero-based index of the item within the header control. The image assigned to this item is the basis for the transparent image.

Return Values

Returns a handle to an image list that contains the new image as its only element.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

HDM_DELETEITEM

Deletes an item from a header control. You can send this message explicitly or use the **Header_DeleteItem** macro.

```
HDM_DELETEITEM
    wParam = (WPARAM) (int) index;
    lParam = 0;
```

Parameters

index

Index of the item to delete.

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: Requires version 1.0 or later.
Header: Declared in commctrl.h.

HDM_EDITFILTER

Starts editing the specified filter.

```
HDM_EDITFILTER
    wParam = (WPARAM) (int) i;
    lParam = MAKELPARAM(fDiscardChanges, 0)
);
```

Parameters

i

Value specifying the column to edit.

fDiscardChanges

Flag that specifies how to handle the user's editing changes. Use this flag to specify what to do if the user is in the process of editing the filter when the message is sent.

TRUE	Discard the changes made by the user.
FALSE	Accept the changes made by the user.

Return Values

Returns the index of the filter being edited.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.

See Also

HDM_CLEARFILTER

HDM_GETBITMAPMARGIN

Retrieves the width of the bitmap margin for a header control. You can send this message explicitly or use the **Header_GetBitmapMargin** macro.

```
HDM_GETBITMAPMARGIN
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the width of the bitmap margin in pixels. If the bitmap margin was not previously specified, the default value of 3*GetSystemMetrics(CX_EDGE) is returned.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.

See Also

HDM_SETBITMAPMARGIN

HDM_GETIMAGELIST

Retrieves the handle to the image list that has been set for an existing header control. You can send this message explicitly or use the **Header_GetImageList** macro.

```
HDM_GETIMAGELIST
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a handle to the image list set for the header control.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).

Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

HDM_GETITEM

Retrieves information about an item in a header control. You can send this message explicitly or use the **Header_GetItem** macro.

```
HDM_GETITEM
    wParam = (WPARAM) (int) index;
    lParam = (LPARAM) (LPHDITEM) phdi;
```

Parameters

index
Index of the item for which information is to be retrieved.

phdi
Address of an **HDITEM** structure. When the message is sent, the **mask** member indicates the type of information being requested. When the message returns, the other members receive the requested information. If the **mask** member specifies zero, the message returns TRUE but copies no information to the structure.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

If the HDI_TEXT flag is set in the **mask** member of the **HDITEM** structure, the control may change the **pszText** member of the structure to point to the new text instead of filling the buffer with the requested text. Applications should not assume that the text will always be placed in the requested buffer.

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 commctrl.h.

HDM_GETITEMCOUNT

Retrieves a count of the items in a header control. You can send this message explicitly or use the **Header_GetItemCount** macro.

```
HDM_GETITEMCOUNT
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the number of items if successful, or –1 otherwise.

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 commctrl.h.

HDM_GETITEMRECT

Retrieves the bounding rectangle for a given item in a header control. You can send this message explicitly or use the **Header_GetItemRect** macro.

```
HDM_GETITEMRECT
    wParam = (WPARAM) (int) iIndex;
    lParam = (LPARAM) lpItemRect;
```

Parameters

iIndex
Zero-based index of the header control item for which to retrieve the bounding rectangle.

lpItemRect
Address of a **RECT** structure that receives the bounding rectangle information.

Return Values

Returns nonzero if successful, or zero otherwise.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

HDM_GETORDERARRAY

Retrieves the current left-to-right order of items in a header control. You can send this message explicitly or use the **Header_GetOrderArray** macro.

```
HDM_GETORDERARRAY
    wParam = (WPARAM) (int) iSize;
    lParam = (LPARAM) (LPINT) lpiArray;
```

Parameters

iSize

Number of integer elements that *lpiArray* can hold. This value must be equal to the number of items in the control (see **HDM_GETITEMCOUNT**).

lpiArray

Address of an array of integers that receive the index values for items in the header. The number of elements in this array is specified in *iSize* and must be equal to or greater than the number of items in the control. For example, the following code fragment will reserve enough memory to hold the index values:

```
int iItems,
    *lpiArray;

// Get memory for buffer.
(iItems = SendMessage(hwndHD, HDM_GETITEMCOUNT, 0,0))!=-1)
    if(!(lpiArray = calloc(iItems,sizeof(int))))
MessageBox(hwnd, "Out of memory.","Error", MB_OK);
```

Return Values

Returns nonzero if successful, and the buffer at *lpiArray* receives the item number for each item in the header control in the order in which they appear from left to right. Otherwise, the message returns zero.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

HDM_GETUNICODEFORMAT

Retrieves the UNICODE character format flag for the control. You can send this message explicitly or use the **Header_GetUnicodeFormat** macro.

```
HDM_GETUNICODEFORMAT
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the UNICODE format flag for the control. If this value is nonzero, the control is using UNICODE characters. If this value is zero, the control is using ANSI characters.

Remarks

See the remarks for **CCM_GETUNICODEFORMAT** for a discussion of this message.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

See Also

HDM_SETUNICODEFORMAT

HDM_HITTEST

Tests a point to determine which header item, if any, is at the specified point.

```
HDM_HITTEST
    wParam = 0;
    lParam = (LPARAM) (LPHDHITTESTINFO) phdht1;
```

Parameters

phdhti

Address of an **HDHITTESTINFO** structure that contains the position to test and receives information about the results of the test.

Return Values

Returns the index of the item at the specified position, if any, or –1 otherwise.

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 commctrl.h.

HDM_INSERTITEM

Inserts a new item into a header control. You can send this message explicitly or use the **Header_InsertItem** macro.

```
HDM_INSERTITEM
    wParam = (WPARAM) (int) index;
    lParam = (LPARAM) (const LPHDITEM) phdi;
```

Parameters

index
Index of the item after which the new item is to be inserted. The new item is inserted at the end of the header control if *index* is greater than or equal to the number of items in the control. If *index* is zero, the new item is inserted at the beginning of the header control.

phdi
Address of an **HDITEM** structure that contains information about the new item.

Return Values

Returns the index of the new item if successful, or –1 otherwise.

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 commctrl.h.

HDM_LAYOUT

Retrieves the correct size and position of a header control within the parent window. You can send this message explicitly or use the **Header_Layout** macro.

```
HDM_LAYOUT
    wParam = 0;
    lParam = (LPARAM) (LPHDLAYOUT) playout;
```

Parameters

playout

Address of an **HDLAYOUT** structure. The **prc** member specifies the coordinates of a rectangle, and the **pwpos** member receives the size and position for the header control within the rectangle.

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: Requires version 1.0 or later.
Header: Declared in commctrl.h.

HDM_ORDERTOINDEX

Retrieves an index value for an item based on its order in the header control. You can send this message explicitly or use the **Header_OrderToIndex** macro.

```
HDM_ORDERTOINDEX
    wParam = (WPARAM) iOrder;
    lParam = 0;
```

Parameters

iOrder

Order in which the item appears within the header control, from left to right. For example, the index value of the item in the far left column would be 0. The value for the next item to the right would be 1, and so on.

Return Values

Returns INT that indicates the item index. If *iOrder* is invalid (negative or too large), the return equals *iOrder*.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

HDM_SETBITMAPMARGIN

Sets the width of the margin, specified in pixels, of a bitmap in an existing header control. You can send this message explicitly or use the **Header_SetBitmapMargin** macro.

```
HDM_SETBITMAPMARGIN
    wParam = (WPARAM) (int) iWidth;
    lParam = 0;
```

Parameters

iWidth

Width, specified in pixels, of the margin that surrounds a bitmap within an existing header control.

Return Values

Returns the width of the bitmap margin, in pixels. If the bitmap margin was not previously specified, the default value of 3*GetSystemMetrics(CX_EDGE) is returned.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.

See Also

HDM_GETBITMAPMARGIN

HDM_SETFILTERCHANGETIMEOUT

Sets the timeout interval between the time a change takes place in the filter attributes and the posting of an **HDN_FILTERCHANGED** notification. You can send this message explicitly or use the **Header_SetFilterChangeTimeout** macro.

```
HDM_SETFILTERCHANGETIMEOUT
    wParam = 0;
    lParam = (LPARAM) i;
```

Parameters

i

Timeout value, in milliseconds.

Return Values

Returns the index of the filter control being modified.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.

See Also

HDN_FILTERCHANGE

HDM_SETHOTDIVIDER

Changes the color of a divider between header items to indicate the destination of an external drag-and-drop operation. You can send this message explicitly or use the **Header_SetHotDivider** macro.

```
HDM_SETHOTDIVIDER
    wParam = (WPARAM) flag;
    lParam = (LPARAM) dwInputValue;
```

Parameters

flag

Value specifying the type of value represented by *dwInputValue*. This value can be one of the following:

TRUE	Indicates that *dwInputValue* holds the client coordinates of the pointer.
FALSE	Indicates that *dwInputValue* holds a divider index value.

dwInputValue

Value held in *dwInputValue* is interpreted depending on the value of *flag*.

If *flag* is TRUE, *dwInputValue* represents the x- and y-coordinates of the pointer. The x-coordinate is in the low word, and the y-coordinate is in the high word. When the header control receives the message, it highlights the appropriate divider based on the *dwInputValue* coordinates.

If *flag* is FALSE, *dwInputValue* represents the integer index of the divider to be highlighted.

Return Values

Returns a value equal to the index of the divider that the control highlighted.

Remarks

This message creates an effect that a header control automatically produces when it has the **HDS_DRAGDROP** style. The HDM_SETHOTDIVIDER message is intended to be used when the owner of the control handles drag-and-drop operations manually.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

HDM_SETIMAGELIST

Assigns an image list to an existing header control. You can send this message explicitly or use the **Header_SetImageList** macro.

```
HDM_SETIMAGELIST
    wParam = 0;
    lParam = (LPARAM) himl;
```

Parameters

himl
Handle to an image list.

Return Values

Returns the handle to the image list previously associated with the control. Returns NULL upon failure or if no image list was set previously.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).

Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

HDM_SETITEM

Sets the attributes of the specified item in a header control. You can send this message explicitly or use the **Header_SetItem** macro.

```
HDM_SETITEM
    wParam = (WPARAM)(int) iIndex;
    lParam = (LPARAM)(const LPHDITEM) phdItem;
```

Parameters

iIndex
Current index of the item whose attributes are to be changed.

phdItem
Address of an **HDITEM** structure that contains item information. When this message is sent, the **mask** member of the structure must be set to indicate which attributes are being set.

Return Values

Returns nonzero upon success, or zero otherwise.

Remarks

The **HDITEM** structure that supports this message supports item order and image list information. By using these members, you can control the order in which items are displayed and specify images to appear with items.

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 commctrl.h.

HDM_SETORDERARRAY

Sets the left-to-right order of header items. You can send this message explicitly or use the **Header_SetOrderArray** macro.

```
HDM_SETORDERARRAY
    wParam = (WPARAM) (int) iSize;
    lParam = (LPARAM) lpiArray;
```

Parameters

iSize

Size of the buffer at *lpiArray*, in elements. This value must equal the value returned by **HDM_GETITEMCOUNT**.

lpiArray

Address of an array that specifies the order in which items should be displayed, from left to right. For example, if the contents of the array are {2,0,1}, the control displays item 2, item 0, and item 1, from left to right.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

HDM_SETUNICODEFORMAT

Sets the UNICODE character format flag for the control. This message allows you to change the character set used by the control at run time rather than having to re-create the control. You can send this message explicitly or use the **Header_SetUnicodeFormat** macro.

```
HDM_SETUNICODEFORMAT
    wParam = (WPARAM)(BOOL)fUnicode;
    lParam = 0;
```

Parameters

fUnicode

Determines the character set that is used by the control. If this value is nonzero, the control will use UNICODE characters. If this value is zero, the control will use ANSI characters.

Return Values

Returns the previous UNICODE format flag for the control.

Remarks

See the remarks for **CCM_SETUNICODEFORMAT** for a discussion of this message.

Version 4.00 and later of Comctl32.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 commctrl.h.

HDM_GETUNICODEFORMAT

Header Control Macros

HDM_SETUNICODEFORMAT

Sets the UNICODE character format flag for the control. This message allows you to change the character set used by the control at run time rather than having to re-create the control. You can send this message explicitly or use the **Header_SetUnicodeFormat** macro.

```
HDM_SETUNICODEFORMAT
    wParam = (WPARAM)(BOOL)fUnicode;
    lParam = 0;
```

Parameters

fUnicode

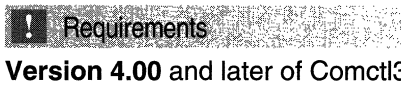
Determines the character set that is used by the control. If this value is nonzero, the control will use UNICODE characters. If this value is zero, the control will use ANSI characters.

Return Values

Returns the previous UNICODE format flag for the control.

Remarks

See the remarks for **CCM_SETUNICODEFORMAT** for a discussion of this message.



Version 4.00 and later of Comctl32.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 commctrl.h.

HDM_GETUNICODEFORMAT

Header_ClearFilter

Clears the filter for a given header control. You can use this macro or send the **HDM_CLEARFILTER** message explicitly.

```
int Header_ClearFilter(
  hwnd,
  i
);
```

Parameters

hwnd
Handle to the header control.

i
Value specifying the column of the filter to be cleared. Specifying –1 will clear all of the filters.

Remarks

If the column value is specified as –1, all the filters will be cleared and the **HDN_FILTERCHANGE** notification will be sent only once.

Version 5.80 and later of Comctl32.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 commctrl.h.

Header_ClearAllFilters

Header_CreateDragImage

Creates a transparent version of an item image within an existing header control. You can use this macro or send the **HDM_CREATEDRAGIMAGE** message explicitly.

```
HIMAGELIST Header_CreateDragImage(
  HWND hwndHD,
  int iIndex
);
```

Parameters

hwndHD
Handle to a header control.

iIndex
Zero-based index of the item within the header control. The image assigned to this item is used as the basis for the transparent image.

Return Values

Returns a handle to an image list that contains the new image as its only element.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

Header_DeleteItem

Deletes an item from a header control. You can use this macro or send the **HDM_DELETEITEM** message explicitly.

```
BOOL Header_DeleteItem(
  hwndHD,
  index
);
```

Parameters

hwndHD
Handle to the header control.

index
Index of the item to delete.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

The **Header_DeleteItem** macro is defined as follows:

```
#define Header_DeleteItem(hwndHD, index)      \
    (BOOL)SendMessage((hwndHD), HDM_DELETEITEM, (WPARAM)(int)(index), 0L)
```

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 commctrl.h.

Header_EditFilter

Starts editing the specified filter control.

```
int Header_EditFilter(
  hwnd,
  i,
  fDiscardChanges
);
```

Parameters

hwnd
Handle to the header control.

i
Value specifying the column to edit.

fDiscardChanges
Flag that specifies how to handle the user's editing changes. Use this flag to specify what to do if the user is in the process of editing the filter when the message is sent.

TRUE
Discard the changes made by the user.

FALSE
Accept the changes made by the user.

Return Values

Returns the index of the filter control being edited.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.

See Also

HDM_EDITFILTER

Header_GetBitmapMargin

Retrieves the width of the margin (in pixels) of a bitmap in an existing header control. You can use this macro or send the **HDM_GETBITMAPMARGIN** message explicitly.

```
Header_GetBitmapMargin(
  hwnd
);
```

Parameters

hwnd
Handle to a header control.

Return Values

Returns the width of the bitmap margin in pixels. If the bitmap margin was not previously specified, the default value of 3***GetSystemMetrics(**CX_EDGE) is returned.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.

See Also

Header_SetBitmapMargin

Header_GetImageList

Retrieves the handle to the image list that has been set for an existing header control. You can use this macro or send the **HDM_GETIMAGELIST** message explicitly.

```
HIMAGELIST Header_GetImageList(HWND hwndHD);
```

Parameters

hwndHD
Handle to a header control.

Return Values

Returns the handle to the image list that is set for the header control.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

Header_GetItem

Retrieves information about an item in a header control. You can use this macro or send the **HDM_GETITEM** message explicitly.

```
BOOL Header_GetItem(
  HWND hwndHD,
  int index,
  LPHDITEM phdi
);
```

Parameters

hwndHD
Handle to the header control.

index
Index of the item for which information is to be retrieved.

phdi
Address of an **HDITEM** structure. When the message is sent, the **mask** member indicates the type of information being requested. When the message returns, the

other members receive the requested information. If the **mask** member specifies zero, the message returns TRUE but copies no information to the structure.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

If the HDI_TEXT flag is set in the **mask** member of the **HDITEM** structure, the control may change the **pszText** member of the structure to point to the new text instead of filling the buffer with the requested text. Applications should not assume that the text will always be placed in the requested buffer.

The **Header_GetItem** macro is defined as follows:

```
#define Header_GetItem(hwndHD, index, phdi)     \
    (BOOL)SendMessage((hwndHD), HDM_GETITEM,   \
    (WPARAM)(int)(index), (LPARAM)(LPHDITEM)(phdi))
```

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 commctrl.h.

Header_GetItemCount

Retrieves a count of the items in a header control. You can use this macro or send the **HDM_GETITEMCOUNT** message explicitly.

```
int Header_GetItemCount(
  hwndHD
);
```

Parameters

hwndHD
Handle to the header control.

Return Values

Returns the number of items if successful, or –1 otherwise.

Remarks

The **Header_GetItemCount** macro is defined as follows:

```
#define Header_GetItemCount(hwndHD)   \
        (int)SendMessage((hwndHD), HDM_GETITEMCOUNT, 0, 0L)
```

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 commctrl.h.

Header_GetItemRect

Retrieves the bounding rectangle for a given item in a header control. You can use this macro or send the **HDM_GETITEMRECT** message explicitly.

```
BOOL Header_GetItemRect(
  HWND hwndHD,
  int iIndex,
  LPRECT lpItemRect
);
```

Parameters

hwndHD
Handle to a header control.

iIndex
Zero-based index of the header control item for which to retrieve the bounding rectangle.

lpItemRect
Address of a **RECT** structure that receives the bounding rectangle information.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

Header_GetOrderArray

Retrieves the current left-to-right order of items in a header control. You can use this macro or send the **HDM_GETORDERARRAY** message explicitly.

```
BOOL Header_GetOrderArray(
  HWND hwndHD,
  int iSize,
  int *lpiArray
);
```

Parameters

hwndHD
Handle to a header control.

iSize
Number of integer elements that *lpiArray* can hold. This value must be equal to or greater than the number of items in the control (see **HDM_GETITEMCOUNT**).

lpiArray
Address of an array of integers that receive the index values for items in the header. The number of elements in this array is specified in *iSize* and must be equal to or greater than the number of items in the control. For example, the following code fragment will reserve enough memory to hold the index values:

```
int iItems,
    *lpiArray;

// Get memory for buffer
if((iItems = SendMessage(hwndHD, HDM_GETITEMCOUNT, 0,0))!=-1)
    if(!(lpiArray = calloc(iItems,sizeof(int))))
MessageBox(hwnd, "Out of memory.","Error", MB_OK);
```

Return Values

Returns nonzero if successful, and the buffer at *lpiArray* receives the item number of each item in the header control in the order in which they appear from left to right. Returns zero otherwise.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

Header_GetUnicodeFormat

Retrieves the UNICODE character format flag for the control. You can use this macro or send the **HDM_GETUNICODEFORMAT** message explicitly.

```
BOOL Header_GetUnicodeFormat(
  HWND hwnd
);
```

Parameters

hwnd
Handle to the control.

Return Values

Returns the UNICODE format flag for the control. If this value is nonzero, the control is using UNICODE characters. If this value is zero, the control is using ANSI characters.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

See Also

Header_SetUnicodeFormat

Header_InsertItem

Inserts a new item into a header control. You can use this macro or send the **HDM_INSERTITEM** message explicitly.

```
int Header_InsertItem(
  hwndHD,
  index,
  phdi
);
```

Parameters

hwndHD
Handle to the header control.

index
Index of the item after which the new item is to be inserted. The new item is inserted at the end of the header control if *index* is greater than or equal to the number of items in the control. If *index* is zero, the new item is inserted at the beginning of the header control.

phdi
Address of an **HDITEM** structure that contains information about the new item.

Return Values
Returns the index of the new item if successful, or –1 otherwise.

Remarks
The **Header_InsertItem** macro is defined as follows:

```
#define Header_InsertItem(hwndHD, index, phdi) \
    (int)SendMessage((hwndHD), HDM_INSERTITEM, (WPARAM)(int)(index), \
    (LPARAM)(const LPHDITEM)(phdi))
```

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 commctrl.h.

Header_Layout

Retrieves the correct size and position of a header control within the parent window. You can use this macro or send the **HDM_LAYOUT** message explicitly.

```
BOOL Header_Layout(
  hwndHD,
  playout
);
```

Parameters
hwndHD
Handle to the header control.

playout
Address of an **HDLAYOUT** structure. The **prc** member specifies the coordinates of a rectangle, and the **pwpos** member receives the size and position for the header control within the rectangle.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

The **Header_Layout** macro is defined as follows:

```
#define Header_Layout(hwndHD, playout) \
    (BOOL)SendMessage((hwndHD), HDM_LAYOUT, 0, \
    (LPARAM)(PLHDLAYOUT)(playout))
```

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 commctrl.h.

Header_OrderToIndex

Retrieves an index value for an item based on its order in the header control. You can use this macro or send the **HDM_ORDERTOINDEX** message explicitly.

```
int Header_OrderToIndex(
  HWND hwndHD,
  int iOrder
);
```

Parameters

hwndHD
Handle to a header control.

iOrder
Order that the item appears within the header control, from left to right. The index value of the item in the far left column would be 0, the next item to the right would be 1, and so on.

Return Values

Returns an INT that specifies the index of the item. If *iOrder* is invalid (negative or too large), the return equals *iOrder*.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

Header_SetBitmapMargin

Sets the width of the margin for a bitmap in an existing header control. You can use this macro or send the **HDM_SETBITMAPMARGIN** message explicitly.

```
int Header_SetBitmapMargin(
  hwnd,
  iWidth
);
```

Parameters

hwnd
Handle to a header control.

iWidth
Width, specified in pixels, of the margin that surrounds a bitmap within an existing header control.

Return Values

Returns width of the bitmap margin in pixels. If the bitmap margin was not previously specified, the default value of 3*GetSystemMetrics(CX_EDGE) is returned.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.

See Also

Header_GetBitmapMargin

Header_SetFilterChangeTimeout

Sets the timeout interval between the time a change takes place in the filter attributes and the posting of an **HDN_FILTERCHANGED** notification. You can use this macro or send the **HDM_SETFILTERCHANGETIMEOUT** message explicitly.

```
int Header_SetFilterChangeTimeout(
  hwnd,
  i
);
```

Parameters

hwnd
Handle to the header control.

i
Timeout value, in milliseconds.

Return Values

Returns the index of the filter control being modified.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.

See Also

HDN_FILTERCHANGE, HDM_SETFILTERCHANGETIMEOUT

Header_SetHotDivider

Changes the color of a divider between header items to indicate the destination of an external drag-and-drop operation. You can use this macro or send the **HDM_SETHOTDIVIDER** message explicitly.

```
int Header_SetHotDivider(
  HWND hwndHD,
  BOOL flag,
  DWORD dwInputValue
);
```

Parameters

hwndHD
Handle to a header control.

flag
Value specifying how *dwInputValue* is to be interpreted. The value in this field can be one of the following:

TRUE	Indicates that *dwInputValue* holds client coordinates of the pointer.
FALSE	Indicates that *dwInputValue* holds a divider index value.

dwInputValue
Value held here is interpreted depending on the value of *flag*.

If *flag* is TRUE, *dwInputValue* represents the x- and y- client coordinates of the pointer. The x-coordinate is in the low word, and the y-coordinate is in the high word. Upon receiving the message, the header control highlights the appropriate divider based on the *dwInputValue* coordinates.

If *flag* is FALSE, *dwInputValue* represents the integer index of the divider that will be highlighted.

Return Values

Returns the index of the divider that the control highlighted.

Remarks

A header control set to the **HDS_DRAGDROP** style produces this effect automatically. This message is intended to be used when the owner of the control handles drag-and-drop operations manually.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

Header_SetImageList

Assigns an image list to an existing header control. You can use this macro or send the **HDM_SETIMAGELIST** message explicitly.

```
HIMAGELIST Header_SetImageList(
  HWND hwndHD,
  HIMAGELIST himl
);
```

Parameters

hwndHD
Handle to a header control.

himl
Handle to an image list.

Return Values

Returns the handle to the image list previously assigned to the header control, or NULL if there is no previous image list.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

Header_SetItem

Sets the attributes of the specified item in a header control. You can use this macro or send the **HDM_SETITEM** message explicitly.

```
BOOL Header_SetItem(
  hwndHD,
  iIndex;
  phdItem
);
```

Parameters

hwndHD
Handle to a header control.

iIndex
Current index of the item whose attributes are to be changed.

phdItem
Address of an **HDITEM** structure that contains item information. When this message is sent, the **mask** member of the structure must be set to indicate which attributes are being set.

Return Values
Returns nonzero if successful, or zero otherwise.

Remarks
The **HDITEM** structure that supports this macro supports item order and image list information. By using these members, you can control the order in which items are displayed and specify images to appear with items.

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 commctrl.h.

Header_SetOrderArray

Sets the left-to-right order of header items. You can use this macro or send the **HDM_SETORDERARRAY** message explicitly.

```
BOOL Header_SetOrderArray(
  HWND hwndHD,
  int iSize,
  int *lpiArray
);
```

Parameters
hwndHD
Handle to a header control.

iSize
Size of the buffer at *lpiArray*, in elements. This value must equal the value returned by **HDM_GETITEMCOUNT**.

lpiArray
Address of an array that specifies the order in which items should be displayed, from left to right. For example, if the contents of the array are {2,0,1}, the control displays item 2, item 0, and item 1, from left to right.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

Header_SetUnicodeFormat

Sets the UNICODE character format flag for the control. This message allows you to change the character set used by the control at run time rather than having to re-create the control. You can use this macro or send the **HDM_SETUNICODEFORMAT** message explicitly.

```
BOOL Header_SetUnicodeFormat(
  HWND hwnd,
  BOOL fUnicode
);
```

Parameters

hwnd
Handle to the control.

fUnicode
Determines the character set that is used by the control. If this value is nonzero, the control will use UNICODE characters. If this value is zero, the control will use ANSI characters.

Return Values

Returns the previous UNICODE format flag for the control.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

See Also

Header_GetUnicodeFormat

Header Control Notification Messages

HDN_BEGINDRAG

Sent by a header control when a drag operation has begun on one of its items. This notification message is sent only by header controls that are set to the **HDS_DRAGDROP** style. This notification is sent in the form of a **WM_NOTIFY** message.

```
HDN_BEGINDRAG
   pNMHeader = (LPNMHEADER) lParam;
```

Parameters

pNMHeader

Address of an **NMHEADER** structure containing information about the header item that is being dragged.

Return Values

To allow the header control to automatically manage drag-and-drop operations, return FALSE. If the owner of the control is manually performing drag-and-drop reordering, return TRUE.

Remarks

A header control defaults to automatically managing drag-and-drop reordering. Returning TRUE to indicate external (manual) drag-and-drop management allows the owner of the control to provide custom services as part of the drag-and-drop process.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

HDN_BEGINTRACK

Notifies a header control's parent window that the user has begun dragging a divider in the control (that is, the user has pressed the left mouse button while the mouse cursor is on a divider in the header control). This notification message is sent in the form of a **WM_NOTIFY** message.

```
HDN_BEGINTRACK
    phdn = (LPNMHEADER) lParam;
```

Parameters

phdn

Address of an **NMHEADER** structure that contains information about the header control and the item whose divider is to be dragged.

Return Values

Returns FALSE to allow tracking of the divider, or TRUE to prevent tracking.

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 commctrl.h.

HDN_DIVIDERDBLCLICK

Notifies a header control's parent window that the user double-clicked the divider area of the control. This notification message is sent in the form of a **WM_NOTIFY** message.

```
HDN_DIVIDERDBLCLICK
    phdn = (LPNMHEADER) lParam;
```

Parameters

phdn

Address of an **NMHEADER** structure that contains information about the header control and the item whose divider was double-clicked.

Return Values

No return value.

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 commctrl.h.

HDN_ENDDRAG

Sent by a header control when a drag operation has ended on one of its items. This notification is sent as a **WM_NOTIFY** message. Only header controls that are set to the **HDS_DRAGDROP** style send this notification.

```
HDN_ENDDRAG
    pNMHeader = (LPNMHEADER) lParam;
```

Parameters

pNMHeader
Address of an **NMHEADER** structure containing information about the header item that was being dragged.

Return Values

To allow the control to automatically place and reorder the item, return FALSE. To prevent the item from being placed, return TRUE.

Remarks

If the owner is performing external (manual) drag-and-drop management, it must return FALSE. The owner then must reorder header items manually by sending **HDM_SETITEM** or **HDM_SETORDERARRAY**.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

HDN_ENDTRACK

Notifies a header control's parent window that the user has finished dragging a divider. This notification message sent in the form of a **WM_NOTIFY** message.

```
HDN_ENDTRACK
    phdn = (LPNMHEADER) lParam;
```

Parameters

phdn

Address of an **NMHEADER** structure that contains information about the header control and the item whose divider was dragged.

Return Values

No return value.

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 commctrl.h.

HDN_FILTERBTNCLICK

Notifies the header control's parent window when the filter button is clicked or in response to an **HDM_SETITEM** message.

```
HDN_FILTERBTNCLICK
    phdr = (LPNMHDFILTERBTNCLICK) lParam;
```

Parameters

phdr

Address of an **NMHDFILTERBTNCLICK** structure that contains information about the header control and the header filter button.

Return Values

If you return TRUE, an **HDN_FILTERCHANGED** notification will be sent to the header control's parent window. This notification gives the parent window an opportunity to synchronize its user interface elements. Return FALSE if you don't want the notification sent.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.

See Also

NMHDFILTERBTNCLICK

HDN_FILTERCHANGE

Notifies the header control's parent window that the attributes of a header control filter are being changed or edited.

```
HDN_FILTERCHANGE
    phdr = (LPNMHEADER) lParam;
```

Parameters

phdr

Address of an **NMHEADER** structure that contains information about the header control and the header item, including the attributes that are about to change.

Return Values

No return value.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.

See Also

HDM_SETFILTERCHANGETIMEOUT

HDN_GETDISPINFO

Sent to the owner of a header control when the control needs information about a callback header item. This notification is sent as a **WM_NOTIFY** message.

```
HDN_GETDISPINFO
    pDispInfo = (LPNMHDDISPINFO) lParam;
```

Parameters

pDispInfo

Address of an **NMHDDISPINFO** structure. On input, the fields of the structure specify what information is required and the item of interest.

Remarks

Fill the appropriate members of the structure to return the requested information to the header control. If your message handler sets the **mask** member of the **NMHDDISPINFO** structure to HDI_DI_SETITEM, the header control stores the information and will not request it again.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

HDN_ITEMCHANGED

Notifies a header control's parent window that the attributes of a header item have changed. This notification message is sent in the form of a **WM_NOTIFY** message.

```
HDN_ITEMCHANGED
    phdr = (LPNMHEADER) lParam;
```

Parameters

phdr

Address of an **NMHEADER** structure that contains information about the header control, including the attributes that have changed.

Return Values

No return value.

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 commctrl.h.

HDN_ITEMCHANGING

Notifies a header control's parent window that the attributes of a header item are about to change. This notification message is sent in the form of a **WM_NOTIFY** message.

```
HDN_ITEMCHANGING
    phdr = (LPNMHEADER) lParam;
```

Parameters

phdr

Address of an **NMHEADER** structure that contains information about the header control and the header item, including the attributes that are about to change.

Return Values

Returns FALSE to allow the changes, or TRUE to prevent them.

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 commctrl.h.

HDN_ITEMCLICK

Notifies a header control's parent window that the user clicked the control. This notification message is sent in the form of a **WM_NOTIFY** message.

```
HDN_ITEMCLICK
    phdr = (LPNMHEADER) lParam;
```

Parameters

phdr

Address of an **NMHEADER** structure that identifies the header control and specifies the index of the header item that was clicked and the mouse button used to click the item. The **pItem** member is set to NULL.

Return Values

No return value.

Remarks

A header control sends this notification message after the user releases the left mouse button.

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 commctrl.h.

HDN_ITEMDBLCLICK

Notifies a header control's parent window that the user double-clicked the control. This notification message is sent in the form of a **WM_NOTIFY** message. Only header controls that are set to the HDS_BUTTONS style send this notification.

```
HDN_ITEMDBLCLICK
    pnmhdr = (LPNMHEADER) lParam;
```

Parameters

pnmhdr
Address of an **NMHEADER** structure that contains information about this notification.

Return Values

No return value.

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 commctrl.h.

HDN_TRACK

Notifies a header control's parent window that the user is dragging a divider in the header control. This notification message is sent in the form of a **WM_NOTIFY** message.

```
HDN_TRACK
    phdr = (LPNMHEADER) lParam;
```

Parameters

phdr
Address of an **NMHEADER** structure that contains information about the header control and the item whose divider is being dragged.

Return Values

Returns FALSE to continue tracking the divider, or TRUE to end tracking.

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 commctrl.h.

NM_CUSTOMDRAW (header)

Sent by a header control to notify its parent window about drawing operations. This notification is sent in the form of a **WM_NOTIFY** message.

```
NM_CUSTOMDRAW
    lpNMCustomDraw = (LPNMCUSTOMDRAW) lParam;
```

Parameters

lpNMCustomDraw
Address of an **NMCUSTOMDRAW** structure that contains information about the drawing operation. The **dwItemSpec** member of this structure contains the index of the item being drawn and the **lItemlParam** member of this structure contains the item's lParam.

Return Values

The value your application can return depends on the current drawing stage. The **dwDrawStage** member of the associated **NMCUSTOMDRAW** structure holds a value that specifies the drawing stage. You must return one of the following values.

When **dwDrawStage** equals CDDS_PREPAINT:

CDRF_DODEFAULT
The control will draw itself. It will not send any additional NM_CUSTOMDRAW messages for this paint cycle.

CDRF_NOTIFYITEMDRAW
The control will notify the parent of any item-related drawing operations. It will send NM_CUSTOMDRAW notification messages before and after drawing items.

CDRF_NOTIFYITEMERASE
The control will notify the parent when an item will be erased. It will send NM_CUSTOMDRAW notification messages before and after erasing items.

CDRF_NOTIFYPOSTERASE
The control will notify the parent after erasing an item.

CDRF_NOTIFYPOSTPAINT
The control will notify the parent after painting an item.

CDRF_NOTIFYSUBITEMDRAW
Version 4.71. The control will notify the parent when a list view sub-item is being drawn.

When **dwDrawStage** equals CDDS_ITEMPREPAINT:

CDRF_NEWFONT
Your application specified a new font for the item; the control will use the new font. For more information on changing fonts, see *Changing Fonts and Colors*.

CDRF_SKIPDEFAULT
Your application drew the item manually. The control will not draw the item.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.02 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

See Also

Using Custom Draw

NM_RCLICK (header)

Notifies a tree view control's parent window that the user has clicked the right mouse button within the control. This notification is sent in the form of a **WM_NOTIFY** message.

```
NM_RCLICK
    lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh
Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

Return nonzero to not allow the default processing, or zero to allow the default processing.

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 commctrl.h.

NM_RELEASEDCAPTURE (header)

Notifies a header control's parent window that the control is releasing mouse capture. This notification is sent in the form of a **WM_NOTIFY** message.

```
NM_RELEASEDCAPTURE
   lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh
Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The control ignores the return value from this notification.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

Header Control Structures

HDHITTESTINFO

Contains information about a hit test. This structure is used with the **HDM_HITTEST** message. This structure supersedes the **HD_HITTESTINFO** structure.

```
typedef struct _HD_HITTESTINFO {
    POINT pt;
    UINT flags;
    int iItem;
} HDHITTESTINFO, FAR *LPHDHITTESTINFO;
```

Members

pt

POINT structure that contains the point to be hit test, in client coordinates.

flags

Variable that receives information about the results of a hit test. This member can be one or more of the following values:

HHT_ABOVE	The point is above the header control's bounding rectangle.
HHT_BELOW	The point is below the header control's bounding rectangle.
HHT_NOWHERE	The point is inside the header control's bounding rectangle but is not over a header item.
HHT_ONDIVIDER	The point is on the divider between two header items.
HHT_ONDIVOPEN	The point is on the divider of an item that has a width of zero. Dragging the divider reveals the item instead of resizing the item to the left of the divider.
HHT_ONHEADER	The point is inside the header control's bounding rectangle.
HHT_ONFILTER	**Version 5.80** The point is over the filter area.
HHT_ONFILTERBUTTON	**Version 5.80** The point is on the filter button.
HHT_TOLEFT	The point is to the left of the header control's bounding rectangle.
HHT_TORIGHT	The point is to the right of the header control's bounding rectangle.

Two of these values can be combined, such as when the position is above and to the left of the client area.

iItem

If the hit test is successful, contains the index of the item at the hit test point.

Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

HDITEM

Contains information about an item in a header control. This structure supersedes the **HD_ITEM** structure.

```
typedef struct _HDITEM {
    UINT    mask;
    int     cxy;
    LPTSTR  pszText;
    HBITMAP hbm;
    int     cchTextMax;
    int     fmt;
    LPARAM  lParam;
#if (_WIN32_IE >= 0x0300)
    int     iImage;
    int     iOrder;
#endif
#if (_WIN32_IE >= 0x0500)
    UINT    type;
    LPVOID  pvFilter;
#endif
} HDITEM, FAR * LPHDITEM;
```

Members

mask

Flags indicating which other structure members contain valid data or must be filled in. This member can be a combination of the following values:

Flag	Description
HDI_BITMAP	The **hbm** member is valid.
HDI_FORMAT	The **fmt** member is valid.
HDI_FILTER	Version 5.80. The **type** and **pvFilter** members are valid. This is used to filter out the values specified in the **type** member.
HDI_HEIGHT	The **cxy** member is valid and specifies the item's height.
HDI_IMAGE	Version 4.70. The **iImage** member is valid and specifies the image to be displayed with the item.
HDI_LPARAM	The **lParam** member is valid.
HDI_ORDER	Version 4.70. The **iOrder** member is valid and specifies the item's order value.
HDI_TEXT	The **pszText** and **cchTextMax** members are valid.
HDI_WIDTH	The **cxy** member is valid and specifies the item's width.

cxy
Width or height of the item.

pszText
Address of an item string. If the text is being retrieved from the control, this member must be initialized to point to a character buffer. If this member is set to LPSTR_TEXTCALLBACK, the control will request text information for this item by sending an **HDN_GETDISPINFO** notification message.

hbm
Handle to the item bitmap.

cchTextMax
Length of the item string, in characters. If the text is being retrieved from the control, this member must contain the number of characters at the address specified by **pszText**.

fmt
Flags that specify the item's format.

Set one of the following flags to specify text justification:

Flag	Description
HDF_CENTER	The item's contents are centered.
HDF_LEFT	The item's contents are left-aligned.
HDF_RIGHT	The item's contents are right-aligned.

Set one of the following flags to control the display:

Flag	Description
HDF_BITMAP	The item displays a bitmap.
HDF_BITMAP_ON_RIGHT	Version 4.70. The bitmap appears to the right of text. This does not affect an image from an image list assigned to the header item.
HDF_OWNERDRAW	The header control's owner draws the item.
HDF_STRING	The item displays a string.

The preceding value can be combined with:

Flag	Description
HDF_IMAGE	Version 4.70. Display an image from an image list. Specify the image list by sending an **HDM_SETIMAGELIST** message. Specify the index of the image in the **iImage** member of this structure.
HDF_JUSTIFYMASK	Isolate the bits corresponding to the three justification flags listed in the preceding table.

Flag	Description
HDF_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, header text is read in same direction as the text in its parent window. If HDF_RTLREADING is set, header text will read in the opposite direction from the text in the parent window.

lParam
Application-defined item data.

iImage
Version 4.70. Zero-based index of an image within the image list. The specified image will be displayed in the header item in addition to any image specified in the **hbm** field. If **iImage** is set to I_IMAGECALLBACK, the control requests text information for this item by using an **HDN_GETDISPINFO** notification message.

iOrder
Version 4.70. Order in which the item appears within the header control, from left to right. That is, the value for the far left item is 0. The value for the next item to the right is 1, and so on.

The header control can display text, an image, and a bitmap for an item all at the same time. The alignment flags determine the order in which they appear. With HDF_LEFT or HDF_CENTER, the order is image, text, and then bitmap. With HDF_RIGHT the order is bitmap, image, and then text.

type
Version 5.80. Type of filter specified by **pvFilter**. The possible types include:

Flag	Description
HDFT_ISTRING	String data.
HDFT_ISNUMBER	Numerical data.
HDFT_HASNOVALUE	Ignore pvFilter.

pvFilter
Version 5.80. Address of an application-defined data item. The data filter type is determined by setting the flag value of the **type** member.

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

HDLAYOUT

Contains information used to set the size and position of a header control. **HDLAYOUT** is used with the **HDM_LAYOUT** message. This structure supersedes the **HD_LAYOUT** structure.

```
typedef struct _HD_LAYOUT {
    RECT FAR*    prc;
    WINDOWPOS FAR* pwpos;
} HDLAYOUT, FAR *LPHDLAYOUT;
```

Members

prc
Address of a **RECT** structure that contains the coordinates of a rectangle that the header control will occupy.

pwpos
Address of a **WINDOWPOS** structure that receives information about the appropriate size and position of the header control.

Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

HDTEXTFILTER Structure

Contains information about header control text filters.

```
typedef struct _HDTEXTFILTER {
    LPTSTR pszText;
    INT cchTextMax;
} HDTEXTFILTER, FAR * LPHD_TEXTFILTER;
```

Members

pszText
[in] Address of the buffer containing the filter.

cchTextMax
[in] Value specifying the maximum size, in characters, for an edit control buffer.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.

NMHDDISPINFO

Contains information used in handling **HDN_GETDISPINFO** notification messages.

```
typedef struct tagNMHDDISPINFO {
    NMHDR   hdr;
    int     iItem;
    UINT    mask;
    LPTSTR  pszText;
    int     cchTextMax;
    int     iImage;
    LPARAM  lParam;
} NMHDDISPINFO, FAR* LPNMHDDISPINFO;
```

Members

hdr
NMHDR structure containing information about this notification message.

iItem
Zero-based index of the item in the header control.

mask
Set of bit flags specifying which members of the structure must be filled in by the owner of the header control. This value can be a combination of the following values:

HDI_TEXT	The **pszText** field must be filled in.
HDI_IMAGE	Version 4.70. The **iImage** field must be filled in.
HDI_LPARAM	The **lParam** field must be filled in.
HDI_DI_SETITEM	Version 4.70. A return value. Indicates that the header control should store the item information and not ask for it again.

pszText
Address of a null-terminated string containing the text that will be displayed for the header item.

cchTextMax
Size of the buffer that **pszText** points to.

iImage
Zero-based index of an image within the image list. The specified image will be displayed with the header item, but it does not take the place of the item's bitmap. If

iImage is set to I_IMAGECALLBACK, the control requests image information for this item by using an **HDN_GETDISPINFO** notification message.

lParam
32-bit value to associate with the item.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

NMHDFILTERBTNCLICK Structure

Specifies or receives the attributes of a filter button click.

```
typedef struct tagNMHDFILTERBTNCLICK {
    NMHDR hdr;
    INT iItem;
    RECT rc;
} NMHDFILTERBTNCLICK, FAR* LPNMHDFILTERBTNCLICK;
```

Members

hdr
Handle of an **NMHDR** structure that contains additional information.

iItem
Zero-based index of the control to which this structure refers.

rc
Address of a **RECT** structure that contains the client rectangle for the filter button.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.

See Also

HDN_FILTERBTNCLICK

NMHEADER

Contains information about header control notification messages. This structure supersedes the **HD_NOTIFY** structure.

```
typedef struct tagNMHEADER{
    NMHDR    hdr;
    int      iItem;
    int      iButton;
    HDITEM  FAR* pitem;
} NMHEADER, FAR* LPNMHEADER;
```

Members

hdr

NMHDR structure that contains information about the notification message.

iItem

Zero-based index of the header item that is the focus of the notification message.

iButton

Value specifying the index of the mouse button used to generate the notification message. This member can be one of the following values:

0	Left button
1	Right button
2	Middle button

pitem

Optional pointer to an **HDITEM** structure containing information about the item specified by **iItem**. The **mask** member of the **HDITEM** structure indicates which of its members are valid.

Remarks

While most header control notifications pass a pointer to an **NMHEADER** structure, only some of them use the **pItem** member to pass an **HDITEM** structure. Those that do use **pItem** may not provide complete information about the item. To obtain more information about an item, use **HDM_GETITEM**.

Requirements

Version 4.00 and later of Comctl32.dll.

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

CHAPTER 16

Hot-Key Controls

A *hot-key control* is a window that enables the user to enter a combination of keystrokes to be used as a *hot-key*. A *hot key* is a key combination that the user can press to perform an action quickly. For example, a user can create a hot key that activates a given window and brings it to the top of the z-order. The hot-key control displays the user's choices and ensures that the user selects a valid key combination.

Using Hot-Key Controls

When the user enters a key combination to be used as a hot key, the names of the keys appear in the hot-key control. A key combination can consist of a modifier key (such as CTRL, ALT, or SHIFT) and an accompanying key (such as a character key, an arrow key, a function key, and so on).

After the user has chosen a key combination, the application retrieves the key combination from the hot-key control and uses it to set up a hot key in the system. The information retrieved from the hot-key control includes a flag indicating the modifier key and the virtual key code of the accompanying key.

The application can use the information provided by a hot-key control to set up a global hot key or a thread-specific hot key. A global hot key is associated with a particular window; it allows the user to activate the window from any part of the system. An application sets a global hot key by using the **WM_SETHOTKEY** message. Whenever the user presses a global hot key, the window specified in **WM_SETHOTKEY** receives a **WM_SYSCOMMAND** message that specifies the SC_HOTKEY value. This message activates the window that receives it. The hot key remains valid until the application that called **WM_SETHOTKEY** exits.

A thread-specific hot key generates a **WM_HOTKEY** message that is posted to the beginning of a particular thread so that it is removed by the next iteration of the message loop. An application sets a thread-specific hot key by using the **RegisterHotKey** function.

Hot-Key Control Creation

You create a hot-key control by using the **CreateWindowEx** function, specifying the HOTKEY_CLASS window class. When the function returns a handle to the hot-key control, an application typically sets some rules about invalid hot-key combinations and can provide a default key combination. If an application does not set any rules, the user can choose any key or key combination as a hot key. Most applications, however, do not allow the user to use a common key (for example, the letter A) as a hot key.

The following function creates a hot-key control, uses the **HKM_SETRULES** and **HKM_SETHOTKEY** messages to initialize it, and returns a handle to the control. This hot-key control does not allow the user to choose a hot key that is a single unmodified key, nor does it permit the user to choose only SHIFT and a key. (These rules effectively prevent the user from choosing a hot key that might be entered accidentally while typing text.)

```
// InitializeHotkey - creates a hot-key control and sets
//    rules and default settings for it.
// Returns the handle of the hot-key control.
// hwndDlg - handle of the parent window (dialog box).
//
// Global variable
//     g_hinst - handle of the application instance.
extern HINSTANCE g_hinst;
HWND WINAPI InitializeHotkey(HWND hwndDlg)
{
    // Ensure that the common control DLL is loaded.
    InitCommonControls();

    hwndHot = CreateWindowEx(
        0,                          // no extended styles
        HOTKEY_CLASS,               // class name
        "",                         // no title (caption)
        WS_CHILD | WS_VISIBLE,      // style
        10, 10,                     // position
        200, 20,                    // size
        hwndDlg,                    // parent window
        NULL,                       // uses class menu
        g_hinst,                    // instance
        NULL                        // no WM_CREATE parameter
        );

    SetFocus(hwndHot);

    // Set rules for invalid key combinations. If the user
    // does not supply a modifier key, use ALT as a
    // modifier. If the user supplies SHIFT as a modifier
    // key, use SHIFT + ALT instead.
    SendMessage(hwndHot, HKM_SETRULES,
        (WPARAM) HKCOMB_NONE | HKCOMB_S, // invalid key
                                         // combinations
        MAKELPARAM(HOTKEYF_ALT, 0));     // add ALT to
                                         // invalid entries
```

```
    // Set CTRL + ALT + A as the default hot-key for this window.
    // 0x41 is the virtual key code for 'A'.
    SendMessage(hwndHot, HKM_SETHOTKEY,
        MAKEWORD(0x41, HOTKEYF_CONTROL | HOTKEYF_ALT), 0);

    return hwndHot;
}
```

Hot-Key Control Messages

After creating a hot-key control, an application interacts with it by using three messages: **HKM_SETRULES**, **HKM_SETHOTKEY**, and **HKM_GETHOTKEY**.

An application can send the **HKM_SETRULES** message to specify a set of CTRL, ALT, and SHIFT key combinations that are considered invalid hot keys. If the application specifies an invalid key combination, it should specify also a default modifier combination to use when the user selects the invalid combination. When the user enters the invalid combination, the system performs a logical OR operation on the invalid combination and the default combination. The result is considered a valid combination; it is converted to a string and displayed in the control.

The **HKM_SETHOTKEY** message allows an application to set the hot-key combination for a hot-key control. This message also is used typically when the hot-key control is created.

Applications use the **HKM_GETHOTKEY** message to retrieve the virtual key code and modifier flags of the hot key chosen by the user.

Hot-Key Control Notifications

The hot-key control does not send any notification messages via the **WM_NOTIFY** message. It will send, however, the **EN_CHANGE** notification via the **WM_COMMAND** message when the user changes the contents of the control.

Retrieving and Setting a Hot-Key

After the user has chosen a hot key, an application should retrieve it from the hot-key control by using the **HKM_GETHOTKEY** message. This message retrieves a 16-bit value that contains the virtual key code and modifier keys describing the hot key.

The following function retrieves a key combination from a hot-key control and then uses the **WM_SETHOTKEY** message to set a global hot key. Note that you cannot set a global hot key for a window that has the WS_CHILD window style:

```
// ProcessHotkey - retrieves the hot key from the hot-key
//     control and sets it as the hot key for the
//     application's main window.
// Returns TRUE if successful, or FALSE otherwise.
```

(continued)

(continued)

```
// hwndHot - handle of the hot-key control.
// hwndMain - handle of the main window.
BOOL WINAPI ProcessHotkey(HWND hwndHot, HWND hwndMain)
{
WORD wHotkey;
UINT iSetResult;

// Retrieve the hot key (virtual key code and modifiers).
wHotkey = SendMessage(hwndHot, HKM_GETHOTKEY, 0, 0);

// Use the result as wParam for WM_SETHOTKEY.
iSetResult = SendMessage(hwndMain, WM_SETHOTKEY, wHotkey, 0);

switch (iSetResult)
   {
   case 2:                 // WM_SETHOTKEY succeeded.
      MessageBox(NULL, "Hot key previously assigned",
         "Okay", MB_OK);
      return TRUE;

   case 1:                 // WM_SETHOTKEY succeeded.
      return TRUE;

   case 0:
      MessageBox(NULL, "Invalid window for hot key",
         "Error", MB_OK);
      return FALSE;

   case -1:
      MessageBox(NULL, "Invalid hot key",
         "Error", MB_OK);
      return FALSE;

   default:
      MessageBox(NULL, "Unknown error", "Error", MB_OK);
      return FALSE;
   }
}
```

Default Hot-Key Message Processing

This section describes the window messages handled by the window procedure for the predefined HOTKEY_CLASS window class used with hot-key controls.

Message	Processing performed
WM_CHAR	Retrieves the virtual key code.
WM_CREATE	Initializes the hot-key control, clears any hot-key rules, and uses the system font.
WM_ERASEBKGND	Hides the caret, calls the DefWindowProc function, and shows the caret again.
WM_GETDLGCODE	Returns a combination of the DLGC_WANTCHARS and DLGC_WANTARROWS values.
WM_GETFONT	Retrieves the font.
WM_KEYDOWN	Calls the DefWindowProc function if the key is ENTER, TAB, SPACE BAR, DEL, ESC, or BACKSPACE. If the key is SHIFT, CTRL, or ALT, it checks whether the combination is valid and, if it is, sets the hot key using the combination. All other keys are set as hot keys without their validity being checked first.
WM_KEYUP	Retrieves the virtual key code.
WM_KILLFOCUS	Destroys the caret.
WM_LBUTTONDOWN	Sets the focus to the window.
WM_NCCREATE	Sets the WS_EX_CLIENTEDGE window style.
WM_PAINT	Paints the hot-key control.
WM_SETFOCUS	Creates and shows the caret.
WM_SETFONT	Sets the font.
WM_SYSCHAR	Retrieves the virtual key code.
WM_SYSKEYDOWN	Calls the DefWindowProc function if the key is ENTER, TAB, SPACE BAR, DEL, ESC, or BACKSPACE. If the key is SHIFT, CTRL, or ALT, it checks whether the combination is valid and, if it is, sets the hot key using the combination. All other keys are set as hot keys without their validity being checked first.
WM_SYSKEYUP	Retrieves the virtual key code.

Hot-Key Control Reference

Hot-Key Control Messages

HKM_GETHOTKEY

Retrieves the virtual key code and modifier flags of a hot key from a hot-key control.

```
HKM_GETHOTKEY
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the virtual key code and modifier flags. 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

Remarks

The 16-bit value returned by this message can be used as the *wParam* parameter in the **WM_SETHOTKEY** 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 commctrl.h.

HKM_SETHOTKEY

Sets the hot key combination for a hot-key control.

```
HKM_SETHOTKEY
    wParam = MAKEWORD(bVKHotKey, bfMods);
    lParam = 0;
```

Parameters

bVKHotKey
Virtual key code of the hot key.

bfMods
Modifier flags indicating the keys that, when used in combination with *bVKHotKey*, define a hot-key combination. For a list of modifier flag values, see the description of the **HKM_GETHOTKEY** message.

Return Values

No return value.

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 commctrl.h.

See Also

WM_GETHOTKEY

HKM_SETRULES

Defines the invalid combinations and the default modifier combination for a hot-key control.

```
HKM_SETRULES
    wParam = (WPARAM) fwCombInv;
    lParam = MAKELPARAM(fwModInv, 0);
```

Parameters

fwCombInv

Array of flags that specify invalid key combinations. This parameter can be a combination of the following values:

HKCOMB_A	ALT
HKCOMB_C	CTRL
HKCOMB_CA	CTRL+ALT
HKCOMB_NONE	Unmodified keys
HKCOMB_S	SHIFT
HKCOMB_SA	SHIFT+ALT
HKCOMB_SC	SHIFT+CTRL
HKCOMB_SCA	SHIFT+CTRL+ALT

fwModInv

Array of flags that specify the key combination to use when the user enters an invalid combination. For a list of modifier flag values, see the description of the **HKM_GETHOTKEY** message.

Return Values

No return value.

Remarks

When a user enters an invalid key combination, as defined by flags specified in *fwCombInv*, the system uses the bitwise-OR operator to combine the keys entered by the user with the flags specified in *fwModInv*. The resulting key combination is converted into a string and then displayed in the hot-key control.

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 commctrl.h.

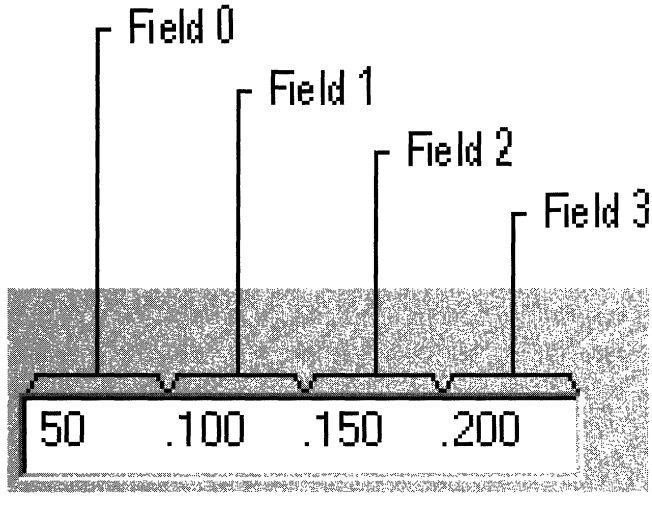
CHAPTER 17

IP Address Controls

An *IP address control* allows the user to enter an IP address in an easily understood format. This control also allows the application to obtain the address in numeric form rather than in text form.

About IP Address Controls

Microsoft Internet Explorer version 4.0 introduces the IP address control, a new control similar to an edit control that allows the user to enter a numeric address in Internet protocol (IP) format. This format consists of four three-digit fields. Each field is treated individually; the field numbers are zero-based and proceed from left to right, as shown in this illustration.



The control allows only numeric text to be entered in each of the fields. Once three digits have been entered in a given field, keyboard focus is moved automatically to the next field. If filling the entire field is not required by the application, the user can enter fewer than three digits. For example, if the field should only contain 21, typing 21 and pressing the RIGHT ARROW key will take the user to the next field.

The default range for each field is 0 to 255, but the application can set the range to any values between those limits with the **IPM_SETRANGE** message.

Note The IP address control is implemented in version 4.71 and later of Comctl32.dll.

Using IP Address Controls

This section describes how to implement an IP address control in your application.

Initializing an IP Address Control

To use an IP address control, call **InitCommonControlsEx** with the ICC_INTERNET_CLASSES flag set in the **dwICC** member of the **INITCOMMONCONTROLSEX** structure.

Creating an IP Address Control

Use the **CreateWindow** or the **CreateWindowEx** API to create an IP address control. The class name for the control is WC_IPADDRESS, which is defined in Commctrl.h. No IP address control-specific styles exist; however, because this is a child control, use the WS_CHILD style as a minimum.

Is an IP Address Control an Edit Control?

An IP address control is not an edit control and it will not respond to EM_ messages. It will send, however, the owner window the following edit control notifications through the **WM_COMMAND** message. Note that the IP address control also will send private IPN_ notifications through the **WM_NOTIFY** message:

Notification	Reason for notification
EN_CHANGE	Sent when any field in the IP address control changes. Like the EN_CHANGE notification from a standard edit control, this notification is received after the screen has been updated.
EN_KILLFOCUS	Sent when the IP address control loses the keyboard focus.
EN_SETFOCUS	Sent when the IP address control gains the keyboard focus.

IP Address Control Reference

IP Address Control Messages

IPM_CLEARADDRESS

Clears the contents of the IP address control.

```
IPM_CLEARADDRESS
    wParam = 0;
    lParam = 0;
```

Return Values

The return value is not used.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

IPM_GETADDRESS

Retrieves the address values for all four fields in the IP address control.

```
IPM_GETADDRESS
    wParam = 0;
    lParam = (LPARAM)(LPDWORD)pdwAddr;
```

Parameters

pdwAddr

Address of a DWORD value that receives the address. The field 3 value will be contained in bits 0 through 7. The field 2 value will be contained in bits 8 through 15. The field 1 value will be contained in bits 16 through 23. The field 0 value will be contained in bits 24 through 31. The **FIRST_IPADDRESS**, **SECOND_IPADDRESS**, **THIRD_IPADDRESS**, and **FOURTH_IPADDRESS** macros also can be used to extract the address information. Zero will be returned as the address for any blank fields.

Return Values

Returns the number of nonblank fields.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

IPM_ISBLANK

Determines if all fields in the IP address control are blank.

```
IPM_ISBLANK
    wParam = 0;
    lParam = 0;
```

Return Values

Returns nonzero if all fields are blank, or zero otherwise.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

IPM_SETADDRESS

Sets the address values for all four fields in the IP address control.

```
IPM_SETADDRESS
    wParam = 0;
    lParam = (LPARAM)(DWORD)dwAddr;
```

Parameters

dwAddr

DWORD value that contains the new address. The field 3 value is contained in bits 0 through 7. The field 2 value is contained in bits 8 through 15. The field 1 value is contained in bits 16 through 23. The field 0 value is contained in bits 24 through 31. The **MAKEIPADDRESS** macro also can be used to create the address information.

Return Values

The return value is not used.

Remarks

This message does not generate an **IPN_FIELDCHANGED** notification.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

IPM_SETFOCUS

Sets the keyboard focus to the specified field in the IP address control. All of the text in that field will be selected.

```
IPM_SETFOCUS
  wParam = (WPARAM)nField;
  lParam = 0;
```

Parameters

nField

Zero-based field index to which the focus should be set. If this value is greater than the number of fields, focus is set to the first blank field. If all fields are nonblank, focus is set to the first field.

Return Values

The return value is not used.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

IPM_SETRANGE

Sets the valid range for the specified field in the IP address control.

```
IPM_SETRANGE
    wParam = (WPARAM)nField;
    lParam = (LPARAM)(WORD)wRange;
```

Parameters

nField
Zero-based field index to which the range will be applied.

wRange
WORD value that contains the lower limit of the range in the low-order byte and the upper limit in the high-order byte. Both of these values are inclusive. The **MAKEIPRANGE** macro also can be used to create the range.

Return Values

Returns nonzero if successful, or zero otherwise.

Remarks

If the user enters a value in the field that is outside of this range, the control will send the **IPN_FIELDCHANGED** notification with the entered value. If the value is still outside of the range after sending the notification, the control will attempt to change the entered value to the closest range limit.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

IP Address Control Notifications

IPN_FIELDCHANGED

Sent when the user changes a field in the control or moves from one field to another. This notification message is sent in the form of a **WM_NOTIFY** message.

```
IPN_FIELDCHANGED
    lpnmipa = (LPNMIPADDRESS) lParam;
```

Parameters

lpnmipa

Address of an **NMIPADDRESS** structure that contains information about the changed address. The **iValue** member of this structure will contain the entered value, even if it is out of the range of the field. You can modify this member to any value that is within the range for the field in response to this notification.

Return Values

The return value is ignored.

Remarks

This notification is not sent in response to a **IPM_SETADDRESS** message.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

IP Address Control Macros

FIRST_IPADDRESS

Extracts the field 0 value from a packed IP address retrieved with the **IPM_GETADDRESS** message.

```
BYTE FIRST_IPADDRESS(
  LPARAM lParam
);
```

Parameters

lParam

Packed IP address value.

Return Values

Returns a BYTE value that contains the field 0 value.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

FOURTH_IPADDRESS

Extracts the field 3 value from a packed IP address retrieved with the **IPM_GETADDRESS** message.

```
BYTE FOURTH_IPADDRESS(
  LPARAM lParam
);
```

Parameters

lParam
Packed IP address value.

Return Values

Returns a BYTE value that contains the field 3 value.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

MAKEIPADDRESS

Packs four byte values into a single LPARAM suitable for use with the **IPM_SETADDRESS** message.

```
LPARAM MAKEIPADDRESS(
  BYTE b0,
  BYTE b1,
  BYTE b2,
  BYTE b3
);
```

Parameters

b0
Field 0 address.

b1
Field 1 address.

b2
Field 2 address.

b3
Field 3 address.

Return Values

Returns an LPARAM value that contains the address.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

MAKEIPRANGE

Packs two byte values into a single LPARAM suitable for use with the **IPM_SETRANGE** message.

```
LPARAM MAKEIPRANGE(
  BYTE low,
  BYTE high
);
```

Parameters

low
Lower limit of the range.

high
Upper limit of the range.

Return Values

Returns an LPARAM value that contains the range.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

SECOND_IPADDRESS

Extracts the field 1 value from a packed IP address retrieved with the **IPM_GETADDRESS** message.

```
BYTE SECOND_IPADDRESS(
  LPARAM lParam
);
```

Parameters

lParam
Packed IP address value.

Return Values

Returns a BYTE value that contains the field 1 value.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

THIRD_IPADDRESS

Extracts the field 2 value from a packed IP address retrieved with the **IPM_GETADDRESS** message.

```
BYTE THIRD_IPADDRESS(
  LPARAM lParam
);
```

Parameters

lParam
Packed IP address value.

Return Values

Returns a BYTE value that contains the field 2 value.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

IP Address Control Structures

NMIPADDRESS

Contains information for the **IPN_FIELDCHANGED** notification message.

```
typedef struct tagNMIPADDRESS{
    NMHDR hdr;
    int iField;
    int iValue;
} NMIPADDRESS, *LPNMIPADDRESS;
```

Members

hdr
NMHDR structure that contains additional information about the notification message.

iField
Zero-based number of the field that was changed.

iValue
New value of the field specified in the **iField** member. While processing the **IPN_FIELDCHANGED** notification, this member can be set to any value that is within the range of the field and the control will place this new value in the field.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

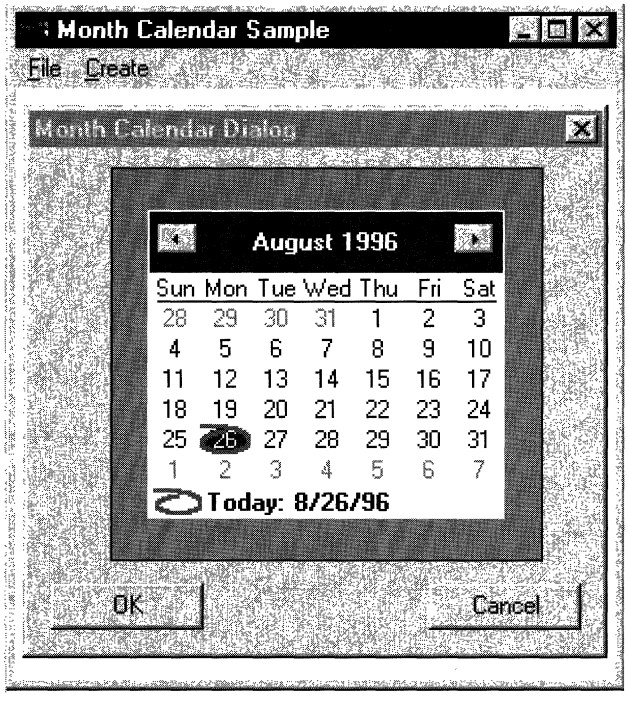
CHAPTER 18

Month-Calendar Controls

A *month-calendar control* implements a calendar-like user interface. This provides the user with a very intuitive and recognizable method of entering or selecting a date. The control also provides the application with the means to obtain and set the date information in the control using existing data types.

About Month-Calendar Controls

The month-calendar control is implemented in version 4.70 and later of Comctl32.dll. It provides a simple and intuitive way for a user to select a date from a familiar interface. The following illustration shows a month-calendar control in a dialog box.



An application creates a month-calendar control by calling the **CreateWindowEx** function, and specifying MONTHCAL_CLASS as the window class. The class is registered when the month-calendar class is loaded from the common controls dynamic-link library (DLL). Register this class by calling the **InitCommonControlsEx** function, specifying the ICC_DATE_CLASSES bit flag in the accompanying **INITCOMMONCONTROLSEX** structure.

Note Windows does not support dates prior to 1601. See **FILETIME** for details.

The month-calendar control is based on the Gregorian calendar, which was introduced in 1753. It will not calculate dates that are consistent with the Julian calendar that was in use prior to 1753.

The Month-Calendar Control User Interface

The month-calendar control user interface allows the user to select a date from the displayed days or change the control's display in various ways.

- **Scrolling the control's display**

By default, when a user clicks the arrow buttons in the top left or top right of the month-calendar control, it updates its display to show the previous or next month. If the month-calendar control is displaying more than one month at a time, the display changes by the number of months currently in view. That is, if the month-calendar displays January, February, and March, and the user clicks the top-right arrow button, the control updates its display to show April, May, and June. The user also can perform the same action by clicking the partial months displayed before the first month and after the last month.

The following keyboard commands also can be used to change the current month:

PAGE UP (VK_NEXT)
Move to the next month.

PAGE DOWN (VK_PRIOR)
Move to previous month.

HOME (VK_HOME)
Move to the first day of the current month.

END (VK_END)
Move to last day of the current month.

CTRL+HOME
Move to the first visible month.

CTRL+END
Move to last visible month.

An application can change the number of months by which the control updates its display by using the **MCM_SETMONTHDELTA** message or the corresponding macro, **MonthCal_SetMonthDelta**. However, the PAGE UP and PAGE DOWN keys, change the selected month by one, regardless of the number of months displayed or the value set by MCM_SETMONTHDELTA.

- **Selecting a nonadjacent month**

When a user clicks the name of a displayed month, a pop-up menu appears that lists all months within the year. The user can select a month on the list. If the user's selection is not visible, the month-calendar control scrolls its display to show the chosen month.

- **Selecting a different year**

If the user clicks the year displayed next to a month name, an up-down control appears in place of the year. The user can change the year with this control. The month-calendar control updates its display for the selected year when the up-down control loses focus. The related keyboard commands are:

CTRL + VK_NEXT
Move to the next year.

CTRL + VK_PRIOR
Move to previous year.

- **Selecting the current day**

If a month-calendar control is not using the **MCS_NOTODAY** style, the user can return to the current day by clicking the "today" text at the bottom of the control. If the current day is not visible, the control updates its display to show it. Related keyboard commands are:

VK_LEFT
Move to previous day.

VK_RIGHT
Move to next day.

VK_UP
Move to previous week.

VK_DOWN
Move to next week.

Day States

Month-calendar controls that use the **MCS_DAYSTATE** style support day states. The control uses day state information to determine how it draws specific days within the control. Day-state information is expressed as a 32-bit data type, **MONTHDAYSTATE**. Each bit in a **MONTHDAYSTATE** bit field (1 through 31) represents the state of a day in a month. If a bit is on, the corresponding day will be displayed in bold; otherwise, it will be displayed with no emphasis.

An application can set daystate information-explicitly by sending the **MCM_SETDAYSTATE** message or by using the corresponding macro, **MonthCal_SetDayState**. Additionally, month-calendar controls that use the **MCS_DAYSTATE** style send **MCN_GETDAYSTATE** notification messages to request day-state information. For more information on supporting day states, see *Processing the MCN_GETDAYSTATE Notification Message* and *Preparing the MONTHDAYSTATE Array*.

Month-Calendar Control Styles

Month-calendar controls have several **styles** that determine their appearance and behavior. When you create the control using **CreateWindowEx**, include the desired styles in the **dwStyle** parameter.

After creating the control, you can change all of the styles except for **MCS_DAYSTATE** and **MCS_MULTISELECT**. To change these styles, you will need to destroy the existing control and create a new one that has the desired styles. To retrieve or change any other window styles, use the **GetWindowLong** and **SetWindowLong** functions.

Month-calendar controls that use the **MCS_MULTISELECT** style allow the user to select a range of days. By default, the control allows the user to select seven contiguous days. Your application can change the control's default behavior by using the **MCM_SETMAXSELCOUNT** message or the accompanying macro, **MonthCal_SetMaxSelCount**.

When a month-calendar control uses the **MCS_WEEKNUMBERS** style, it displays week numbers at the left side of each month. If the **MCS_NOTODAY** style is applied, the control no longer circles the current day.

The **MCS_DAYSTATE** style is helpful when you want the control to highlight specific dates by displaying them in bold. For more information, see *Day States*.

Localization

The month-calendar control gets its format and all strings from LOCALE_USER_DEFAULT. For Windows 2000 and later systems, it gets the month-title format from LOCALE_SYEARMONTH. Even with the same DLL version, the appearance of the control may vary slightly depending on which system your application is running on. For example, with Windows NT 4.0, the month title will look like: "September 1998". On Windows 2000, it will look like: "September, 1998".

Month-Calendar Control Notification Messages

A month-calendar control sends notification messages when it receives user input or must request day-state information (**MCS_DAYSTATE** style only). The control's parent receives these notifications as **WM_NOTIFY** messages.

The following notification messages are used with month-calendar controls:

Notification	Description
MCN_GETDAYSTATE	Requests information about which days should be displayed in bold. For more information, see *Preparing the MONTHDAYSTATE Array*.

Notification	Description
MCN_SELCHANGE	Notifies the parent that the selected date or range of dates has changed. The control sends this notification when the user explicitly changes the selection within the current month or when the selection is changed implicitly in response to next/previous month exploration.
MCN_SELECT	Notifies the parent that the user has explicitly selected a date.

Times in the Month-Calendar Control

Because the month-calendar control cannot be used to select a time, the time related fields of the **SYSTEMTIME** structure need to be handled differently. When the control is created, it will insert the current time into its "today" date and time.

When a time is set programmatically later, the control either will copy the time fields as they are or validate them first and then, if invalid, store the current default times. Following is a list of the messages that set a date and a description of how the time fields are treated by the message:

Notification	Description
MCM_SETCURSEL	The control will copy the time fields as they are, without validation or modification.
MCM_SETRANGE	The time fields of the structures passed in will be validated. If they are valid, the time fields will be copied without modification. If they are invalid, the control will copy the time fields from the "today" date and time.
MCM_SETSELRANGE	The time fields of the structures passed in will be validated. If they are valid, the time fields will be copied without modification. If they are invalid, the control will retain the time fields from the current selection ranges.
MCM_SETTODAY	The control will copy the time fields as they are, without validation or modification.

When a date is retrieved from the control, the time fields will be copied from the stored times without modification. Handling of the time fields by the control is provided as a convenience to the programmer. The control does not examine or modify the time fields as a result of any operation other than those listed above.

Using Month-Calendar Controls

This section provides information and sample code for implementing month-calendar controls.

Creating a Month-Calendar Control

To create a month-calendar control, use the **CreateWindowEx** function, specifying MONTHCAL_CLASS as the window class. You first must register the window class by calling the **InitCommonControlsEx** function, specifying the ICC_DATE_CLASSES bit in the accompanying **INITCOMMONCONTROLSEX** structure.

The following example demonstrates how to create a month-calendar control in an existing modeless dialog box. Note that the size values passed to **CreateWindowEx** are all zeros. Because the minimum required size depends on the font the control uses, the DoNotify example function uses the **MonthCal_GetMinReqRect** macro to request size information and then resizes the control by calling **SetWindowPos**. If you subsequently change the font with **WM_SETFONT**, the dimensions of the control will not change. You must call **MonthCal_GetMinReqRect** again and resize the control to fit the new font:

```
//    CreateMonthCal -- Creates a month-calendar control in a dialog
//                      box.  Returns the handle to the month-calendar
//                      control if successful, or NULL otherwise.
//
//    hwndOwner   -- Handle to the owner of the dialog box.
//    g_hinst     -- Global handle to the program instance.
//
/////
HWND WINAPI CreateMonthCal(HWND hwndOwner)
{
    HWND hwnd;
    RECT rc;
    INITCOMMONCONTROLSEX icex;
    // Load the window class.
    icex.dwSize = sizeof(icex);
    icex.dwICC  = ICC_DATE_CLASSES;
    InitCommonControlsEx(&icex);

    // Create a modeless dialog box to hold the control.
    g_hwndDlg = CreateDialog(g_hinst,
        MAKEINTRESOURCE(IDD_DIALOG1),
        hwndOwner,
        DlgProc);

    // Create the month-calendar.
    hwnd = CreateWindowEx(0,
        MONTHCAL_CLASS,
        "",
        WS_BORDER | WS_CHILD | WS_VISIBLE | MCS_DAYSTATE,
        0,0,0,0, // resize it later
            g_hwndDlg,
            NULL,
```

```
            g_hinst,
            NULL);

    // Get the size required to show an entire month.
    MonthCal_GetMinReqRect(hwnd, &rc);

// Arbitrary values
#define LEFT 35
#define TOP  40

    // Resize the control now that the size values have
    // been obtained.
    SetWindowPos(hwnd, NULL, TOP, LEFT,
        LEFT + rc.right, TOP + rc.bottom,
        SWP_NOZORDER);

    // Set colors for aesthetics.
    MonthCal_SetColor(hwnd, MCSC_BACKGROUND,RGB(175,175,175));
    MonthCal_SetColor(hwnd, MCSC_MONTHBK, RGB(248,245,225));

    return(hwnd);
}
```

Processing the MCN_GETDAYSTATE Notification Message

Month-calendar controls send the **MCN_GETDAYSTATE** notification message to request information about how the days within the visible months should be displayed. The following application-defined function, DoNotify, processes **MCN_GETDAYSTATE** by filling an array of MONTHDAYSTATE values to highlight the 15th day of each month.

DoNotify extracts the number of MONTHDAYSTATE values needed from the **cDayState** member of the **NMDAYSTATE** structure that *lParam* points to. It then loops to set the 15th bit in each element of the array, using the application-defined **BOLDDAY** macro.

```
BOOL WINAPI DoNotify(HWND hwnd, UINT msg, WPARAM wParam,LPARAM lParam)
{
#define BOLDDAY(ds,iDay) if(iDay>0 && iDay<32)\
                (ds)|=(0x00000001<<(iDay-1))

#define lpnmDS ((NMDAYSTATE *)lParam)
#define MAX_MONTHS 12

    MONTHDAYSTATE mds[MAX_MONTHS];
    INT i, iMax;
    LPNMHDR hdr = (LPNMHDR)lParam;
```

(continued)

(continued)

```
    switch(hdr->code){
        case MCN_GETDAYSTATE:
            iMax=lpnmDS->cDayState;

            for(i=0;i<iMax;i++){
                mds[i] = (MONTHDAYSTATE)0;
                BOLDDAY(mds[i],15);
            }
            lpnmDS->prgDayState = mds;
            break;
    }
    return FALSE;
}
```

Preparing the MONTHDAYSTATE Array

Both the **MCM_SETDAYSTATE** message and **MCN_GETDAYSTATE** notification message require an array of MONTHDAYSTATE values to determine how dates will be displayed. Each month that the control displays must have a corresponding element within the array.

To support these messages, your application must properly prepare the array. The following is a simple macro that sets a bit in a MONTHDAYSTATE value for a given day within that month:

```
#define BOLDDAY(ds,iDay) if(iDay>0 && iDay<32)\
                (ds)|=(0x00000001<<(iDay-1))
```

Using this macro, an application could simply loop through an array of important dates, setting bits within the corresponding array elements. This approach is not the most efficient, of course, but works well for many purposes. As long as your application sets MONTHDAYSTATE bits appropriately, it does not matter how those bits were set.

Month-Calendar Control Styles

The following are the styles used with month-calendar controls:

MCS_DAYSTATE
Version 4.70. The month-calendar will send **MCN_GETDAYSTATE** notifications to request information about which days should be displayed in bold. For more information about supporting this style, see *Processing the MCN_GETDAYSTATE Notification Message*.

MCS_MULTISELECT
Version 4.70. The month-calendar will allow the user to select a range of dates within the control. By default, the maximum range is one week. You can change the

maximum range that can be selected by using the **MCM_SETMAXSELCOUNT** message.

MCS_NOTODAY
Version 4.70. The month-calendar control will not display the "today" date at the bottom of the control.

MCS_NOTODAYCIRCLE
Version 4.70. The month-calendar control will not circle the "today" date.

MCS_WEEKNUMBERS
Version 4.70. The month-calendar control will display week numbers (1–52) to the left of each row of days. Week 1 is defined as the first week that contains at least four days.

Month-Calendar Day Numbers

The following list contains the numeric representations of the days of the week that are used by the month-calendar control:

Value	Day of Week
0	Monday
1	Tuesday
2	Wednesday
3	Thursday
4	Friday
5	Saturday
6	Sunday

Month-Calendar Control Reference

Month-Calendar Control Messages

MCM_GETCOLOR

Retrieves the color for a given portion of a month-calendar control. You can send this message explicitly or by using the **MonthCal_GetColor** macro.

```
MCM_GETCOLOR
    wParam = (WPARAM)(INT)iColor;
    lParam = 0;
```

Parameters

iColor

INT value specifying which month-calendar color to retrieve. This value can be one of the following:

MCSC_BACKGROUND	Retrieve the background color displayed between months.
MCSC_MONTHBK	Retrieve the background color displayed within the month.
MCSC_TEXT	Retrieve the color used to display text within a month.
MCSC_TITLEBK	Retrieve the background color displayed in the calendar's title.
MCSC_TITLETEXT	Retrieve the color used to display text within the calendar's title.
MCSC_TRAILINGTEXT	Retrieve the color used to display header day and trailing day text. Header and trailing days are the days from the previous and following months that appear on the current month-calendar.

Return Values

Returns a **COLORREF** value that represents the color setting for the specified portion of the month-calendar control, if successful. Otherwise, this message returns –1.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

MCM_GETCURSEL

Retrieves the currently selected date. You can send this message explicitly or by using the **MonthCal_GetCurSel** macro.

```
MCM_GETCURSEL
    wParam = 0;
    lParam = (LPARAM) (LPSYSTEMTIME) lpSysTime;
```

Parameters

lpSysTime

Address of a **SYSTEMTIME** structure that will receive the currently selected date information. This parameter must be a valid address and cannot be NULL.

Return Values

Returns nonzero if successful, or zero otherwise. This message always will fail when applied to month-calendar controls set to the **MCS_MULTISELECT** style.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

Times in the Month-Calendar Control

MCM_GETFIRSTDAYOFWEEK

Retrieves the first day of the week for a month-calendar control. You can send this message explicitly or by using the **MonthCal_GetFirstDayOfWeek** macro.

```
MCM_GETFIRSTDAYOFWEEK
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a DWORD value that contains two values. The high word is a BOOL value that is nonzero if the first day of the week is set to something other than LOCALE_IFIRSTDAYOFWEEK, or zero otherwise. The low word is an INT value that represents the first day of the week. This will be one of the **day numbers**.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

MCM_GETMAXSELCOUNT

Retrieves the maximum date range that can be selected in a month-calendar control. You can send this message explicitly or by using the **MonthCal_GetMaxSelCount** macro.

```
MCM_GETMAXSELCOUNT
    wParam = 0;
    lParam = 0;
```

Return Values

Returns an INT value that represents the total number of days that can be selected for the control.

Remarks

You can change the maximum date range that can be selected by using the **MCM_SETMAXSELCOUNT** message.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

MCM_GETMAXTODAYWIDTH

Retrieves the maximum width of the "today" string in a month-calendar control. This includes the label text and the date text. You can send this message explicitly or by using the **MonthCal_GetMaxTodayWidth** macro.

```
MCM_GETMAXTODAYWIDTH
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the width of the "today" string, in pixels.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

MCM_GETMINREQRECT

Retrieves the minimum size required to display a full month in a month-calendar control. You can send this message explicitly or by using the **MonthCal_GetMinReqRect** macro.

```
MCM_GETMINREQRECT
    wParam = 0;
    lParam = (LPARAM) (LPRECT) lpRectInfo;
```

Parameters

lpRectInfo

Address of a **RECT** structure that will receive bounding rectangle information. This parameter must be a valid address and cannot be NULL.

Return Values

Returns nonzero and *lpRectInfo* receives the applicable bounding information if successful. Otherwise, the message returns zero.

Remarks

The minimum required window size for a month-calendar control depends on the currently selected font, control styles, system metrics, and regional settings. When an application changes anything that affects the minimum window size, or processes a **WM_SETTINGCHANGE** message, it should send **MCM_GETMINREQRECT** to determine the new minimum size.

Note The rectangle returned by **MCM_GETMINREQRECT** does not include the width of the "Today" string, if it is present. If the **MCS_NOTODAY** style is not set, your application should also retrieve the rectangle that defines the "Today" string width by sending a **MCM_GETMAXTODAYWIDTH** message. Use the larger of the two rectangles to ensure that the "Today" string is not clipped.

The **top** and **left** members of the structure pointed to by *lpRectInfo* will always be zero. The **right** and **bottom** members represent the minimum *cx* and *cy* required for the control.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

MCM_GETMONTHDELTA

Retrieves the scroll rate for a month-calendar control. The scroll rate is the number of months that the control moves its display when the user clicks a scroll button. You can send this message explicitly or by using the **MonthCal_GetMonthDelta** macro.

```
MCM_GETMONTHDELTA
    wParam = 0;
    lParam = 0;
```

Return Values

If the month delta was previously set using the **MCM_SETMONTHDELTA** message, returns an INT value that represents the month-calendar's current scroll rate. If the month delta was not previously set using the **MCM_SETMONTHDELTA** message, or the month delta was reset to the default, returns an INT value that represents the current number of months visible.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

MCM_GETMONTHRANGE

Retrieves date information (using **SYSTEMTIME** structures) that represents the high and low limits of a month-calendar control's display. You can send this message explicitly or by using the **MonthCal_GetMonthRange** macro.

```
MCM_GETMONTHRANGE
    wParam = (WPARAM)(DWORD) dwFlag;
    lParam = (LPARAM)(LPSYSTEMTIME) lprgSysTimeArray;
```

Parameters

dwFlag

Value specifying the scope of the range limits to be retrieved. This value must be one of the following:

GMR_DAYSTATE	Include preceding and trailing months of visible range that are only partially displayed.
GMR_VISIBLE	Include only those months that are entirely displayed.

lprgSysTimeArray

Address of a two-element array of **SYSTEMTIME** structures that will receive the lower and upper limits of the scope specified by *dwFlag*. The lower and upper limits are placed in lprgSysTimeArray[0] and lprgSysTimeArray[1], respectively. This parameter must be a valid address and cannot be NULL.

Return Values

Returns an INT value that represents the range, in months, spanned by the two limits returned at *lprgSysTimeArray*.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

See Also

Times in the Month-Calendar Control

MCM_GETRANGE

Retrieves the minimum and maximum allowable dates set for a month-calendar control. You can send this message explicitly or by using the **MonthCal_GetRange** macro.

```
MCM_GETRANGE
    wParam = 0;
    lParam = (LPARAM)(LPSYSTEMTIME) lprgSysTimeArray;
```

Parameters

lprgSysTimeArray

Address of a two-element array of **SYSTEMTIME** structures that will receive the date limit information. The minimum limit is set in lprgSysTimeArray[0], and lprgSysTimeArray[1] receives the maximum limit. If either element is set to all zeros, then no corresponding limit is set for the month-calendar control. This parameter must be a valid address and cannot be NULL.

Return Values

Returns a DWORD that can be zero (no limits are set) or a combination of the following values that specify limit information:

GDTR_MAX	A maximum limit is set for the control; lprgSysTimeArray[0] is valid and contains the applicable date information.
GDTR_MIN	A minimum limit is set for the control; lprgSysTimeArray[1] is valid and contains the applicable date information.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

See Also

Times in the Month-Calendar Control

MCM_GETSELRANGE

Retrieves date information that represents the upper and lower limits of the date range currently selected by the user. You can send this message explicitly or by using the **MonthCal_GetSelRange** macro.

```
MCM_GETSELRANGE
    wParam = 0;
    lParam = (LPARAM)(LPSYSTEMTIME) lprgSysTimeArray;
```

Parameters

lprgSysTimeArray

Address of a two-element array of **SYSTEMTIME** structures that will receive the lower and upper limits of the user's selection. The lower and upper limits are placed in lprgSysTimeArray[0] and lprgSysTimeArray[1], respectively. This parameter must be a valid address and cannot be NULL.

Return Values

Returns nonzero if successful, or zero otherwise. MCM_GETSELRANGE will fail if applied to a month-calendar control that does not use the **MCS_MULTISELECT** style.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

See Also

Times in the Month-Calendar Control

MCM_GETTODAY

Retrieves the date information for the date specified as "today" for a month-calendar control. You can send this message explicitly or by using the **MonthCal_GetToday** macro.

```
MCM_GETTODAY
    wParam = 0;
    lParam = (LPARAM)(LPSYSTEMTIME) lpToday;
```

Parameters

lpToday

Address of a **SYSTEMTIME** structure that will receive the date information. This parameter must be a valid address and cannot be NULL.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

See Also

Times in the Month-Calendar Control

MCM_GETUNICODEFORMAT

Retrieves the UNICODE character format flag for the control. You can send this message explicitly or use the **MonthCal_GetUnicodeFormat** macro.

```
MCM_GETUNICODEFORMAT
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the UNICODE format flag for the control. If this value is nonzero, the control is using UNICODE characters. If this value is zero, the control is using ANSI characters.

Remarks

See the remarks for **CCM_GETUNICODEFORMAT** for a discussion of this message.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

See Also

MCM_SETUNICODEFORMAT

MCM_HITTEST

Determines which portion of a month-calendar control is at a given point on the screen. You can send this message explicitly or by using the **MonthCal_HitTest** macro.

```
MCM_HITTEST
    wParam = 0;
    lParam = (LPARAM)(PMCHITTESTINFO) pMCHitTest;
```

Parameters

pMCHitTest
Address of an **MCHITTESTINFO** structure. Upon sending the message, the **cbSize** member must be set to the size of the **MCHITTESTINFO** structure, and **pt** must be set to the point you want to hit-test.

Return Values

Sets values in members of the **MCHITTESTINFO** structure at *pMCHitTest* and returns a DWORD value that contains one or more of the following:

MCHT_CALENDAR	The given point was within the calendar.
MCHT_CALENDARBK	The given point was in the calendar's background.
MCHT_CALENDARDATE	The given point was on a particular date within the calendar. The **SYSTEMTIME** structure at *lpMCHitTest*->st is set to the date at the given point.
MCHT_CALENDARDATENEXT	The given point was over a date from the next month (partially displayed at the end of the currently displayed month). If the user clicks here, the month-calendar will scroll its display to the next month or set of months.
MCHT_CALENDARDATEPREV	The given point was over a date from the previous month (partially displayed at the end of the currently displayed month). If the user clicks here, the month-calendar will scroll its display to the previous month or set of months.

(continued)

(continued)

MCHT_CALENDARDAY	The given point was over a day abbreviation ("Fri", for example). The **SYSTEMTIME** structure at *lpMCHitTest*->st is set to the corresponding date in the top row.
MCHT_CALENDARWEEKNUM	The given point was over a week number (**MCS_WEEKNUMBERS** style only). The **SYSTEMTIME** structure at *lpMCHitTest*->st is set to the corresponding date in the leftmost column.
MCHT_NEXT	The given point is in an area that will cause the month-calendar to scroll its display to the next month or set of months. This flag is used to modify other hit-test flags.
MCHT_NOWHERE	The given point was not on the month-calendar control, or it was in an inactive portion of the control.
MCHT_PREV	The given point is in an area that will cause the month-calendar to scroll its display to the previous month or set of months. This flag is used to modify other hit-test flags.
MCHT_TITLE	The given point was over a month's title.
MCHT_TITLEBK	The given point was over the background of a month's title.
MCHT_TITLEBTNNEXT	The given point was over the button at the top right corner of the control. If the user clicks here, the month-calendar will scroll its display to the next month or set of months.
MCHT_TITLEBTNPREV	The given point was over the button at the top left corner of the control. If the user clicks here, the month-calendar will scroll its display to the previous month or set of months.
MCHT_TITLEMONTH	The given point was in a month's title bar, over a month name.
MCHT_TITLEYEAR	The given point was in a month's title bar, over the year value.
MCHT_TODAYLINK	The given point was on the "today" link at the bottom of the month-calendar control.

The **uHit** member of the **MCHITTESTINFO** structure at **pMCHitTest** will equal the return value.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

MCM_SETCOLOR

Sets the color for a given portion of a month-calendar control. You can send this message explicitly or by using the **MonthCal_SetColor** macro.

```
MCM_SETCOLOR
    wParam = (WPARAM)(INT) iColor;
    lParam = (LPARAM)(COLORREF) clr;
```

Parameters

iColor

INT value specifying which month-calendar color to set. This value can be one of the following:

MCSC_BACKGROUND	Set the background color displayed between months.
MCSC_MONTHBK	Set the background color displayed within the month.
MCSC_TEXT	Set the color used to display text within a month.
MCSC_TITLEBK	Set the background color displayed in the calendar's title.
MCSC_TITLETEXT	Set the color used to display text within the calendar's title.
MCSC_TRAILINGTEXT	Set the color used to display header day and trailing day text. Header and trailing days are the days from the previous and following months that appear on the current month-calendar.

clr

COLORREF value that represents the color that will be set for the specified area of the month-calendar.

Return Values

Returns a **COLORREF** value that represents the previous color setting for the specified portion of the month-calendar control if successful. Otherwise, the return is –1.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

MCM_SETCURSEL

Sets the currently selected date for a month-calendar control. If the specified date is not in view, the control updates the display to bring it into view. You can send this message explicitly or by using the **MonthCal_SetCurSel** macro.

```
MCM_SETCURSEL
    wParam = 0;
    lParam = (LPARAM)(LPSYSTEMTIME) lpSysTime;
```

Parameters

lpSysTime
Address of a **SYSTEMTIME** structure that contains the date to be set as the current selection.

Return Values

Returns nonzero if successful, or zero otherwise. This message will fail if applied to a month-calendar control that uses the **MCS_MULTISELECT** style.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

See Also

Times in the Month-Calendar Control

MCM_SETDAYSTATE

Sets the day states for all months that are currently visible within a month-calendar control. You can send this message explicitly or by using the **MonthCal_SetDayState** macro.

```
MCM_SETDAYSTATE
    wParam = (WPARAM) iMonths;
    lParam = (LPARAM)(LPMONTHDAYSTATE) lpDayStateArray;
```

Parameters

iMonths
Value indicating how many elements are in the array that *lpDayStateArray* points to.

lpDayStateArray
Address of an array of **MONTHDAYSTATE** values that define how the month-calendar control will draw each day in its display.

Return Values

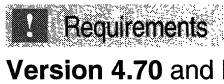
Returns nonzero if successful, or zero otherwise.

Remarks

The array at *lpDayStateArray* must contain as many elements as the value returned by the following macro:

```
MonthCal_GetMonthRange(hwndMC, GMR_DAYSTATE, NULL);
```

Keep in mind that the array at *lpDayStateArray* must contain **MONTHDAYSTATE** values that correspond with all months currently in the control's display, in chronological order. This includes the two months only partially displayed before the first month and after the last month. For more information about preparing your array, see **Preparing the MONTHDAYSTATE Array**.



Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

MCM_SETFIRSTDAYOFWEEK

Sets the first day of the week for a month-calendar control. You can send this message explicitly or by using the **MonthCal_SetFirstDayOfWeek** macro.

```
MCM_SETFIRSTDAYOFWEEK
    wParam = 0;
    lParam = (LPARAM)(INT) iDay;
```

Parameters

iDay

INT value representing which day is to be set as the first day of the week. This value must be one of the day numbers.

Return Values

Returns a DWORD value that contains two values. The high word is a BOOL value that is nonzero if the previous first day of the week did not equal LOCALE_IFIRSTDAYOFWEEK, or zero otherwise. The low word is an INT value that represents the previous first day of the week.

Remarks

If the first day of the week is set to anything other than the default (LOCALE_IFIRSTDAYOFWEEK), the control will not automatically update first-day-of-the-week changes based on locale changes.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

MCM_SETMAXSELCOUNT

Sets the maximum number of days that can be selected in a month-calendar control. You can send this message explicitly or by using the **MonthCal_SetMaxSelCount** macro.

```
MCM_SETMAXSELCOUNT
    wParam = (WPARAM)(INT) iMax;
    lParam = 0;
```

Parameters

iMax

INT value that will be set to represent the maximum number of days that can be selected.

Return Values

Returns nonzero if successful, or zero otherwise. This message will fail if applied to a month-calendar control that does not use the **MCS_MULTISELECT** style.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

MCM_SETMONTHDELTA

Sets the scroll rate for a month-calendar control. The scroll rate is the number of months that the control moves its display when the user clicks a scroll button. You can send this message explicitly or by using the **MonthCal_SetMonthDelta** macro.

```
MCM_SETMONTHDELTA
    wParam = (WPARAM)(INT) iDelta;
    lParam = 0;
```

Parameters

iDelta

Value representing the number of months to be set as the control's scroll rate. If this value is zero, the month delta is reset to the default, which is the number of months displayed in the control.

Return Values

Returns an INT value that represents the previous scroll rate. If the scroll rate was not previously set, the return value is zero.

Remarks

The PAGE UP and PAGE DOWN keys, VK_PRIOR and VK_NEXT, change the selected month by one, regardless of the number of months displayed or the value set by MCM_SETMONTHDELTA.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

MCM_SETRANGE

Sets the minimum and maximum allowable dates for a month-calendar control. You can send this message explicitly or by using the **MonthCal_SetRange** macro.

```
MCM_SETRANGE
    wParam = (WPARAM)(SHORT) fWhichLimit;
    lParam = (LPARAM)(LPSYSTEMTIME) lprgSysTimeArray;
```

Parameters

fWhichLimit

Flag values that specify which date limits are being set. This value must be one or both of the following:

GDTR_MAX	The maximum allowable date is being set. The **SYSTEMTIME** structure at lprgSysTimeArray[1] must contain date information.
GDTR_MIN	The minimum allowable date is being set. The **SYSTEMTIME** structure at lprgSysTimeArray[0] must contain date information.

lprgSysTimeArray

Address of a two-element array of **SYSTEMTIME** structures that contain the date limits. The maximum limit must be in lpSysTimeArray[1] if GDTR_MAX is specified, and lpSysTimeArray[0] must contain the minimum limit if GDTR_MIN is specified.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).

Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

See Also

Times in the Month-Calendar Control

MCM_SETSELRANGE

Sets the selection for a month-calendar control to a given date range. You can send this message explicitly or by using the **MonthCal_SetSelRange** macro.

```
MCM_SETSELRANGE
    wParam = 0;
    lParam = (LPARAM)(LPSYSTEMTIME) lprgSysTimeArray;
```

Parameters

lprgSysTimeArray
Address of a two-element array of **SYSTEMTIME** structures that contain date information representing the selection limits. The first selected date must be specified in lpSysTimeArray[0], and the last selected date must be specified in lpSysTimeArray[1].

Return Values

Returns nonzero if successful, or zero otherwise. This message will fail if applied to a month-calendar control that does not use the MCS_MULTISELECT style.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

See Also

Times in the Month-Calendar Control

MCM_SETTODAY

Sets the "today" selection for a month-calendar control. You can send this message explicitly or by using the **MonthCal_SetToday** macro.

```
MCM_SETTODAY
    wParam = 0;
    lParam = (LPARAM)(LPSYSTEMTIME) lpSysTime;
```

Parameters

lpSysTime

Address of a **SYSTEMTIME** structure that contains the date to be set as the "today" selection for the control. If this parameter is set to NULL, the control returns to the default setting.

Return Values

The return value for this message is not used.

Remarks

If the "today" selection is set to any date other than the default, the following conditions apply:

- The control will not automatically update the "today" selection when the time passes midnight for the current day.
- The control will not automatically update its display based on locale changes.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

See Also

Times in the Month-Calendar Control

MCM_SETUNICODEFORMAT

Sets the UNICODE character format flag for the control. This message allows you to change the character set used by the control at run time instead of having to re-create

the control. You can send this message explicitly or use the **MonthCal_SetUnicodeFormat** macro.

```
MCM_SETUNICODEFORMAT
    wParam = (WPARAM)(BOOL)fUnicode;
    lParam = 0;
```

Parameters

fUnicode

Determines the character set that is used by the control. If this value is nonzero, the control will use UNICODE characters. If this value is zero, the control will use ANSI characters.

Return Values

Returns the previous UNICODE format flag for the control.

Remarks

See the remarks for **CCM_SETUNICODEFORMAT** for a discussion of this message.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

See Also

MCM_GETUNICODEFORMAT

Month-Calendar Control Macros

MonthCal_GetColor

Retrieves the color for a given portion of a month-calendar control. You can use this macro or send the **MCM_GETCOLOR** message explicitly.

```
COLORREF MonthCal_GetColor(
  HWND hwndMC,
  INT iColor
);
```

Parameters

hwndMC
Handle to a month-calendar control.

iColor
INT value specifying which month-calendar color to retrieve. This value can be one of the following:

MCSC_BACKGROUND	Retrieve the background color displayed between months.
MCSC_MONTHBK	Retrieve the background color displayed within the month.
MCSC_TEXT	Retrieve the color used to display text within a month.
MCSC_TITLEBK	Retrieve the background color displayed in the calendar's title.
MCSC_TITLETEXT	Retrieve the color used to display text within the calendar's title.
MCSC_TRAILINGTEXT	Retrieve the color used to display header day and trailing day text. Header and trailing days are, respectively, the days from the previous and following months that appear on the current month-calendar.

Return Values

Returns a **COLORREF** value that represents the color setting for the specified portion of the month-calendar control if successful. Otherwise, the return is –1.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_GetCurSel

Retrieves the currently selected date. You can use this macro or send the **MCM_GETCURSEL** message explicitly.

```
BOOL MonthCal_GetCurSel(
  HWND hwndMC,
  LPSYSTEMTIME lpSysTime
);
```

Parameters

hwndMC
Handle to a month-calendar control.

lpSysTime
Address of a **SYSTEMTIME** structure that will receive the currently selected date information. This parameter must be a valid address and cannot be NULL.

Return Values

Returns nonzero if successful, or zero otherwise. This macro will always fail when applied to month-calendar controls that are set to the **MCS_MULTISELECT** style.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_GetFirstDayOfWeek

Retrieves the first day of the week for a month-calendar control. You can use this macro or send the **MCM_GETFIRSTDAYOFWEEK** message explicitly.

```
DWORD MonthCal_GetFirstDayOfWeek(
  HWND hwndMC
);
```

Parameters

hwndMC
Handle to a month-calendar control.

Return Values

Returns a DWORD value that contains two values. The high word is a BOOL value that is nonzero if the first day of the week is set to something other than LOCALE_IFIRSTDAYOFWEEK, or zero otherwise. The low word is an INT value that represents the first day of the week. This will be one of the day numbers.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_GetMaxSelCount

Retrieves the maximum date range that can be selected in a month-calendar control. You can use this macro or send the **MCM_GETMAXSELCOUNT** message explicitly.

```
DWORD MonthCal_GetMaxSelCount(
  HWND hwndMC
);
```

Parameters

hwndMC
Handle to a month-calendar control.

Return Values

Returns an INT value that represents the total number of days that can be selected for the control.

Remarks

You can change the maximum date range that can be selected by using the **MCM_SETMAXSELCOUNT** message.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_GetMaxTodayWidth

Retrieves the maximum width of the "today" string in a month-calendar control. This includes the label text and the date text. You can use this macro or send the **MCM_GETMAXTODAYWIDTH** message explicitly.

```
DWORD MonthCal_GetMaxTodayWidth(
  HWND hwndMC
);
```

Parameters

hwndMC
Handle to a month-calendar control.

Return Values

Returns the width of the "today" string, in pixels.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_GetMinReqRect

Retrieves the minimum size required to display a full month in a month-calendar control. Size information is presented in the form of a **RECT** structure. You can use this macro or send the **MCM_GETMINREQRECT** message explicitly.

```
BOOL MonthCal_GetMinReqRect(
  HWND hwndMC,
  LPRECT lpRectInfo
);
```

Parameters

hwndMC
Handle to a month-calendar control.

lpRectInfo
Address of a **RECT** structure that will receive bounding rectangle information. This parameter must be a valid address and cannot be NULL.

Return Values

Returns nonzero and *lpRectInfo* receives the applicable bounding information if successful. Otherwise, the return is zero.

Remarks

The minimum required window size for a month-calendar control depends on the currently selected font, control styles, system metrics, and regional settings. When an application changes anything that affects the minimum window size, or processes a **WM_SETTINGCHANGE** message, it should call **MonthCal_GetMinReqRect** to determine the new minimum size.

Note The rectangle returned by **MonthCal_GetMinReqRect** does not include the width of the "Today" string, if it is present. If the **MCS_NOTODAY** style is not set, your application should also retrieve the rectangle that defines the "Today" string width by calling the **MonthCal_GetMaxTodayWidth** macro. Use the larger of the two rectangles to ensure that the "Today" string is not clipped.

The **top** and **left** members of *lpRectInfo* will always be zero. The **right** and **bottom** members represent the minimum *cx* and *cy* required for the control.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_GetMonthDelta

Retrieves the scroll rate for a month-calendar control. The scroll rate is the number of months that the control moves its display when the user clicks a scroll button. You can use this macro or send the **MCM_GETMONTHDELTA** message explicitly.

```
INT MonthCal_GetMonthDelta(
  HWND hwndMC
);
```

Parameters

hwndMC
Handle to a month-calendar control.

Return Values

If the month delta previously was set using the **MonthCal_SetMonthDelta** macro, returns an INT value that represents the month-calendar's current scroll rate. If the month delta previously was not set using the **MonthCal_SetMonthDelta** macro, or the month delta was reset to the default, returns an INT value that represents the current number of months visible.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_GetMonthRange

Retrieves date information (using **SYSTEMTIME** structures) that represents the high and low limits of a month-calendar control's display. You can use this macro or send the **MCM_GETMONTHRANGE** message explicitly.

```
DWORD MonthCal_GetMonthRange(
  HWND hwndMC,
  DWORD dwFlag,
  LPSYSTEMTIME lprgSysTimeArray
);
```

Parameters

hwndMC
Handle to a month-calendar control.

dwFlag
Value specifying the scope of the range limits to be retrieved. This value must be one of the following:

GMR_DAYSTATE	Include preceding and trailing months of visible range that are only partially displayed.
GMR_VISIBLE	Include only those months that are entirely displayed.

lprgSysTimeArray
Address of a two-element array of **SYSTEMTIME** structures that will receive the lower and upper limits of the scope specified by *dwFlag*. The lower and upper limits are placed in lprgSysTimeArray[0] and lprgSysTimeArray[1], respectively. The time

members of these structures will not be modified. This parameter must be a valid address and cannot be NULL.

Return Values

Returns an INT value that represents the range, in months, spanned by the two limits returned at *lprgSysTimeArray*.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_GetRange

Retrieves the minimum and maximum allowable dates set for a month-calendar control. You can use this macro or send the **MCM_GETRANGE** message explicitly.

```
DWORD MonthCal_GetRange(
  HWND hwndMC,
  LPSYSTEMTIME lprgSysTimeArray
);
```

Parameters

hwndMC
Handle to a month-calendar control.

lprgSysTimeArray
Address of a two-element array of **SYSTEMTIME** structures that will receive the date limit information. The minimum limit is set in lprgSysTimeArray[0], and lprgSysTimeArray[1] receives the maximum limit. If either element is set to all zeros, then no corresponding limit is set for the month-calendar control. The time members of these structures will not be modified. This parameter must be a valid address and cannot be NULL.

Return Values

Returns a DWORD value that can be zero (no limits are set) or a combination of the following values that specify limit information:

GDTR_MAX There is a maximum limit set for the control; lprgSysTimeArray[0] is valid and contains the applicable date information.

GDTR_MIN There is a minimum limit set for the control; lprgSysTimeArray[1] is valid and contains the applicable date information.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_GetSelRange

Retrieves date information that represents the upper and lower limits of the date range currently selected by the user. You can use this macro or send the **MCM_GETSELRANGE** message explicitly.

```
BOOL MonthCal_GetSelRange(
  HWND hwndMC,
  LPSYSTEMTIME lprgSysTimeArray
);
```

Parameters

hwndMC
Handle to a month-calendar control.

lprgSysTimeArray
Address of a two-element array of **SYSTEMTIME** structures that will receive the lower and upper limits of the user's selection. The lower and upper limits are placed in lprgSysTimeArray[0] and lprgSysTimeArray[1], respectively. The time members of these structures will not be modified. This parameter must be a valid address and cannot be NULL.

Return Values

Returns nonzero if successful, or zero otherwise. **MonthCal_GetSelRange** will fail if applied to a month-calendar control that does not use the **MCS_MULTISELECT** style.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_GetToday

Retrieves the date information for the date specified as "today" for a month-calendar control. You can use this macro or send the **MCM_GETTODAY** message explicitly.

```
BOOL MonthCal_GetToday(
  HWND hwndMC,
  LPSYSTEMTIME lpToday
);
```

Parameters

hwndMC
Handle to a month-calendar control.

lpToday
Address of a **SYSTEMTIME** structure that will receive the date information. The time members of this structure will not be modified. This parameter must be a valid address and cannot be NULL.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_GetUnicodeFormat

Retrieves the UNICODE character format flag for the control. You can use this macro or send the **MCM_GETUNICODEFORMAT** message explicitly.

```
BOOL MonthCal_GetUnicodeFormat(
  HWND hwnd
);
```

Parameters

hwnd
Handle to the control.

Return Values

Returns the UNICODE format flag for the control. If this value is nonzero, the control is using UNICODE characters. If this value is zero, the control is using ANSI characters.

Requirements

Version 4.70 and later of Comctl32.dll/

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

See Also

MonthCal_SetUnicodeFormat

MonthCal_HitTest

Determines which portion of a month-calendar control is at a given point on the screen. You can use this macro or send the **MCM_HITTEST** message explicitly.

```
DWORD MonthCal_HitTest(
  HWND hwndMC,
  PMCHITTESTINFO pMCHitTest
);
```

Parameters

hwndMC
Handle to a month-calendar control.

pMCHitTest
Address of an **MCHITTESTINFO** structure. Upon calling the macro, the **cbSize** member must be set to the size of the **MCHITTESTINFO** structure, and **pt** must be set to the point you want to hit-test.

Return Values

Sets values in members of the **MCHITTESTINFO** structure at *pMCHitTest* and returns a DWORD value that contains a set of hit-test result flags. See the return value description of **MCM_HITTEST** for a list of the hit-test result flags.

The **uHit** member of the **MCHITTESTINFO** structure at *pMCHitTest* will equal the return value.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_SetColor

Sets the color for a given portion of a month-calendar control. You can use this macro or send the **MCM_SETCOLOR** message explicitly.

```
COLORREF MonthCal_SetColor(
  HWND hwndMC,
  INT iColor,
  COLORREF clr
);
```

Parameters

hwndMC
Handle to a month-calendar control.

iColor
INT value specifying which month-calendar color to set. This value can be one of the following:

MCSC_BACKGROUND	Retrieve the background color displayed between months.
MCSC_MONTHBK	Retrieve the background color displayed within the month.
MCSC_TEXT	Retrieve the color used to display text within a month.
MCSC_TITLEBK	Retrieve the background color displayed in the calendar's title.

MCSC_TITLETEXT	Retrieve the color used to display text within the calendar's title.
MCSC_TRAILINGTEXT	Retrieve the color used to display header day and trailing day text. Header and trailing days are the days from the previous and following months that appear on the current month-calendar.

clr
COLORREF value that represents the color that will be set for the specified area of the month-calendar.

Return Values
Returns a **COLORREF** value that represents the previous color setting for the specified portion of the month-calendar control, if successful. Otherwise, the return is –1.

Requirements
Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_SetCurSel

Sets the currently selected date for a month-calendar control. If the specified date is not in view, the control updates the display to bring it into view. You can use this macro or send the **MCM_SETCURSEL** message explicitly.

```
BOOL MonthCal_SetCurSel(
  HWND hwndMC,
  LPSYSTEMTIME lpSysTime
);
```

Parameters
hwndMC
Handle to a month-calendar control.

lpSysTime
Address of a **SYSTEMTIME** structure that contains the date to be set as the current selection. The time members of this structure are ignored.

Return Values

Returns nonzero if successful, or zero otherwise. This macro will fail if applied to a month-calendar control that uses the **MCS_MULTISELECT** style.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_SetDayState

Sets the day states for all months that are currently visible within a month-calendar control. You can use this macro or send the **MCM_SETDAYSTATE** message explicitly.

```
BOOL MonthCal_SetDayState(
  HWND hwndMC,
  INT iMonths,
  LPMONTHDAYSTATE lpDayStateArray
);
```

Parameters

hwndMC
Handle to a month-calendar control.

iMonths
INT value indicating how many elements are in the array that *lpDayStateArray* points to.

lpDayStateArray
Address of an array of **MONTHDAYSTATE** values that define how the month-calendar control will draw each day in its display.

Return Values

Returns nonzero if successful, or zero otherwise.

Remarks

The array at *lpDayStateArray* must contain as many elements as the value returned by the following macro:

```
MonthCal_GetMonthRange(hwndMC, GMR_DAYSTATE, NULL);
```

The preceding macro returns the total number of months that are in complete or partial view within the month-calendar's display.

Keep in mind that the array at *lpDayStateArray* must contain **MONTHDAYSTATE** values that correspond with all months currently in the control's display, in chronological order. This includes the two months only partially displayed before the first month and after the last month. For more information about preparing your array, see *Preparing the MONTHDAYSTATE Array.*

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_SetFirstDayOfWeek

Sets the first day of the week for a month-calendar control. You can use this macro or send the **MCM_SETFIRSTDAYOFWEEK** message explicitly.

```
DWORD MonthCal_SetFirstDayOfWeek(
  HWND hwndMC,
  INT iDay
);
```

Parameters

hwndMC
Handle to a month-calendar control.

iDay
INT value representing which day is to be set as the first day of the week. This value must be one of the day numbers.

Return Values

Returns a DWORD value that contains two values. The high word is a BOOL value that is nonzero if the previous first day of the week did not equal LOCALE_IFIRSTDAYOFWEEK, or zero otherwise. The low word is an INT value that represents the previous first day of the week.

Remarks

If the first day of the week is set to anything other than the default (LOCALE_IFIRSTDAYOFWEEK), the control will not update automatically first-day-of-the-week changes based on locale changes.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_SetMaxSelCount

Sets the maximum number of days that can be selected in a month-calendar control. You can use this macro or send the **MCM_SETMAXSELCOUNT** message explicitly.

```
BOOL MonthCal_SetMaxSelCount(
  HWND hwndMC,
  UINT iMax
);
```

Parameters

hwndMC
Handle to a month-calendar control.

iMax
INT value that will be set to represent the maximum number of days that can be selected.

Return Values

Returns nonzero if successful, or zero otherwise. This macro will fail if applied to a month-calendar control that does not use the **MCS_MULTISELECT** style.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).

Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_SetMonthDelta

Sets the scroll rate for a month-calendar control. The scroll rate is the number of months that the control moves its display when the user clicks a scroll button. You can use this macro or send the **MCM_SETMONTHDELTA** message explicitly.

```
INT MonthCal_SetMonthDelta(
  HWND hwndMC,
  INT iDelta
);
```

Parameters

hwndMC
Handle to a month-calendar control.

iDelta
Value representing the number of months to be set as the control's scroll rate. If this value is zero, the month delta is reset to the default, which is the number of months displayed in the control.

Return Values

Returns an INT value that represents the previous scroll rate. If the scroll rate was not previously set, the return value is zero.

Remarks

The PAGE UP and PAGE DOWN keys, VK_PRIOR and VK_NEXT, change the selected month by one, regardless of the number of months displayed or the value set by MCM_SETMONTHDELTA.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_SetRange

Sets the minimum and maximum allowable dates for a month-calendar control. You can use this macro or send the **MCM_SETRANGE** message explicitly.

```
BOOL MonthCal_SetRange(
  HWND hwndMC,
  DWORD fWhichLimit,
  LPSYSTEMTIME lprgSysTimeArray
);
```

Parameters

hwndMC
Handle to a month-calendar control.

fWhichLimit
Flag values that specify which date limits are being set. This value must be one or both of the following:

GDTR_MAX	The maximum allowable date is being set. The **SYSTEMTIME** structure at lprgSysTimeArray[1] must contain date information.
GDTR_MIN	The minimum allowable date is being set. The SYSTEMTIME structure at lprgSysTimeArray[0] must contain date information.

lprgSysTimeArray
Address of a two-element array of **SYSTEMTIME** structures that contain the date limits. The maximum limit must be in lpSysTimeArray[1] if GDTR_MAX is specified, and lpSysTimeArray[0] must contain the minimum limit if GDTR_MIN is specified.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_SetSelRange

Sets the selection for a month-calendar control to a given date range. You can use this macro or send the **MCM_SETSELRANGE** message explicitly.

```
BOOL MonthCal_SetSelRange(
  HWND hwndMC,
  LPSYSTEMTIME lprgSysTimeArray
);
```

Parameters

hwndMC
Handle to a month-calendar control.

lprgSysTimeArray
Address of a two-element array of **SYSTEMTIME** structures that contain date information representing the selection limits. The first selected date must be specified in lpSysTimeArray[0], and the last selected date must be specified in lpSysTimeArray[1]. The time members of these structures are ignored.

Return Values

Returns nonzero if successful, or zero otherwise. This macro will fail if applied to a month-calendar control that does not use the **MCS_MULTISELECT** style.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_SetToday

Sets the "today" selection for a month-calendar control. You can use this macro or send the **MCM_SETTODAY** message explicitly.

```
void MonthCal_SetToday(
  HWND hwndMC,
  LPSYSTEMTIME lpSysTime
);
```

Parameters

hwndMC

Handle to a month-calendar control.

lpSysTime

Address of a **SYSTEMTIME** structure that contains the date to be set as the "today" selection for the control. If this parameter is set to NULL, the control returns to the default setting. The time members of this structure are ignored.

If the "today" selection is set to any date other than the default, the following conditions apply:

- The control will not automatically update the "today" selection when the time passes midnight for the current day.
- The control will not automatically update its display based on locale changes.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

MonthCal_SetUnicodeFormat

Sets the UNICODE character format flag for the control. This message allows you to change the character set used by the control at run time instead of having to re-create the control. You can use this macro or send the **MCM_SETUNICODEFORMAT** message explicitly.

```
BOOL MonthCal_SetUnicodeFormat(
  HWND hwnd,
  BOOL fUnicode
);
```

Parameters

hwnd

Handle to the control.

fUnicode

Determines the character set that is used by the control. If this value is nonzero, the control will use UNICODE characters. If this value is zero, the control will use ANSI characters.

Return Values

Returns the previous UNICODE format flag for the control.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

See Also

MonthCal_GetUnicodeFormat

Month-Calendar Control Notifications

MCN_GETDAYSTATE

Sent by a month-calendar control to request information about how individual days should be displayed. This notification message is sent only by month-calendar controls that use the **MCS_DAYSTATE** style, and it is sent in the form of a **WM_NOTIFY** message.

```
MCN_GETDAYSTATE
    lpNMDayState = (LPNMDAYSTATE) lParam;
```

Parameters

lpNMDayState
Address of an **NMDAYSTATE** structure. The structure contains information about the time frame for which the control needs information, and it receives the address of an array that provides this data.

Remarks

Handling this notification message allows your application to customize its display by specifying that certain days be displayed in bold. For more information about processing this message, see *Processing the MCN_GETDAYSTATE Notification Message.*

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

MCN_SELCHANGE

Sent by a month-calendar control when the currently selected date or range of dates changes. This notification message is sent in the form of a **WM_NOTIFY** message.

```
MCN_SELCHANGE
    lpNMSelChange = (LPNMSELCHANGE) lParam;
```

Parameters

lpNMSelChange
Address of an **NMSELCHANGE** structure that contains information about the currently selected date range.

Remarks

For example, the control sends MCN_SELCHANGE when the user explicitly changes the selection within the current month, or when the selection is implicitly changed in response to next/previous month exploration.

This notification message is similar to **MCN_SELECT**, but it is sent in response to any selection change. **MCN_SELECT** is sent only for an explicit date selection.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

MCN_SELECT

Sent by a month-calendar control when the user makes an explicit date selection within a month-calendar control. This notification is sent in the form of a **WM_NOTIFY** message.

```
MCN_SELECT
    lpNMSelChange = (LPNMSELCHANGE) lParam;
```

Parameters

lpNMSelChange

Address of an **NMSELCHANGE** structure that contains information about the currently selected date range.

Remarks

This notification message is similar to **MCN_SELCHANGE**, but it is sent only in response to a user's explicit date selections. **MCN_SELCHANGE** applies to any selection change.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

NM_RELEASEDCAPTURE (monthcal)

Notifies a monthcal control's parent window that the control is releasing mouse capture. This notification is sent in the form of a **WM_NOTIFY** message.

```
NM_RELEASEDCAPTURE
    lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh

Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The control ignores the return value from this notification.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

Month-Calendar Control Structures

MCHITTESTINFO

Carries information specific to hit-testing points for a month-calendar control. This structure is used with the **MCM_HITTEST** message and the corresponding **MonthCal_HitTest** macro.

```
typedef struct {
    UINT        cbSize;
    POINT       pt;
    UINT        uHit;   // An output member
    SYSTEMTIME st;      // An output member
} MCHITTESTINFO, *PMCHITTESTINFO;
```

Members

cbSize
Size of this structure, in bytes.

pt
POINT structure that contains information about the point to be hit-tested.

uHit
Output member that receives a bit flag representing the result of the hit-test operation. This value will be one of the following:

MCHT_CALENDARBK	The given point was in the calendar's background.
MCHT_CALENDARDATE	The given point was on a particular date within the calendar. The **SYSTEMTIME** structure at *lpMCHitTest*->st is set to the date at the given point.
MCHT_CALENDARDATENEXT	The given point was over a date from the next month (partially displayed at the end of the currently displayed month). If the user clicks here, the month-calendar will scroll its display to the next month or set of months.

MCHT_CALENDARDATEPREV	The given point was over a date from the previous month (partially displayed at the end of the currently displayed month). If the user clicks here, the month-calendar will scroll its display to the previous month or set of months.
MCHT_CALENDARDAY	The given point was over a day abbreviation ("Fri", for example). The **SYSTEMTIME** structure at *lpMCHitTest*->st is set to the corresponding date in the top row.
MCHT_CALENDARWEEKNUM	The given point was over a week number (**MCS_WEEKNUMBERS** style only). The **SYSTEMTIME** structure at *lpMCHitTest*->st is set to the corresponding date in the leftmost column.
MCHT_NOWHERE	The given point was not on the month-calendar control, or it was in an inactive portion of the control.
MCHT_TITLEBK	The given point was over the background of a month's title.
MCHT_TITLEBTNNEXT	The given point was over the button at the top-right corner of the control. If the user clicks here, the month-calendar will scroll its display to the next month or set of months.
MCHT_TITLEBTNPREV	The given point was over the button at the top-left corner of the control. If the user clicks here, the month-calendar will scroll its display to the previous month or set of months.
MCHT_TITLEMONTH	The given point was in a month's title bar, over a month name.
MCHT_TITLEYEAR	The given point was in a month's title bar, over the year value.

st

SYSTEMTIME structure that receives date and time information specific to the location that was hit-tested.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).

Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

NMDAYSTATE

Carries information required to process the **MCN_GETDAYSTATE** notification message. All members of this structure are for input, except **prgDayState**, which the receiving application must set when processing **MCN_GETDAYSTATE**.

```
typedef struct tagNMDAYSTATE {
    NMHDR            nmhdr;
    SYSTEMTIME       stStart;
    int              cDayState;
    LPMONTHDAYSTATE prgDayState;
} NMDAYSTATE, FAR * LPNMDAYSTATE;
```

Members

nmhdr
NMHDR structure that contains information about this notification message.

stStart
SYSTEMTIME structure that contains the starting date.

cDayState
INT value specifying the total number of elements that must be in the array at **prgDayState**.

prgDayState
Address of an array of **MONTHDAYSTATE** values. The buffer at this address must be large enough to contain at least **cDayState** elements. The first element in the array corresponds to the date in **stStart**.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

NMSELCHANGE

Carries information required to process the **MCN_SELCHANGE** notification message.

```
typedef struct tagNMSELCHANGE{
    NMHDR           nmhdr;
    SYSTEMTIME      stSelStart;
    SYSTEMTIME      stSelEnd;
} NMSELCHANGE, FAR * LPNMSELCHANGE;
```

Members

nmhdr
NMHDR structure that contains information about this notification message.

stSelStart
SYSTEMTIME structure that contains the date for the first day in the user's selection range.

stSelEnd
SYSTEMTIME structure that contains the date for the last day in the user's selection range.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorers 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

Month-Calendar Control Data Types

MONTHDAYSTATE

This is a new data type that is defined in Commctrl.h as follows:

```
typedef DWORD MONTHDAYSTATE, FAR * LPMONTHDAYSTATE;
```

The MONTHDAYSTATE type is a bit field, where each bit (1 through 31) represents the state of a day in a month. If a bit is on, the corresponding day will be displayed in bold; otherwise, it will be displayed with no emphasis.

This data type is used with the **MCM_SETDAYSTATE** message and the corresponding macro, **MonthCal_SetDayState**. When MONTHDAYSTATE values are used in reference to months shorter than 31 days, only the needed bits will be accessed.

CHAPTER 19

Pager Controls

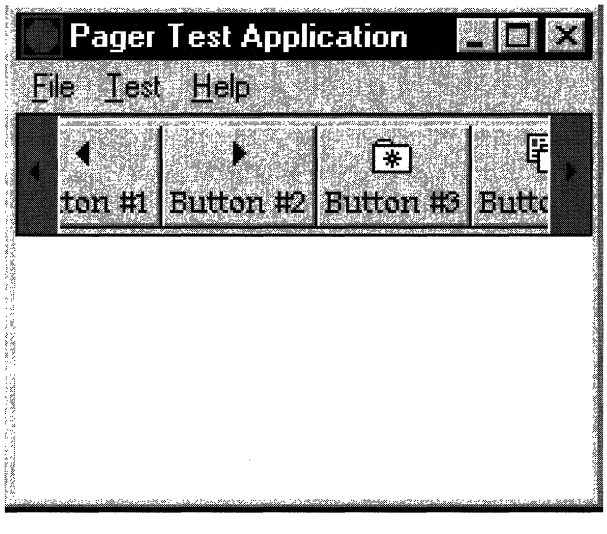
A *pager control* is a window container that is used with a window that does not have enough display area to show all of its content. The pager control allows the user to scroll to the area of the window that is not currently in view.

About Pager Controls

Microsoft Internet Explorer Version 4.0 (commctrl.dll version 4.71) introduces the pager control. This control is useful in situations where a window does not have enough area to display a child window. For example, if your application has a toolbar that is not wide enough to show all of its items, you can assign the toolbar to a pager control and users will be able to scroll to the left or right to access all of the items. You can also create pager controls that scroll vertically.

A window assigned to the pager control is referred to as the *contained window*.

The following illustration shows a toolbar contained inside of a pager control. The pager control is shaded to show which areas of the control are visible.



Note The pager control is implemented in version 4.71 and later of Comctl32.dll.

Using Pager Controls

This section describes how to implement the pager control in your application.

Initializing the Pager Control

To use the pager control, you must call **InitCommonControlsEx** with the ICC_PAGESCROLLER_CLASS flag set in the **dwICC** member of the **INITCOMMONCONTROLSEX** structure.

Creating the Pager Control

Use the **CreateWindow** or the **CreateWindowEx** API to create a pager control. The class name for the control is WC_PAGESCROLLER, which is defined in Commctrl.h. The **PGS_HORZ** style is used to create a horizontal pager, and the **PGS_VERT** style is used to create a vertical pager. Because this is a child control, the WS_CHILD style should also be used.

Once the pager control is created, you will most likely want to assign a contained window to it. If the contained window is a child window, you should make the child window a child of the pager control so that the size and position will be calculated correctly. You then assign the window to the pager control with the **PGM_SETCHILD** message. Be aware that this message does not actually change the parent window of the contained window; it simply assigns the contained window. If the contained window is one of the common controls, it must have the CCS_NORESIZE style to prevent the control from attempting to resize itself to the pager control's size.

Processing Pager Control Notifications

At a minimum, it is necessary to process the **PGN_CALCSIZE** notification. If you don't process this notification and enter a value for the width or height, the scroll arrows in the pager control will not be displayed. This is because the pager control uses the width or height supplied in the **PGN_CALCSIZE** notification to determine the "ideal" size of the contained window.

The following example demonstrates how to process the **PGN_CALCSIZE** notification case. In this example, the contained window is a toolbar control that contains an unknown number of buttons at an unknown size. The example shows how to use the **TB_GETMAXSIZE** message to determine the size of all of the items in the toolbar. The example then places the width of all of the items into the **iWidth** member of the **NMPGCALCSIZE** structure passed to the notification.

```
case PGN_CALCSIZE:
  {
  LPNMPGCALCSIZE   pCalcSize = (LPNMPGCALCSIZE)lParam;

  switch(pCalcSize->dwFlag)
```

```
    {
    case PGF_CALCWIDTH:
       {
       SIZE  size;

       //Get the optimum width of the toolbar.
       SendMessage(hwndToolbar, TB_GETMAXSIZE, 0, (LPARAM)&size);
       pCalcSize->iWidth = size.cx;
       }
       break;

    }

 }
 return 0;
```

Processing the **PGN_SCROLL** notification is optional. Process this notification if you need to know when a scroll action occurs, need to track the scroll position, or wish to change the scroll delta. To cancel the scroll, simply place zero in the **iScroll** member of the **NMPGSCROLL** structure passed to the notification.

The following example shows how to modify the scroll delta:

```
case PGN_SCROLL:
 {
 LPNMPGSCROLL   pScroll = (LPNMPGSCROLL)lParam;

 switch(pScroll->iDir)
    {
    case PGF_SCROLLLEFT:
    case PGF_SCROLLRIGHT:
    case PGF_SCROLLUP:
    case PGF_SCROLLDOWN:
       pScroll->iScroll = 20;
       break;
    }
 }
 return 0;
```

Pager Control Styles

The following window styles are used when creating pager controls:

PGS_AUTOSCROLL The pager control will scroll when the user hovers the mouse over one of the scroll buttons.

(continued)

(continued)

PGS_DRAGNDROP	The contained window can be a drag-and-drop target. The pager control will automatically scroll if an item is dragged from outside the pager over one of the scroll buttons.
PGS_HORZ	Creates a pager control that can be scrolled horizontally. This style and the **PGS_VERT** style are mutually exclusive and cannot be combined.
PGS_VERT	Creates a pager control that can be scrolled vertically. This is the default direction if no direction style is specified. This style and the **PGS_HORZ** style are mutually exclusive and cannot be combined.

Pager Control Reference

Pager Control Messages

PGM_FORWARDMOUSE

Enables or disables mouse forwarding for the pager control. When mouse forwarding is enabled, the pager control forwards **WM_MOUSEMOVE** messages to the contained window. You can send this message explicitly or use the **Pager_ForwardMouse** macro.

```
PGM_FORWARDMOUSE
    wParam = (WPARAM)(BOOL)bForward;
    lParam = 0;
```

Parameters

bForward

BOOL value that determines if mouse forwarding is enabled or disabled. If this value is nonzero, mouse forwarding is enabled. If this value is zero, mouse forwarding is disabled.

Return Values

The return value is not used.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

PGM_GETBKCOLOR

Retrieves the current background color for the pager control. You can send this message explicitly or use the **Pager_GetBkColor** macro.

```
PGM_GETBKCOLOR
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a **COLORREF** value that contains the current background color.

Remarks

By default, the pager control will use the system button face color as the background color. This is the same color that can be retrieved by calling **GetSysColor** with COLOR_BTNFACE.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

PGM_GETBORDER

Retrieves the current border size for the pager control. You can send this message explicitly or use the **Pager_GetBorder** macro.

```
PGM_GETBORDER
    wParam = 0;
    lParam = 0;
```

Return Values

Returns an INT value that contains the current border size, in pixels.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

See Also

PGM_SETBORDER

PGM_GETBUTTONSIZE

Retrieves the current button size for the pager control. You can send this message explicitly or use the **Pager_GetButtonSize** macro.

```
PGM_GETBUTTONSIZE
    wParam = 0;
    lParam = 0;
```

Return Values

Returns an INT value that contains the current button size, in pixels.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

See Also

PGM_SETBUTTONSIZE

PGM_GETBUTTONSTATE

Retrieves the state of the specified button in a pager control. You can send this message explicitly or use the **Pager_GetButtonState** macro.

```
PGM_GETBUTTONSTATE
    wParam = 0;
    lParam = (LPARAM)(int)iButton;
```

Parameters

iButton

Indicates which button to retrieve the state for. This can be one of the following values:

PGB_TOPORLEFT	Indicates the top button in a **PGS_VERT** pager control or the left button in a **PGS_HORZ** pager control.
PGB_BOTTOMORRIGHT	Indicates the bottom button in a **PGS_VERT** pager control or the right button in a **PGS_HORZ** pager control.

Return Values

Returns the state of the button specified in *iButton*. This will be one of the following values:

PGF_INVISIBLE	The button is not visible.
PGF_NORMAL	The button is in normal state.
PGF_GRAYED	The button is in grayed state.
PGF_DEPRESSED	The button is in pressed state.
PGF_HOT	The button is in hot state.

Version 4.71 and later of Comctl32.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 commctrl.h.

PGM_GETDROPTARGET

Retrieves a pager control's **IDropTarget** interface pointer. You can send this message explicitly or use the **Pager_GetDropTarget** macro.

```
PGM_GETDROPTARGET
    wParam = 0;
    lParam = (IDropTarget**)ppDropTarget;
```

Parameters

ppDropTarget
Address of an **IDropTarget** pointer that receives the interface pointer. It is the caller's responsibility to call **Release** on this pointer when it is no longer needed.

Return Values

The return value for this message is not used.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

PGM_GETPOS

Retrieves the current scroll position of the pager control. You can send this message explicitly or use the **Pager_GetPos** macro.

```
PGM_GETPOS
    wParam = 0;
    lParam = 0;
```

Return Values

Returns an INT value that contains the current scroll position, in pixels.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

PGM_RECALCSIZE

Forces the pager control to recalculate the size of the contained window. Sending this message will result in a **PGN_CALCSIZE** notification being sent. You can send this message explicitly or use the **Pager_RecalcSize** macro.

```
PGM_RECALCSIZE
    wParam = 0;
    lParam = 0;
```

Return Values

The return value is not used.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

PGM_SETBKCOLOR

Sets the current background color for the pager control. You can send this message explicitly or use the **Pager_SetBkColor** macro.

```
PGM_SETBKCOLOR
    wParam = 0;
    lParam = (LPARAM)(COLORREF)clrBk;
```

Parameters

clrBk

COLORREF value that contains the new background color of the pager control.

Return Values

Returns a **COLORREF** value that contains the previous background color.

Remarks

By default, the pager control will use the system button face color as the background color. This is the same color that can be retrieved by calling **GetSysColor** with COLOR_BTNFACE.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

PGM_SETBORDER

Sets the current border size for the pager control. You can send this message explicitly or use the **Pager_SetBorder** macro.

```
PGM_SETBORDER
    wParam = 0;
    lParam = (LPARAM)(int)iBorder;
```

Parameters

iBorder
New size of the border, in pixels. This value should not be larger than the pager button or less than zero. If *iBorder* is too large, the border will be drawn the same size as the button. If *iBorder* is negative, the border size will be set to zero.

Return Values

Returns an INT value that contains the previous border size, in pixels.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

PGM_SETBUTTONSIZE

Sets the current button size for the pager control. You can send this message explicitly or use the **Pager_SetButtonSize** macro.

```
PGM_SETBUTTONSIZE
    wParam = 0;
    lParam = (LPARAM)(int)iButtonSize;
```

Parameters

iButtonSize

INT value that contains the new button size, in pixels.

Return Values

Returns an INT value that contains the previous button size, in pixels.

Remarks

If the pager control has the **PGS_HORZ** style, the button size determines the width of the pager buttons. If the pager control has the **PGS_VERT** style, the button size determines the height of the pager buttons. By default, the pager control sets its button size to three-fourths of the width of the scroll bar.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

See Also

PGM_GETBUTTONSIZE

PGM_SETCHILD

Sets the contained window for the pager control. This message will not change the parent of the contained window; it only assigns a window handle to the pager control for scrolling. In most cases, the contained window will be a child window. If this is the case, the contained window should be a child of the pager control. You can send this message explicitly or use the **Pager_SetChild** macro.

```
PGM_SETCHILD
    wParam = 0;
    lParam = (LPARAM)(HWND)hwndChild;
```

Parameters

hwndChild

Handle to the window to be contained.

Return Values

The return value is not used.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

PGM_SETPOS

Sets the current scroll position for the pager control. You can send this message explicitly or use the **Pager_SetPos** macro.

```
PGM_SETPOS
    wParam = 0;
    lParam = (LPARAM)(int)iPos;
```

Parameters

iPos

INT value that contains the new scroll position, in pixels.

Return Values

The return value is not used.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

Pager Control Macros

Pager_ForwardMouse

Enables or disables mouse forwarding for the pager control. When mouse forwarding is enabled, the pager control forwards **WM_MOUSEMOVE** messages to the contained window. You can use this macro or send the **PGM_FORWARDMOUSE** message explicitly.

```
VOID Pager_ForwardMouse(
  HWND hwndPager,
  BOOL bForward
);
```

Parameters

hwndPager
Handle to the pager control.

bForward
BOOL value that determines if mouse forwarding is enabled or disabled. If this value is nonzero, mouse forwarding is enabled. If this value is zero, mouse forwarding is disabled.

Return Values

The return value is not used.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

Pager_GetBkColor

Retrieves the current background color for the pager control. You can use this macro or send the **PGM_GETBKCOLOR** message explicitly.

```
COLORREF Pager_GetBkColor(
  HWND hwndPager
);
```

Parameters

hwndPager
Handle to the pager control.

Return Values

Returns a **COLORREF** value that contains the current background color.

Remarks

By default, the pager control will use the system button face color as the background color. This is the same color that can be retrieved by calling **GetSysColor** with COLOR_BTNFACE.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

Pager_GetBorder

Retrieves the current border size for the pager control. You can use this macro or send the **PGM_GETBORDER** message explicitly.

```
int Pager_GetBorder(
  HWND hwndPager
);
```

Parameters

hwndPager
Handle to the pager control.

Return Values

Returns an INT value that contains the current border size, in pixels.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

Pager_GetButtonSize

Retrieves the current button size for the pager control. You can use this macro or send the **PGM_GETBUTTONSIZE** message explicitly.

```
int Pager_GetButtonSize(
  HWND hwndPager
);
```

Parameters

hwndPager
Handle to the pager control.

Return Values

Returns an INT value that contains the current button size, in pixels.

See Also

Pager_SetButtonSize

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

Pager_GetButtonState

Retrieves the state of the specified button in a pager control. You can use this macro or send the **PGM_GETBUTTONSTATE** message explicitly.

```
DWORD Pager_GetButtonState(
  HWND hwndPager;
  int iButton
);
```

Parameters

hwndPager
Handle to the pager control.

iButton
Indicates which button to retrieve the state for. See the description for *iButton* in **PGM_GETBUTTONSTATE** for a list of possible values.

Return Values

Returns the state of the button specified in *iButton*. See the return value description in **PGM_GETBUTTONSTATE** for a list of possible values.

Version 4.71 and later of Comctl32.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 commctrl.h.

Pager_GetDropTarget

Retrieves a pager control's **IDropTarget** interface pointer. You can use this macro or send the **PGM_GETDROPTARGET** message explicitly.

```
void Pager_GetDropTarget(
  HWND hwndPager,
  IDropTarget **ppDropTarget
);
```

Parameters

hwndPager
Handle to the pager control.

ppDropTarget
Address of an **IDropTarget** pointer that receives the interface pointer. It is the caller's responsibility to call **Release** on this pointer when it is no longer needed.

Version 4.71 and later of Comctl32.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 commctrl.h.

Pager_GetPos

Retrieves the current scroll position of the pager control. You can use this macro or send the **PGM_GETPOS** message explicitly.

```
COLORREF Pager_GetPos(
  HWND hwndPager
);
```

Parameters

hwndPager
Handle to the pager control.

Return Values

Returns an INT value that contains the current scroll position, in pixels.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

Pager_RecalcSize

Forces the pager control to recalculate the size of the contained window. Using this macro will result in a **PGN_CALCSIZE** notification being sent. You can use this macro or send the **PGM_RECALCSIZE** message explicitly.

```
COLORREF Pager_RecalcSize(
  HWND hwndPager
);
```

Parameters

hwndPager
Handle to the pager control.

Return Values

The return value is not used.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

Pager_SetBkColor

Sets the current background color for the pager control. You can use this macro or send the **PGM_SETBKCOLOR** message explicitly.

```
COLORREF Pager_SetBkColor(
  HWND hwndPager,
  COLORREF clrBk
);
```

Parameters

hwndPager
Handle to the pager control.

clrBk
COLORREF value that contains the new background color of the pager control.

Return Values

Returns a **COLORREF** value that contains the previous background color.

Remarks

By default, the pager control will use the system button face color as the background color. This is the same color that can be retrieved by calling **GetSysColor** with COLOR_BTNFACE.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

Pager_SetBorder

Sets the current border size for the pager control. You can use this macro or send the **PGM_SETBORDER** message explicitly.

```
INT Pager_SetBorder(
  HWND hwndPager,
  int iBorder
);
```

Parameters

hwndPager
Handle to the pager control.

iBorder
New size of the border, in pixels. This value should not be larger than the pager button or less than zero. If *iBorder* is too large, the border will be drawn the same size as the button. If *iBorder* is negative, the border size will be set to zero.

Return Values

Returns an INT value that contains the previous border size, in pixels.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

Pager_SetButtonSize

Sets the current button size for the pager control. You can use this macro or send the **PGM_SETBUTTONSIZE** message explicitly.

```
int Pager_SetButtonSize(
  HWND hwndPager,
  Int iButtonSize
);
```

Parameters

hwndPager
Handle to the pager control.

iButtonSize
INT value that contains the new button size, in pixels.

Return Values

Returns an INT value that contains the previous button size, in pixels.

Remarks

If the pager control has the **PGS_HORZ** style, the button size determines the width of the pager buttons. If the pager control has the **PGS_VERT** style, the button size determines the height of the pager buttons. By default, the pager control sets its button size to three-fourths of the width of the scroll bar.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

See Also

Pager_GetButtonSize

Pager_SetChild

Sets the contained window for the pager control. This macro will not change the parent of the contained window; it only assigns a window handle to the pager control for

scrolling. In most cases, the contained window will be a child window. If this is the case, the contained window should be a child of the pager control. You can use this macro or send the **PGM_SETCHILD** message explicitly.

```
COLORREF Pager_SetChild(
  HWND hwndPager,
  HWND hwndChild
);
```

Parameters

hwndPager
Handle to the pager control.

hwndChild
Handle to the window to be contained.

Return Values

The return value is not used.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

Pager_SetPos

Sets the scroll position for the pager control. You can use this macro or send the **PGM_SETPOS** message explicitly.

```
INT Pager_SetPos(
  HWND hwndPager,
  int iPos
);
```

Parameters

hwndPager
Handle to the pager control.

iPos
INT value that contains the new scroll position, in pixels.

Return Values

The return value is not used.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

Pager Control Notifications

NM_RELEASEDCAPTURE (pager)

Notifies a pager control's parent window that the control has released the mouse capture. NM_RELEASEDCAPTURE is sent in the form of a **WM_NOTIFY** message.

```
NM_RELEASEDCAPTURE
    lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh
Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The return value is ignored by the pager control.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

PGN_CALCSIZE

Notification sent by a pager control to obtain the scrollable dimensions of the contained window. These dimensions are used by the pager control to determine the scrollable size of the contained window. This notification is sent in the form of a **WM_NOTIFY** message.

```
PGN_CALCSIZE
    lpnmcs = (LPNMPGCALCSIZE) lParam;
```

Parameters

lpnmcs

Address of an **NMPGCALCSIZE** structure that contains and receives information about the notification. The **dwFlag** member of this structure indicates which dimension is being calculated. Depending on the value of **dwFlags**, you should place the desired dimension in the **iWidth** or **iHeight** member of this structure.

Return Values

The return value is ignored.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

PGN_SCROLL

Notification sent by a pager control prior to the contained window being scrolled. This notification is sent in the form of a **WM_NOTIFY** message.

```
PGN_SCROLL
    lpnms = (LPNMPGSCROLL) lParam;
```

Parameters

lpnms

Address of an **NMPGSCROLL** structure that contains and receives information about the notification. The **iDir** member of this structure indicates the direction of the scroll. The **iXpos** and **iYpos** members contain the horizontal and vertical position of the

contained window prior to the scroll. The **iScroll** member contains the default scroll delta amount. You can modify this member to a different scroll amount if desired.

Return Values

The return value is ignored.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

Pager Control Structures

NMPGCALCSIZE

Contains and receives information that the pager control uses to calculate the scrollable area of the contained window. It is used with the **PGN_CALCSIZE** notification.

```
typedef struct {
    NMHDR   hdr;
    DWORD   dwFlag;
    int     iWidth;
    int     iHeight;
}NMPGCALCSIZE, *LPNMPGCALCSIZE;
```

Members

hdr

NMHDR structure that contains information about the notification message.

dwFlag

Value that indicates which dimension is being requested. This will be one of the following values:

PGF_CALCHEIGHT	The height of the scrollable area is being requested. The height must be placed in the **iHeight** member before returning from the notification.
PGF_CALCWIDTH	The width of the scrollable area is being requested. The width must be placed in the **iWidth** member before returning from the notification.

iWidth
Receives the desired width of the scrollable area, in pixels.

iHeight
Receives the desired height of the scrollable area, in pixels.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

NMPGSCROLL

Contains and receives information that the pager control uses when scrolling the contained window. It is used with the **PGN_SCROLL** notification.

```
typedef struct {
    NMHDR hdr;
    BOOL fwKeys;
    RECT rcParent;
    int iDir;
    int iXpos;
    int iYpos;
    int iScroll;
}NMPGSCROLL, *LPNMPGSCROLL;
```

Members

hdr
NMHDR structure that contains information about the notification message.

fwKeys
Modifier keys that are down when the scroll occurs. This can be one or more of the following values:

0	None of the modifier keys are down.
PGK_SHIFT	The SHIFT key is down.
PGK_CONTROL	The CONTROL key is down.
PGK_MENU	The ALT key is down.

rcParent
Contains the client rectangle of the pager control.

iDir
Value that indicates in which direction the scroll is occurring. This will be one of the following values:

PGF_SCROLLDOWN	The contained window is being scrolled down.
PGF_SCROLLLEFT	The contained window is being scrolled to the left.
PGF_SCROLLRIGHT	The contained window is being scrolled to the right.
PGF_SCROLLUP	The contained window is being scrolled up.

iXpos
Contains the horizontal scroll position of the contained window, in pixels, before the scroll action.

iYpos
Contains the vertical scroll position of the contained window, in pixels, before the scroll action.

iScroll
On entry, contains the default scroll delta in pixels. This member can be modified to contain a different scroll delta amount if desired. This value is always positive, regardless of the scroll direction.

Requirements

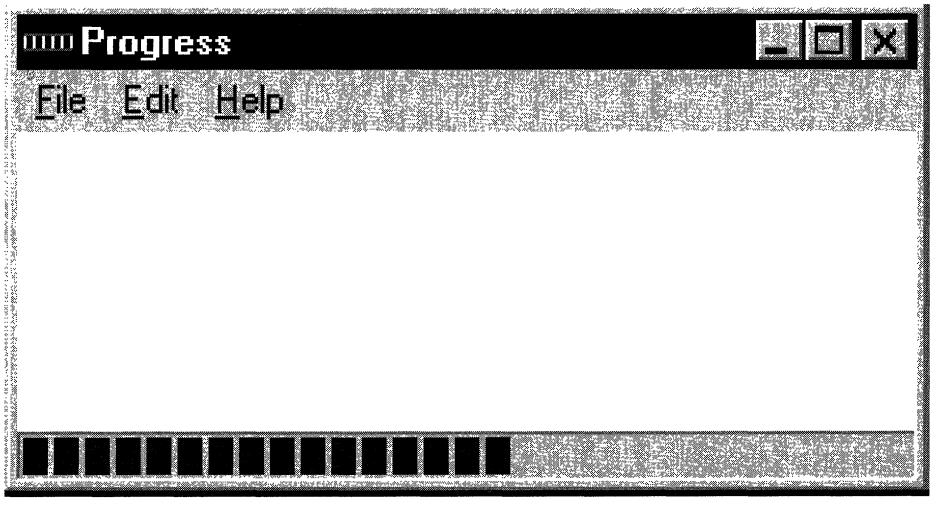
Version 4.71 and later of Comctl32.dll.

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 commctrl.h.

CHAPTER 20

Progress Bar Controls

A *progress bar* is a window that an application can use to indicate the progress of a lengthy operation. It consists of a rectangle that is gradually filled with the system highlight color as an operation progresses. The following illustration shows a progress bar positioned along the bottom of a window.



Using Progress Bars

You create a progress bar by using the **CreateWindowEx** function, specifying the PROGRESS_CLASS window class. This window class is registered when the common control dynamic-link library (DLL) is loaded. To ensure that this DLL is loaded, use the **InitCommonControls** function first.

Range and Current Position

A progress bar's *range* represents the entire duration of the operation, and the *current position* represents the progress that the application has made toward completing the operation. The window procedure uses the range and the current position to determine the percentage of the progress bar to fill with the highlight color. Because the range and current position values are expressed as unsigned integers, the highest possible range or current position value is 65,535.

The minimum value in the range can be from 0 to 65,535. Likewise, the maximum value can be from 0 to 65,535. If you do not set the range values, the system sets the

minimum value to 0 and the maximum value to 100. You can adjust the range to convenient integers by using the **PBM_SETRANGE** message.

A progress bar provides several messages that you can use to set the current position. The **PBM_SETPOS** message sets the position to a given value. The **PBM_DELTAPOS** message advances the position by adding a specified value to the current position.

The **PBM_SETSTEP** message allows you to specify a step increment for a progress bar. Subsequently, whenever you send the **PBM_STEPIT** message to the progress bar, the current position advances by the specified increment. By default, the step increment is set to 10.

Default Progress Bar Message Processing

This section describes the messages handled by the window procedure for the PROGRESS_CLASS class.

Message	Processing performed
WM_CREATE	Allocates and initializes an initial structure.
WM_DESTROY	Frees all resources associated with the progress bar.
WM_ERASEBKGND	Draws the background and borders of the progress bar.
WM_GETFONT	Returns the handle to the current font. The progress bar does not currently draw text, so sending this message has no effect on the control.
WM_PAINT	Draws the progress bar. If the *wParam* parameter is non-NULL, the control assumes that the value is an HDC and paints using that device context.
WM_SETFONT	Saves the handle to the new font and returns the handle to the previous font. The progress bar does not currently draw text, so sending this message has no effect on the control.

Progress Bar Example

The following example shows how to use a progress bar to indicate the progress of a lengthy file-parsing operation. The example creates a progress bar and positions it along the bottom of the parent window's client area. The height of the progress bar is based on the height of the arrow bitmap used in a scroll bar. The range is based on the size of the file divided by 2048, which is the size of each "chunk" of data read from the file. The example also sets an increment and advances the current position of the progress bar by the increment after parsing each chunk.

```
// ParseALargeFile - parses a large file and uses a progress bar to
//    indicate the progress of the parsing operation.
// Returns TRUE if successful, or FALSE otherwise.
// hwndParent - parent window of the progress bar.
// lpszFileName - name of the file to parse.
```

```
//
// Global variable
//     g_hinst - instance handle
extern HINSTANCE g_hinst;

BOOL ParseALargeFile(HWND hwndParent, LPSTR lpszFileName)
{
    RECT rcClient;  // client area of parent window
int cyVScroll;  // height of scroll bar arrow
HWND hwndPB;    // handle of progress bar
HANDLE hFile;   // handle of file
DWORD cb;       // size of file and count of bytes read
LPCH pch;       // address of data read from file
LPCH pchTmp;    // temporary pointer

// Ensure that the common control DLL is loaded and create a
// progress bar along the bottom of the client area of the
// parent window. Base the height of the progress bar on
// the height of a scroll bar arrow.
InitCommonControls();
GetClientRect(hwndParent, &rcClient);
cyVScroll = GetSystemMetrics(SM_CYVSCROLL);
hwndPB = CreateWindowEx(0, PROGRESS_CLASS, (LPSTR) NULL,
    WS_CHILD | WS_VISIBLE, rcClient.left,
    rcClient.bottom - cyVScroll,
    rcClient.right, cyVScroll,
    hwndParent, (HMENU) 0, g_hinst, NULL);

// Open the file for reading, and retrieve the size of the file.
hFile = CreateFile(lpszFileName, GENERIC_READ, FILE_SHARE_READ,
    (LPSECURITY_ATTRIBUTES) NULL, OPEN_EXISTING,
    FILE_ATTRIBUTE_NORMAL, (HANDLE) NULL);

if (hFile == (HANDLE) INVALID_HANDLE_VALUE)
    return FALSE;

cb = GetFileSize(hFile, (LPDWORD) NULL);

// Set the range and increment of the progress bar.
SendMessage(hwndPB, PBM_SETRANGE, 0, MAKELPARAM(0, cb / 2048));
SendMessage(hwndPB, PBM_SETSTEP, (WPARAM) 1, 0);

// Parse the file.
pch = (LPCH) LocalAlloc(LPTR, sizeof(char) * 2048);
pchTmp = pch;
```

(continued)

(continued)

```
    do {
        ReadFile(hFile, pchTmp, sizeof(char) * 2048, &cb,
            (LPOVERLAPPED) NULL);
        .
        . // Include here any code that parses the file.
        .

        // Advance the current position of the progress bar
        // by the increment.
        SendMessage(hwndPB, PBM_STEPIT, 0, 0);
    } while (cb);
    CloseHandle((HANDLE) hFile);

    DestroyWindow(hwndPB);
    return TRUE;
    }
```

Progress Bar Control Updates in Internet Explorer

Progress bar controls in Microsoft Internet Explorer support the following new features:

New Progress Bar Control Styles
Progress bar controls can now display progress information vertically, using the **PBS_VERTICAL** style. Also, a smooth progress mode is supported using the **PBS_SMOOTH** style. Using the **PBS_SMOOTH** style causes the control to display a contiguous progress bar instead of a segmented bar.

Extended Range Values
Progress bar controls now support 32-bit range values. To set range values in excess of 65,535, use the **PBM_SETRANGE32** message. To retrieve 32-bit range values, use the **PBM_GETRANGE** message. The progress bar high limit, low limit, and position parameters are signed integers. To make full use of the 32-bit range, set the range to –0x7FFFFFFF to 0x7FFFFFFF and treat the position as a signed integer.

Programmable Colors
An application can now control the colors used in a progress bar control with the **PBM_SETBARCOLOR** and **PBM_SETBKCOLOR** messages.

Progress Bar Control Styles

Progress bar controls now support control styles. You can set progress bar styles in the same way as other common controls (**CreateWindowEx**, **GetWindowLong**, **SetWindowLong**). The following are the supported styles:

PBS_SMOOTH	**Version 4.70**. The progress bar displays progress status in a smooth scrolling bar instead of the default segmented bar.
PBS_VERTICAL	**Version 4.70**. The progress bar displays progress status vertically, from bottom to top.

Progress Bar Control Reference

Progress Bar Control Messages

PBM_DELTAPOS

Advances the current position of a progress bar by a specified increment and redraws the bar to reflect the new position.

```
PBM_DELTAPOS
    wParam = (WPARAM) nIncrement;
    lParam = 0;
```

Parameters

nIncrement
Amount to advance the position.

Return Values

Returns the previous position.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

PBM_GETPOS

Retrieves the current position of the progress bar.

```
PBM_GETPOS
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a UINT value that represents the current position of the progress bar.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

PBM_GETRANGE

Retrieves information about the current high and low limits of a given progress bar control.

```
PBM_GETRANGE
    wParam = (WPARAM)(BOOL) fWhichLimit;
    lParam = (LPARAM)(PPBRANGE) ppBRange;
```

Parameters

fWhichLimit

Flag value specifying which limit value is to be used as the message's return value. This parameter can be one of the following values:

TRUE	Return the low limit.
FALSE	Return the high limit.

ppBRange

Address of a **PBRANGE** structure that is to be filled with the high and low limits of the progress bar control. If this parameter is set to NULL, the control will return only the limit specified by *fWhichLimit*.

Return Values

Returns an INT that represents the limit value specified by *fWhichLimit*. If lParam is not NULL, lParam must point to a **PBRANGE** structure that is to be filled with both limit values.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

PBM_SETBARCOLOR

Sets the color of the progress indicator bar in the progress bar control.

```
PBM_SETBARCOLOR
    wParam = 0;
    lParam = (LPARAM)(COLORREF)clrBar;
```

Parameters

clrBar

COLORREF value that specifies the new progress indicator bar color. Specify the CLR_DEFAULT value to cause the progress bar to use its default progress indicator bar color.

Return Values

Returns the previous progress indicator bar color, or CLR_DEFAULT if the progress indicator bar color is the default color.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

PBM_SETBKCOLOR

Sets the background color in the progress bar.

```
PBM_SETBKCOLOR
    wParam = 0;
    lParam = (LPARAM)(COLORREF)clrBk;
```

Parameters

clrBk

COLORREF value that specifies the new background color. Specify the CLR_DEFAULT value to cause the progress bar to use its default background color.

Return Values

Returns the previous background color, or CLR_DEFAULT if the background color is the default color.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

PBM_SETPOS

Sets the current position for a progress bar and redraws the bar to reflect the new position.

```
PBM_SETPOS
    wParam = (WPARAM) nNewPos;
    lParam = 0;
```

Parameters

nNewPos

Signed integer that becomes the new position.

Return Values

Returns the previous position.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

PBM_SETRANGE

Sets the minimum and maximum values for a progress bar and redraws the bar to reflect the new range.

```
PBM_SETRANGE
    wParam = 0;
    lParam = MAKELPARAM(nMinRange, nMaxRange);
```

Parameters

nMinRange
Minimum range value. By default, the minimum value is zero.

nMaxRange
Maximum range value. By default, the maximum value is 100.

Return Values

Returns the previous range values if successful, or zero otherwise. The low-order word specifies the previous minimum value, and the high-order word specifies the previous maximum value.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

PBM_SETRANGE32

Sets the range of a progress bar control to a 32-bit value.

```
PBM_SETRANGE32
    wParam = (WPARAM)(int) iLowLim;
    lParam = (LPARAM)(int) iHighLim;
```

Parameters

iLowLim
A signed integer that represents the low limit to be set for the progress bar control.

iHighLim
A signed integer that represents the high limit to be set for the progress bar control.

Return Values

Returns a DWORD value that holds the previous 16-bit low limit in its low word and the previous 16-bit high limit in its high word. If the previous ranges were 32-bit values, the return value consists of the low words of both 32-bit limits.

Remarks

To retrieve the entire high and low 32-bit values, use the **PBM_GETRANGE** message.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

PBM_SETSTEP

Specifies the step increment for a progress bar. The step increment is the amount by which the progress bar increases its current position whenever it receives a **PBM_STEPIT** message. By default, the step increment is set to 10.

```
PBM_SETSTEP
    wParam = (WPARAM) nStepInc;
    lParam = 0;
```

Parameters

nStepInc
New step increment.

Return Values

Returns the previous step increment.

Version 4.00 and later of Comctl32.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 commctrl.h.

PBM_STEPIT

Advances the current position for a progress bar by the step increment and redraws the bar to reflect the new position. An application sets the step increment by sending the **PBM_SETSTEP** message.

```
PBM_STEPIT
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the previous position.

Remarks

When the position exceeds the maximum range value, this message resets the current position so that the progress indicator starts over again from the beginning.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

Progress Bar Control Structures

PBRANGE

Contains information about the high and low limits of a progress bar control. This structure is used with the **PBM_GETRANGE** message.

```
typedef struct {
    int iLow;
    int iHigh;
} PBRANGE, *PPBRANGE;
```

Members

iLow
Low limit for the progress bar control. This is a signed integer.

iHigh
High limit for the progress bar control. This is a signed integer.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

CHAPTER 21

Property Sheets

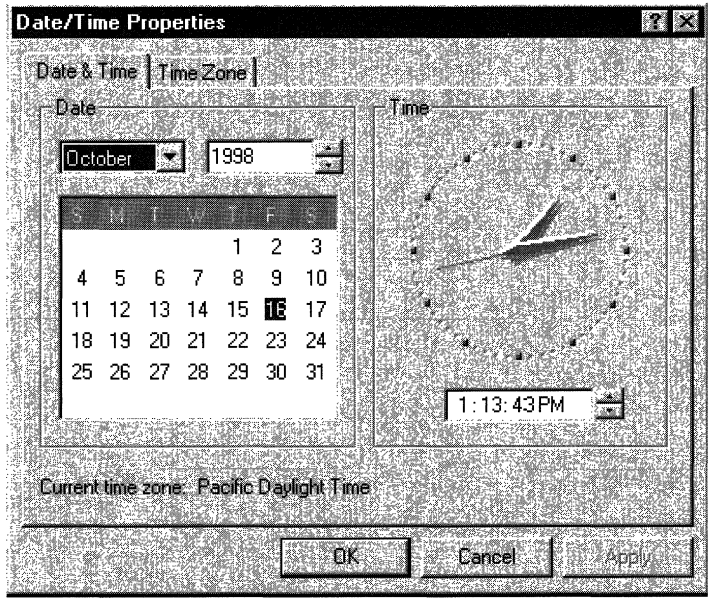
A *property sheet* is a window that allows the user to view and edit the properties of an item. For example, a spreadsheet application can use a property sheet to allow the user to set the font and border properties of a cell or to view and set the properties of a device, such as a disk drive, printer, or mouse.

About Property Sheets

This document assumes that you have a thorough understanding of dialog box templates and dialog box procedures. If you don't, you should read the "Dialog Boxes" chapter in the Platform SDK before continuing with this overview chapter.

To implement property sheets in your application, include the Prsht.h header file in your project. Prsht.h contains all of the identifiers used with property sheets.

A property sheet contains one or more overlapping child windows called *pages*, each containing control windows for setting a group of related properties. For example, a page can contain the controls for setting the font properties of an item, including the type style, point size, color, and so on. Each page has a tab that the user can select to bring the page to the foreground of the property sheet. For example, the Date/Time Control Panel application displays the following property sheet:



There is also a special type of property sheet called a wizard. Wizards are designed to present pages one at a time in a sequence that is controlled by the application. Instead of selecting from a group of pages by clicking a tab, users navigate forward and backward through the sequence, one page at a time, by clicking Next or Back buttons located at the bottom of the wizard. For example, this is the welcome page from the Hardware control panel application's wizard:

See *Creating Wizards* for a complete discussion of wizards.

Property Sheet Dialog Boxes

A property sheet and the pages it contains are actually dialog boxes. The property sheet is a system-defined dialog box that manages the pages and provides a common container for them. The property sheet dialog box can be modal or modeless. It includes a frame, a title bar, and four buttons: OK, Cancel, Apply Now, and Help. (The Help button may be hidden as in the preceding illustration.) The dialog box procedures for the pages receive notification messages when the user clicks the buttons.

Each page in a property sheet is an application-defined modeless dialog box that manages the control windows used to view and edit the properties of an item. You provide the dialog box template used to create each page as well as the dialog box procedure that manages the controls and sets the properties of the corresponding item.

A property sheet sends notification messages to the dialog box procedure for a page when the page is gaining or losing the activation and when the user clicks the OK, Cancel, Apply Now, or Help button. The notifications are sent in the form of **WM_NOTIFY** messages. The *lParam* parameter is the address of an **NMHDR** structure that includes the window handle to the property sheet dialog box.

Some notification messages require a page to return either TRUE or FALSE in response to the **WM_NOTIFY** message. To do this, the page must use the **SetWindowLong** function to set the DWL_MSGRESULT value for the page dialog box to either TRUE or FALSE.

Pages

A property sheet must contain at least one page, but it cannot contain more than the value of MAXPROPPAGES as defined in the Win32 header files. Each page has a zero-based index that the property sheet assigns according to the order in which the page is added to the property sheet. The indexes are used in messages that you send to the property sheet.

A property page can contain a nested dialog box. If it does, you must include the WS_EX_CONTROLPARENT style for the top-level dialog box and call the **IsDialogMessage** function with the handle to the parent dialog box. This ensures that the user can use mnemonics and the dialog box navigation keys to move the focus to controls in the nested dialog box.

Each page has a corresponding icon and label. The property sheet creates a tab for each page and displays the icon and label in the tab. All property sheet pages are expected to use a nonbold font. To ensure that the font is not bold, specify the DS_3DLOOK style in the dialog box template.

The dialog box procedure for a page must not call the **EndDialog** function. Doing so will destroy the entire property sheet, not just the page.

The minimum size for a property sheet page is 212 dialog units horizontally and 114 dialog units vertically. If a page dialog is smaller than this, the page will be enlarged until it meets the minimum size. The Prsht.h header file contains three sets of recommended sizes for property sheet pages. PROP_SM_CXDLG and PROP_SM_CYDLG define the recommende dimensions for a small property sheet page. PROP_MED_CXDLG and PROP_MED_CYDLG define the recommended dimensions for a medium-sized property sheet page. PROP_LG_CXDLG and PROP_LG_CYDLG define the recommended dimensions for a large property sheet page. Using these recommended sizes will help ensure visual consistency between your application and other Microsoft Windows applications.

Use the following values to set the sizes of the elements in your property sheet pages:

PROP_SM_CXDLG	Width, in dialog units, of a small property sheet page.
PROP_SM_CYDLG	Height, in dialog units, of a small property sheet page.
PROP_MED_CXDLG	Width, in dialog units, of a medium-sized property sheet page.
PROP_MED_CYDLG	Height, in dialog units, of a medium-sized property sheet page.
PROP_LG_CXDLG	Width, in dialog units, of a large property sheet page.
PROP_LG_CYDLG	Height, in dialog units, of a large property sheet page.

Property Sheet Creation

Before creating a property sheet, you must define one or more pages. This involves filling a **PROPSHEETPAGE** structure with information about the page—its icon, label, dialog box template, dialog box procedure, and so on—and then specifying the address of the structure in a call to the **CreatePropertySheetPage** function. The function returns a handle to the HPROPSHEETPAGE type that uniquely identifies the page.

To create a property sheet, you specify the address of a **PROPSHEETHEADER** structure in a call to the **PropertySheet** function. The structure defines the icon and title for the property sheet and also includes the address of an array of HPROPSHEETPAGE handles. When **PropertySheet** creates the property sheet, it includes the pages identified in the array. The pages appear in the property sheet in the same order that they are contained in the array.

Another way to create a property sheet is to specify an array of **PROPSHEETPAGE** structures instead of an array of HPROPSHEETPAGE handles. In this case, **PropertySheet** creates handles for the pages before adding them to the property sheet.

When a page is created, its dialog box procedure receives a WM_INITDIALOG message. The message's *lParam* parameter is a pointer to a copy of the **PROPSHEETPAGE** structure that is defined when the page is created. In particular, when a page is created, the structure's **lParam** member can be used to pass application-defined information to the dialog procedure. With the exception of the **lParam** member, this structure should be treated as read-only. Modifying anything other than **lParam** will have unpredictable consequences.

When the system subsequently passes a copy of the page's PROPSHEETPAGE structure to your application, it uses the same pointer. Any changes to the structure will be passed along. Because the **lParam** member is ignored by the system, it can safely be modified to send information to other parts of your application. You can, for instance, use **lParam** to pass information to the page's callback function.

PropertySheet automatically sets the size and initial position of a property sheet. The position is based on the position of the owner window, and the size is based on the largest page specified in the array of pages when the property sheet was created. If you want the pages to match the width of the four buttons at the bottom of the property sheet, set the width of the widest page to 190 dialog units.

Adding and Removing Pages

After creating a property sheet, an application can add a page by using the **PSM_ADDPAGE** message. Note that the size of the property sheet cannot change after it has been created, so the new page must be no larger than the largest page currently in the property sheet.

An application removes a page by using the **PSM_REMOVEPAGE** message. When you define a page, you can specify the address of a **PropSheetPageProc** callback function that the property sheet calls when it is creating or removing the page. Using

PropSheetPageProc gives you an opportunity to perform initialization and cleanup operations for individual pages.

When a property sheet is destroyed, it automatically destroys all of the pages that have been added to it. The pages are destroyed in reverse order from that specified in the array used to create the pages. To destroy a page that was created by the **CreatePropertySheetPage** function but was not added to the property sheet, use the **DestroyPropertySheetPage** function.

Property Sheet Title and Page Labels

You specify the title of a property sheet in the **PROPSHEETHEADER** structure used to create the property sheet. If the **dwFlags** member includes the PSH_PROPTITLE value, the property sheet adds the "Properties for" prefix to the specified title string. You can change the title after a property sheet is created by using the **PSM_SETTITLE** message.

By default, a property sheet uses the name string specified in the dialog box template as the label for a page. You can override the name string by including the PSP_USETITLE value in the **dwFlags** member of the **PROPSHEETPAGE** structure that defines the page. When PSP_USETITLE is specified, the **pszTitle** member must contain the address of the label string for the page.

Page Activation

A property sheet can have only one active page at a time. The page that has the activation is at the foreground of the overlapping stack of pages. The user activates a page by selecting its tab; an application activates a page by using the **PSM_SETCURSEL** message.

The property sheet sends the **PSN_KILLACTIVE** notification message to the page that is about to lose the activation. In response, the page should validate any changes that the user has made to the page. If the page requires additional user input before losing the activation, it should use the **SetWindowLong** function to set the DWL_MSGRESULT value of the page to TRUE. Also, the page should display a message box that describes the problem and provides the recommended action. The page should set DWL_MSGRESULT to FALSE when it is okay to lose the activation.

Before the page that is gaining the activation is visible, the property sheet sends the **PSN_SETACTIVE** notification message to the page. The page should respond by initializing its control windows.

Help Button

Property sheets can display two help buttons, a property sheet help button that is displayed at the bottom the frame, next to the OK/Cancel/Apply buttons, and a standard caption bar button that provides context-sensitive help.

The property sheet help button is optional, and can be enabled on a page by page basis. To display the property sheet help button for one or more pages:

- Set the PSH_HASHELP flag in the **dwFlags** member of the property sheet's **PROPSHEETHEADER** structure.
- For each page that will display a help button, set the PSP_HASHELP flag in the **dwFlags** member of the page's **PROPSHEETPAGE** structure.

When the user clicks the Help button, the active page receives a **PSN_HELP** notification message. The page should respond by displaying Help information, typically by calling the **WinHelp** function.

Removing the Caption Bar Help Button

The caption bar help button is displayed by default, so that context-sensitive help is always available for the OK/Cancel/Apply buttons. However, this button can be removed, if necessary. To remove a property sheet's caption bar help button:

- For versions of the common controls prior to version 5.80, you must implement a property sheet callback function.
- For version 5.80 and later of the common controls you can simply set the PSH_NOCONTEXTHELP flag in the **dwFlags** member of the property sheet's **PROPSHEETHEADER** structure. However, if you need backward compatibility with earlier common control versions, you must implement the callback function.

To implement a property sheet callback function that removes the caption bar help button:

- Set the PSH_USECALLBACK flag in the **dwFlags** member of the property sheet's **PROPSHEETHEADER** structure.
- Set the **pfnCallBack** member of the **PROPSHEETHEADER** structure to point to the callback function.
- Implement the callback function. When this function receives the PSCB_PRECREATE message, it will also receive a pointer to the property sheet's dialog box template. Remove the DS_CONTEXTHELP style from this template.

The following sample illustrates how to implement such a callback function:

```
int CALLBACK RemoveContextHelpProc(HWND hwnd, UINT message, LPARAM lParam)
{
    switch (message) {
    case PSCB_PRECREATE:
        // Remove the DS_CONTEXTHELP style from the dialog template
        if (((LPDLGTEMPLATEEX)lParam)->signature == 0xFFFF){
            ((LPDLGTEMPLATEEX)lParam)->style &= ~DS_CONTEXTHELP;
        }
        else {
```

```
            ((LPDLGTEMPLATE)lParam)->style &= ~DS_CONTEXTHELP;
        }
        return TRUE;
    }
    return TRUE;
}
```

If the **DLGTEMPLATEEX** structure is not defined, include the following declaration:

```
#include <pshpack1.h>
typedef struct DLGTEMPLATEEX
{
    WORD dlgVer;
    WORD signature;
    DWORD helpID;
    DWORD exStyle;
    DWORD style;
    WORD cDlgItems;
    short x;
    short y;
    short cx;
    short cy;
} DLGTEMPLATEEX, *LPDLGTEMPLATEEX;
#include <poppack.h>
```

OK, Cancel, and Apply Now Buttons

The OK and Apply Now buttons are similar; both direct a property sheet's pages to validate and apply the property changes that the user has made. The only difference is that clicking the OK button causes the property sheet to be destroyed after the changes are applied.

When the user clicks the OK or Apply Now button, the property sheet sends the **PSN_KILLACTIVE** notification message to the active page, giving it an opportunity to validate the user's changes. If the page determines that the changes are valid, it should call the **SetWindowLong** function to set the DWL_MSGRESULT value for the page to FALSE. In this case, the property sheet sends the **PSN_APPLY** notification message to each page, directing them to apply the new properties to the corresponding item. If the page determines that the user's changes are not valid, it should set DWL_MSGRESULT to TRUE and display a dialog box informing the user of the problem. The page remains active until it sets DWL_MSGRESULT to FALSE in response to a **PSN_KILLACTIVE** message. An application can use the **PSM_APPLY** message to simulate the selection of the Apply Now button.

The Apply Now button is initially disabled when a page becomes active, indicating that there are not yet any property changes to apply. When the page receives input through one of its controls indicating that the user has edited a property, the page should send the **PSM_CHANGED** message to the property sheet. The message causes the property

sheet to enable the Apply Now button. If the user subsequently clicks the Apply Now or Cancel button, the page should reinitialize its controls and then send the **PSM_UNCHANGED** message to again disable the Apply Now button.

Sometimes the Apply Now button causes a page to make a change to a property sheet, and the change cannot be undone. When this happens, the page should send the **PSM_CANCELTOCLOSE** message to the property sheet. The message causes the property sheet to change the text of the OK button to "Close," indicating that the applied changes cannot be canceled.

Sometimes a page makes a change to the system configuration that requires Windows to be restarted or the system rebooted before the change can take effect. After making such a change, a page should send either the **PSM_RESTARTWINDOWS** or **PSM_REBOOTSYSTEM** message to the property sheet. These messages cause the **PropertySheet** function to return the ID_PSRESTARTWINDOWS or ID_PSREBOOTSYSTEM value after the property sheet is destroyed.

The property sheet sends the **PSN_RESET** notification message to all pages when the user clicks the Cancel button, indicating that the property sheet is about to be destroyed. A page should use the notification to perform cleanup operations.

Property Sheet Extensions

A *property sheet extension* is a dynamic-link library (DLL) that adds one or more pages to a property sheet created by another module. The module that creates the property sheet includes an **AddPropSheetPageProc** callback function that the extension DLL calls to add a page. The function receives the handle to a page and an application-defined 32-bit value.

The extension DLL also contains a callback function called **ExtensionPropSheetPageProc**, which receives the address of **AddPropSheetPageProc** from the module that creates the property sheet. The extension DLL must export **ExtensionPropSheetPageProc** by name.

The Windows header files include two prototypes for defining property sheet callback functions. To define **AddPropSheetPageProc**, use the following prototype:

```
typedef BOOL (CALLBACK FAR * LPFNADDPROPSHEETPAGE)(HPROPSHEETPAGE, LPARAM);
```

To define **ExtensionPropSheetPageProc**, use the following prototype:

```
typedef BOOL (CALLBACK FAR * LPFNADDPROPSHEETPAGES)(LPVOID, LPFNADDPROPSHEETPAGE,
LPARAM);
```

Using Property Sheets

This section contains examples that demonstrate how to create a property sheet and process notification messages.

Creating a Property Sheet

The example in this section creates a property sheet that contains two pages—one for setting the font properties of a cell in a spreadsheet and another for setting the border properties of the cell. The example defines the pages by filling a pair of **PROPSHEETPAGE** structures and specifying the address in the **PROPSHEETHEADER** structure that is passed to the **PropertySheet** function. The dialog box templates, icons, and labels for the pages are loaded from the resources contained in the application's executable file. The icon for the property sheet is also loaded from the application's resources.

```
// DoPropertySheet - creates a property sheet that contains two pages.
// hwndOwner - handle to the owner window of the property sheet.
//
// Global variables
//     g_hinst - instance handle
extern HINSTANCE g_hinst;

VOID DoPropertySheet(HWND hwndOwner)
{
PROPSHEETPAGE psp[2];
PROPSHEETHEADER psh;

psp[0].dwSize = sizeof(PROPSHEETPAGE);
psp[0].dwFlags = PSP_USEICONID | PSP_USETITLE;
psp[0].hInstance = g_hinst;
psp[0].pszTemplate = MAKEINTRESOURCE(DLG_FONT);
psp[0].pszIcon = MAKEINTRESOURCE(IDI_FONT);
psp[0].pfnDlgProc = FontDialogProc;
psp[0].pszTitle = MAKEINTRESOURCE(IDS_FONT)
psp[0].lParam = 0;
psp[0].pfnCallback = NULL;

psp[1].dwSize = sizeof(PROPSHEETPAGE);
psp[1].dwFlags = PSP_USEICONID | PSP_USETITLE;
psp[1].hInstance = g_hinst;
psp[1].pszTemplate = MAKEINTRESOURCE(DLG_BORDER);
psp[1].pszIcon = MAKEINTRESOURCE(IDI_BORDER);
psp[1].pfnDlgProc = BorderDialogProc;
psp[1].pszTitle = MAKEINTRESOURCE(IDS_BORDER);
psp[1].lParam = 0;
psp[1].pfnCallback = NULL;

psh.dwSize = sizeof(PROPSHEETHEADER);
psh.dwFlags = PSH_USEICONID | PSH_PROPSHEETPAGE;
```

(continued)

(continued)

```
    psh.hwndParent = hwndOwner;
    psh.hInstance = g_hinst;
    psh.pszIcon = MAKEINTRESOURCE(IDI_CELL_PROPERTIES);
    psh.pszCaption = (LPSTR) "Cell Properties";
    psh.nPages = sizeof(psp) / sizeof(PROPSHEETPAGE);
    psh.nStartPage = 0;
    psh.ppsp = (LPCPROPSHEETPAGE) &psp;
    psh.pfnCallback = NULL;

    PropertySheet(&psh);
    return;
}
```

Processing Notification Messages

A property sheet sends **WM_NOTIFY** messages to retrieve information from the pages and to notify the pages of user actions. The *lParam* parameter of the message is the address of an **NMHDR** structure, which contains the handle to the property sheet dialog box, the handle to the page dialog box, and a notification code. The page must respond to some notification messages by setting the DWL_MSGRESULT value of the page to either TRUE or FALSE.

The following example is a code fragment from the dialog box procedure for a page. It shows how to process the **PSN_HELP** notification message.

```
    case WM_NOTIFY:
        switch (((NMHDR FAR *) lParam)->code) {

        case PSN_HELP:
            {
            char szBuf[FILE_LEN]; // buffer for name of help file

            // Display help for the font properties page.
            LoadString(g_hinst, IDS_HELPFILE, &szBuf, FILE_LEN)
            WinHelp(((NMHDR FAR *) lParam)->hwndFrom, &szBuf,
                HELP_CONTEXT, IDH_FONT_PROPERTIES);
            break;
            }
        .
        . // Process other property sheet notifications here.
        .
    }
```

Property Sheet Updates in Internet Explorer

Property sheets in Microsoft Internet Explorer support the following new features:

New Notification
The **PSN_GETOBJECT** notification has been added to allow a page to be an OLE drop target.

Updated Structures
The **PROPSHEETHEADER** and **PROPSHEETPAGE** structures have been updated to support new features. See the references for these structures for more information.

Property Sheet Reference

Property Sheet Functions

AddPropSheetPageProc

Specifies an application-defined callback function that a property sheet extension uses to add a page to a property sheet.

```
BOOL CALLBACK AddPropSheetPageProc(
  HPROPSHEETPAGE hpage,
  LPARAM lParam
);
```

Parameters

hpage
Handle to a property sheet page.

lParam
Application-defined 32-bit value.

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: Requires version 1.0 or later.
Header: Declared in prsht.h.
Import Library: user-defined.

CreatePropertySheetPage

Creates a new page for a property sheet.

```
HPROPSHEETPAGE CreatePropertySheetPage(
  LPCPROPSHEETPAGE lppsp
);
```

Parameters

lppsp
Address of a **PROPSHEETPAGE** structure that defines a page to be included in a property sheet.

Return Values

Returns the handle to the new property page if successful, or NULL otherwise.

Remarks

An application uses the **PropertySheet** function to create a property sheet that includes the new page, or it uses the **PSM_ADDPAGE** message to add the new page to an existing property sheet.

Windows 95: The system can support a maximum of 16,364 window handles.

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 prsht.h.
Import Library: comctl32.lib.

DestroyPropertySheetPage

Destroys a property sheet page. An application must call this function for pages that have not been passed to the **PropertySheet** function.

```
BOOL DestroyPropertySheetPage(
  HPROPSHEETPAGE hPSPage
);
```

Parameters

hPSPage
Handle to the property sheet page to delete.

Return Values

Returns nonzero if successful, or zero otherwise.

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 prsht.h.
Import Library: comctl32.lib.

ExtensionPropSheetPageProc

Specifies an application-defined callback function that receives the address of the **AddPropSheetPageProc** function, which resides in the module that creates a property sheet. A property sheet extension must export the **ExtensionPropSheetPageProc** function by name.

```
BOOL CALLBACK ExtensionPropSheetPageProc(
  LPVOID lpv,
  LPFNADDPROPSHEETPAGE lpfnAddPropSheetPageProc,
  LPARAM lParam
);
```

Parameters

lpv
Address of an application-defined value that describes an item for which a property sheet page is to be created. This parameter can be NULL.

lpfnAddPropSheetPageProc
Address of the **AddPropSheetPageProc** function. The extension dynamic-link library (DLL) calls this function to add a page to the property sheet.

lParam
Application-defined 32-bit value.

Return Values

Returns nonzero if successful, or zero otherwise.

Remarks

This function is supported for backward compatibility reasons. Property sheet extension handlers should instead use **AddPropSheetPageProc**.

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 prsht.h.
Import Library: user-defined.

PropertySheet

Creates a property sheet and adds the pages defined in the specified property sheet header structure.

```
int PropertySheet(
  LPCPROPSHEETHEADER lppsph
);
```

Parameters

lppsph
Pointer to a **PROPSHEETHEADER** structure that defines the frame and pages of a property sheet.

Return Values

Returns a positive value if successful, or –1 otherwise for modal property sheets.

Returns the property sheet's window handle for modeless property sheets.

The following return values have a special meaning:

ID_PSREBOOTSYSTEM	A page sent the **PSM_REBOOTSYSTEM** message to the property sheet. The computer must be restarted for the user's changes to take effect.
ID_PSRESTARTWINDOWS	A page sent the **PSM_RESTARTWINDOWS** message to the property sheet. Windows must be restarted for the user's changes to take effect.

To get extended error information, call **GetLastError**.

Remarks

By default, the **PropertySheet** function creates a modal dialog box. If the **dwFlags** member of the **PROPSHEETHEADER** structure specifies the PSH_MODELESS flag, **PropertySheet** creates a modeless dialog box and returns immediately after it is created. In this case, the **PropertySheet** return value is the window handle to the modeless dialog box.

For a modeless property sheet, your message loop should use **PSM_ISDIALOGMESSAGE** to pass messages to the property sheet dialog box. Your message loop should use **PSM_GETCURRENTPAGEHWND** to determine when to destroy the dialog box. When the user clicks the OK or Cancel button, **PSM_GETCURRENTPAGEHWND** returns NULL. You can then use the **DestroyWindow** function to destroy the dialog box.

Version 5.80. The **PropertySheet** return value carries different information for modal and modeless property sheets. In some cases, modeless property sheets may need the information they would have received from **PropertySheet** if they had been modal. In particular, they may need to know whether ID_PSREBOOTSYSTEM or ID_PSRESTARTWINDOWS would have been returned. A modeless property sheet can retrieve the value that a modal property sheet would have received from **PropertySheet** by waiting until **PSM_GETCURRENTPAGEHWND** returns NULL, and then sending a **PSM_GETRESULT** message.

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 prsht.h.
Import Library: comctl32.lib.

PropSheetPageProc

Specifies an application-defined callback function that a property sheet calls when a page is created and when it is about to be destroyed. An application can use this function to perform initialization and cleanup operations for the page.

```
UINT CALLBACK PropSheetPageProc(
  HWND hwnd,
  UINT uMsg,
  LPPROPSHEETPAGE ppsp
);
```

Parameters

hwnd
Reserved; must be NULL.

uMsg
[in] Action flag. This parameter can be one of the following values:

PSPCB_ADDREF	Version 5.80. A page is being created. The return value is not used.
PSPCB_CREATE	A dialog box for a page is being created. Return nonzero to allow it to be created, or zero to prevent it.
PSPCB_RELEASE	A page is being destroyed. The return value is ignored.

ppsp
[in/out] Address of a **PROPSHEETPAGE** structure that defines the page being created or destroyed. See the Remarks section for further discussion.

Return Values

The return value depends on the value of the *uMsg* parameter.

Remarks

An application must specify the address of this callback function in the **pfnCallback** member of a **PROPSHEETPAGE** structure before specifying the address of the structure in a call to the **CreatePropertySheetPage** function.

With the exception of the **lParam** member, your application should not modify the **PROPSHEETPAGE** structure. Doing so will have unpredictable results. The **lParam** member contains application-defined data and can be modified as needed.

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 prsht.h.
Import Library: user-defined.

PropSheetProc

An application-defined callback function that the system calls when the property sheet is being created and initialized.

```
int CALLBACK PropSheetProc(
    HWND hwndDlg,
    UINT uMsg,
    LPARAM lParam
);
```

Parameters

hwndDlg
Handle to the property sheet dialog box.

uMsg
Message being received. This parameter is one of the following values:

PSCB_INITIALIZED	Indicates that the property sheet is being initialized. The *lParam* value is zero for this message.
PSCB_PRECREATE	Indicates that the property sheet is about to be created. The *hwndDlg* parameter is NULL, and the *lParam* parameter is the address of a dialog template in memory. This template is in the form of a **DLGTEMPLATE** structure followed by one or more **DLGITEMTEMPLATE** structures.

lParam
Additional information about the message. The meaning of this value depends on the *uMsg* parameter.

Return Values
Returns zero.

Remarks
To enable a **PropSheetProc** callback function, use the **PROPSHEETHEADER** structure when you call the **PropertySheet** function to create the property sheet. Use the **pfnCallback** member to specify an address of the callback function, and set the PSP_USECALLBACK flag in the **dwFlags** member.

PropSheetProc is a placeholder for the application-defined function name. The PFNPROPSHEETCALLBACK type is the address of a **PropSheetProc** callback function.

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 prsht.h.
Import Library: user-defined.

Property Sheet Messages

PSM_ADDPAGE

Adds a new page to the end of an existing property sheet. You can send this message explicitly or by using the **PropSheet_AddPage** macro.

```
PSM_ADDPAGE
    wParam = 0;
    lParam = (LPARAM) (HPROPSHEETPAGE) hpage;
```

Parameters
hpage
Handle to the page to add. The page must have been created by a previous call to the **CreatePropertySheetPage** function.

Return Values
Returns TRUE if successful, FALSE otherwise.

Remarks

The new page should be no larger than the largest page currently in the property sheet because the property sheet is not resized to fit the new page.

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 prsht.h.

PSM_APPLY

Simulates the selection of the Apply button, indicating that one or more pages have changed and the changes need to be validated and recorded.

```
PSM_APPLY
    wParam = 0;
    lParam = 0;
```

Return Values

Returns TRUE if all pages successfully applied the changes, or FALSE otherwise.

Remarks

The property sheet sends the **PSN_KILLACTIVE** notification message to the current page. If the current page returns FALSE, the property sheet sends the **PSN_APPLY** notification message to all pages. You can send the PSM_APPLY message explicitly or by using the **PropSheet_Apply** macro.

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 prsht.h.

PSM_CANCELTOCLOSE

Sent by an application when it has performed changes since the most recent **PSN_APPLY** notification that cannot be canceled. You can send this message explicitly or by using the **PropSheet_CancelToClose** macro.

```
PSM_CANCELTOCLOSE
    wParam = 0;
    lParam = 0;
```

Return Values

No return value.

Remarks

PSM_CANCELTOCLOSE disables the Cancel button and changes the text of the OK button to "Close."

Most property sheets wait to perform irreversible changes until a PSN_APPLY notification is received. However, in some circumstances, a property sheet may make irreversible changes outside the standard PSN_APPLY/PSN_RESET sequence. One example is a property sheet that contains an Edit button that is used to display a subdialog box for editing a property. When the user clicks OK to submit the change, the property sheet page has several options:

- It can record the changes, but wait until it receives a PSN_APPLY notification to apply them. This is the preferred approach.
- It can apply the changes immediately after exiting the subdialog box, but remember the original settings. Those settings can be used to restore the original state if a PSN_RESET notification is received.
- It can apply the changes immediately and not attempt to restore the original settings when it receives a PSN_RESET notification. This approach is not recommended, but may be necessary if the changes are too far-reaching for the other two options to be practical.

For the third option, applications should send a PSM_CANCELTOCLOSE message to the property sheet. It indicates to the user that the changes made with the subdialog box cannot be reversed by clicking the Cancel button.

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 prsht.h.

PSM_CHANGED

Informs a property sheet that information in a page has changed. You can send this message explicitly or by using the **PropSheet_Changed** macro.

```
PSM_CHANGED
    wParam = (WPARAM) (HWND) hwndPage;
    lParam = 0;
```

Parameters

hwndPage
Handle to the page that has changed.

Return Values

No return value.

Remarks

The property sheet will enable the Apply button.

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 prsht.h.

PSM_GETCURRENTPAGEHWND

Retrieves a handle to the window of the current page of a property sheet. You can send this message explicitly or by using the **PropSheet_GetCurrentPageHwnd** macro.

```
PSM_GETCURRENTPAGEHWND
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a handle to the window of the current property sheet page.

Remarks

Use the PSM_GETCURRENTPAGEHWND message with modeless property sheets to determine when to destroy the dialog box. When the user clicks the OK or Cancel button, PSM_GETCURRENTPAGEHWND returns NULL, and you can then use the **DestroyWindow** function to destroy the dialog box.

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 prsht.h.

See Also

PropertySheet

PSM_GETTABCONTROL

Retrieves the handle to the tab control of a property sheet. You can send this message explicitly or by using the **PropSheet_GetTabControl** macro.

```
PSM_GETTABCONTROL
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the handle to the tab control.

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 prsht.h.

PSM_HWNDTOINDEX

Takes the window handle of the property sheet page and returns its zero-based index. You can send this message explicitly or use the **PropSheet_HwndToIndex** macro.

```
PSM_HWNDTOINDEX
    wParam = (WPARAM) (HWND) hPageDlg;
```

Parameters

hPageDlg
Handle to the page's window.

Return Values

Returns the zero-based index of the property sheet page specified by *hPageDlg* if successful. Otherwise, it returns –1.

Requirements

Version 5.80 and later of Comctl32.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 prsht.h.

PSM_IDTOINDEX

Takes the resource identifier (ID) of a property sheet page and returns its zero-based index. You can send this message explicitly or use the **PropSheet_IdToIndex** macro.

```
PSM_IDTOINDEX
    lParam = (LPARAM) (int) iPageID;
```

Parameters

iPageID
Resource ID of the page.

Return Values

Returns the zero-based index of the property sheet page specified by *iPageID* if successful. Otherwise, it returns –1.

Requirements

Version 5.80 and later of Comctl32.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 prsht.h.

PSM_INDEXTOHWND

Takes the index of a property sheet page and returns its window handle. You can send this message explicitly or use the **PropSheet_IndexToHwnd** macro.

```
PSM_INDEXTOHWND
    wParam = (WPARAM) (int) iPageIndex;
```

Parameters

iPageIndex
Zero-based index of the page.

Return Values

Returns the handle to the window of the property sheet page specified by *iPageIndex* if successful. Otherwise, it returns zero.

Requirements

Version 5.80 and later of Comctl32.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 prsht.h.

PSM_INDEXTOID

Takes the index of a property sheet page and returns its resource identifier (ID). You can send this message explicitly or use the **PropSheet_IndexToId** macro.

```
PSM_INDEXTOID
    wParam = (WPARAM) (int) iPageIndex;
```

Parameters

iPageIndex
Zero-based index of the page.

Return Values

Returns the resource ID of the property sheet page specified by *iPageIndex* if successful. Otherwise, it returns zero.

Requirements

Version 5.80 and later of Comctl32.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 prsht.h.

PSM_INDEXTOPAGE

Takes the index of a property sheet page and returns its HPROPSHEETPAGE handle. You can send this message explicitly or use the **PropSheet_IndexToPage** macro.

```
PSM_INDEXTOPAGE
   wParam = (WPARAM) (int) iPageIndex;
```

Parameters

iPageIndex
Zero-based index of the page.

Return Values

Returns the HPROPSHEETPAGE handle of the property sheet page specified by *iPageIndex* if successful. Otherwise, it returns zero.

Requirements

Version 5.80 and later of Comctl32.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 prsht.h.

PSM_INSERTPAGE

Inserts a new page into an existing property sheet. It can be inserted either at a specified index or after a specified page. You can send this message explicitly or by using the **PropSheet_InsertPage** macro.

```
PSM_INSERTPAGE
   wParam = (WPARAM) index;
   - or -
   wParam = (WPARAM) (HPROPSHEETPAGE) hpageInsertAfter

   lParam = (LPARAM) (HPROPSHEETPAGE) hpage;
```

Parameters

wParam
Location where the page is to be inserted. Set this parameter to NULL to make the new page the first page. To specify where the new page is to be inserted, you can either pass an index or an existing page's HPROPSHEETPAGE handle.

index
If the *wParam* parameter is less than MAXUSHORT, it specifies the zero-based index for the new page. For example, to make the inserted page the third one on the property sheet, set *index* to 2. To make it the first page, set *index* to 0. If *index* has a value greater than the number of pages and less than MAXUSHORT, the page will be appended.

hpageInsertAfter
If you set the *wParam* parameter to an existing page's HPROPSHEETPAGE handle, the new page will be inserted after it.

hpage
Handle to the page to be inserted. It must first be created by a call to the **CreatePropertySheetPage** function.

Return Values

Returns a nonzero value if the page was successfully inserted. Otherwise, it returns zero.

Remarks

The pages after the insertion point are shifted to the right to accommodate the new page.

The property sheet is not resized to fit the new page. Do not make the new page larger than the property sheet's largest page.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.

PSM_ISDIALOGMESSAGE

Passes a message to a property sheet dialog box and indicates whether the dialog box processed the message. You can send this message explicitly or by using the **PropSheet_IsDialogMessage** macro.

```
PSM_ISDIALOGMESSAGE
   wParam = 0;
   lParam = (LPARAM)pMsg;
```

Parameters

pMsg
Address of an **MSG** structure that contains the message to be checked.

Return Values

Returns TRUE if the message has been processed, or FALSE if the message has not been processed.

Remarks

Your message loop should use the PSM_ISDIALOGMESSAGE message with modeless property sheets to pass messages to the property sheet dialog box.

If the return value indicates that the message was processed, it must not be passed to the **TranslateMessage** or **DispatchMessage** function.

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 prsht.h.

See Also

PropertySheet

PSM_PAGETOINDEX

Takes the HPROPSHEETPAGE handle of the property sheet page and returns its zero-based index. You can send this message explicitly or use the **PropSheet_PageToIndex** macro.

```
PSM_PAGETOINDEX
    lParam = (LPARAM) (HPROPSHEETPAGE) hPage;
```

Parameters

hPage
HPROPSHEETPAGE handle to the property sheet page.

Return Values

Returns the zero-based index of the property sheet page specified by *hPage* if successful. Otherwise, it returns –1.

Requirements

Version 5.80 and later of Comctl32.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 prsht.h.

PSM_PRESSBUTTON

Simulates the selection of a property sheet button. You can send this message explicitly or by using the **PropSheet_PressButton** macro.

```
PSM_PRESSBUTTON
    wParam = (WPARAM) (int) iButton;
    lParam = 0;
```

Parameters

iButton

Index of the button to select. This parameter can be one of the following values:

PSBTN_APPLYNOW	Selects the Apply button.
PSBTN_BACK	Selects the Back button.
PSBTN_CANCEL	Selects the Cancel button.
PSBTN_FINISH	Selects the Finish button.
PSBTN_HELP	Selects the Help button.
PSBTN_NEXT	Selects the Next button.
PSBTN_OK	Selects the OK button.

Return Values

No return value.

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 prsht.h.

PSM_QUERYSIBLINGS

Sent to a property sheet, which then forwards the message to each of its pages. You can send this message explicitly or by using the **PropSheet_QuerySiblings** macro.

```
PSM_QUERYSIBLINGS
    wParam = (WPARAM) param1;
    lParam = (LPARAM) param2;
```

Parameters

param1
First application-defined parameter.

param2
Second application-defined parameter.

Return Values

Returns the nonzero value from a page in the property sheet, or zero if no page returns a nonzero value.

Remarks

If a page returns a nonzero value, the property sheet does not send the message to subsequent pages.

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 prsht.h.

PSM_REBOOTSYSTEM

Indicates the system needs to be restarted for the changes to take effect. You can send the PSM_REBOOTSYSTEM message explicitly or by using the **PropSheet_RebootSystem** macro.

```
PSM_REBOOTSYSTEM
    wParam = 0;
    lParam = 0;
```

Return Values

No return value.

Remarks

An application should send this message only in response to the **PSN_APPLY** or **PSN_KILLACTIVE** notification message.

This message causes the **PropertySheet** function to return the ID_PSREBOOTSYSTEM value, but only if the user clicks the OK button to close the property sheet. It is the application's responsibility to reboot the system, which can be done by using the **ExitWindowsEx** function.

This message supersedes all **PSM_RESTARTWINDOWS** messages that precede or follow it.

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 prsht.h.

PSM_REMOVEPAGE

Removes a page from a property sheet. You can send this message explicitly or by using the **PropSheet_RemovePage** macro.

```
PSM_REMOVEPAGE
    wParam = (WPARAM) (int) index;
    lParam = (LPARAM) (HPROPSHEETPAGE) hpage);
```

Parameters

index and *hpage*
Zero-based index of the page and the handle to the page to remove, respectively. An application can specify the index or the handle, or both. If both are specified, *hpage* takes precedence.

Return Values

No return value.

Remarks

Sending PSM_REMOVEPAGE will destroy the property sheet page that is being removed.

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 prsht.h.

PSM_RESTARTWINDOWS

Indicates that Windows needs to be restarted for the changes to take effect.

```
PSM_RESTARTWINDOWS
    wParam = 0;
    lParam = 0;
```

Return Values

No return value.

Remarks

An application should send this message only in response to the **PSN_APPLY** or **PSN_KILLACTIVE** notification message. You can send the PSM_RESTARTWINDOWS message explicitly or by using the **PropSheet_RestartWindows** macro.

This message causes the **PropertySheet** function to return the ID_PSRESTARTWINDOWS value, but only if the user clicks the OK button to close the property sheet. It is the application's responsibility to restart Windows, which can be done by using the **ExitWindowsEx** function.

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 prsht.h.

PSM_SETCURSEL

Activates the specified page in a property sheet. You can send this message explicitly or by using the **PropSheet_SetCurSel** macro.

```
PSM_SETCURSEL
    wParam = (WPARAM) (int) index;
    lParam = (LPARAM) (HPROPSHEETPAGE) hpage;
```

Parameters

index and *hpage*

The zero-based index of the page and the handle to the page to activate, respectively. An application can specify the index or the handle, or both. If both are specified, *hpage* takes precedence.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

The window that is losing the activation receives the **PSN_KILLACTIVE** notification message, and the window that is gaining the activation receives the **PSN_SETACTIVE** notification message.

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 prsht.h.

PSM_SETCURSELID

Activates the given page in a property sheet based on the resource identifier of the page. You can send this message explicitly or by using the **PropSheet_SetCurSelByID** macro.

```
PSM_SETCURSELID
    wParam = 0;
    lParam = (LPARAM) (int) id;
```

Parameters

id

Resource identifier of the page to activate.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

The window that is losing the activation receives the **PSN_KILLACTIVE** notification message, and the window that is gaining the activation receives the **PSN_SETACTIVE** notification message.

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 prsht.h.

PSM_SETFINISHTEXT

Sets the text of the Finish button in a wizard, shows and enables the button, and hides the Next and Back buttons. You can send this message explicitly or by using the **PropSheet_SetFinishText** macro.

```
PSM_SETFINISHTEXT
    wParam = 0;
    lParam = (LPARAM) (LPSTR) lpszText;
```

Parameters

lpszText
Address of the new text for the Finish button.

Return Values

No return value.

Remarks

This message causes the DM_SETDEFID message to be sent to the wizard dialog box. The *wParam* parameter specifies the identifier of the Finish button.

Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 2.0 or later.
Header: Declared in prsht.h.

PSM_SETHEADERSUBTITLE

Sets the subtitle text for the header of a wizard's interior page. You can send this message explicitly or use the **PropSheet_SetHeaderSubTitle** macro.

```
PSM_SETHEADERSUBTITLE
    wParam = (WPARAM) (int) iPageIndex;
    lParam = (LPARAM) (LPCTSTR) pszHeaderSubTitle;
```

Parameters

iPageIndex
Zero-based index of the wizard's page.

pszHeaderSubTitle
New header subtitle.

Return Values
No return value.

Remarks
If you specify the current page, it will immediately be repainted to display the new subtitle.

Requirements

Version 5.80 and later of Comctl32.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 prsht.h.

See Also

PSM_HWNDTOINDEX, **PSM_IDTOINDEX**, **PSM_PAGETOINDEX**

PSM_SETHEADERTITLE

Sets the title text for the header of a wizard's interior page. You can send this message explicitly or use the **PropSheet_SetHeaderTitle** macro.

```
PSM_SETHEADERTITLE
    wParam = (WPARAM) (int) iPageIndex;
    lParam = (LPARAM) (LPCTSTR) pszHeaderTitle;
```

Parameters
iPageIndex
Zero-based index of the wizard's page.

pszHeaderTitle
New header subtitle.

Return Values
No return value.

Remarks
If you specify the current page, it will immediately be repainted to display the new title.

Requirements

Version 5.80 and later of Comctl32.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 prsht.h.

See Also

PSM_HWNDTOINDEX, **PSM_IDTOINDEX**, **PSM_PAGETOINDEX**

PSM_SETTITLE

Sets the title of a property sheet. You can send this message explicitly or by using the **PropSheet_SetTitle** macro.

```
PSM_SETTITLE
    wParam = (WPARAM) (DWORD) dwStyle
    lParam = (LPARAM) (LPCSTR) lpszText;
```

Parameters

dwStyle
Flag that indicates whether to include the prefix "Properties for" with the specified title string. If *dwStyle* is the PSH_PROPTITLE value, the prefix is included. Otherwise, the prefix is not used.

lpszText
Address of a buffer that contains the title string. If the high-order word of this parameter is NULL, the property sheet loads the string resource specified in the low-order word.

Return Values

No return value.

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 prsht.h.

PSM_SETWIZBUTTONS

Enables or disables the Back, Next, and Finish buttons in a wizard. You can also use the **PropSheet_SetWizButtons** macro to post the message.

```
PSM_SETWIZBUTTONS
    wParam = 0;
    lParam = (LPARAM) (DWORD) dwFlags;
```

Parameters

dwFlags

Value that specifies which property sheet buttons are enabled. You can combine one or more of the following flags:

PSWIZB_BACK	Enables the Back button. If this flag is not set, the Back button is displayed as disabled.
PSWIZB_DISABLEDFINISH	Displays a disabled Finish button.
PSWIZB_FINISH	Displays an enabled Finish button.
PSWIZB_NEXT	Enables the Next button. If this flag is not set, the Next button is displayed as disabled.

Return Values

No return value.

Remarks

If your notification handler uses **PostMessage** to send a **PSM_SETWIZBUTTONS** message, do not do anything that will affect window focus until after the handler returns. For example, if you call **MessageBox** immediately after using **PostMessage** to send **PSM_SETWIZBUTTONS**, the message box will receive focus. Since posted messages are not delivered until they reach the head of the message queue, the **PSM_SETWIZBUTTONS** message will not be delivered until after the wizard has lost focus to the message box. As a result, the property sheet will not be able to properly set the focus for the buttons.

If you send the PSM_SETWIZBUTTONS message during your handling of the **PSN_SETACTIVE** notification message, use the **PostMessage** function rather than the **SendMessage** function. Otherwise, the system will not update the buttons properly. If you use the **PropSheet_SetWizButtons** macro to send this message, it will be posted. At any other time, you can use **SendMessage** to send PSM_SETWIZBUTTONS.

Wizards display either three or four buttons below each page. This message is used to specify which buttons are enabled. Wizards normally display Back, Cancel, and either a Next or a Finish button. You typically enable only the Next button for the welcome page, Next and Back for interior pages, and Back and Finish for the completion page. The Cancel button is always enabled. Normally, setting PSWIZB_FINISH or PSWIZB_DISABLEDFINISH replaces the Next button with a Finish button. To display

Next and Finish buttons simultaneously, set the PSH_WIZARDHASFINISH FLAG in the **dwFlags** member of the wizard's **PROPSHEETHEADER** structure when you create the wizard. Every page will then display all four buttons.

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 prsht.h.

PSM_UNCHANGED

Informs a property sheet that information in a page has reverted to the previously saved state. You can send this message explicitly or by using the **PropSheet_UnChanged** macro.

```
PSM_UNCHANGED
    wParam = (WPARAM) (HWND) hwndPage;
    lParam = 0;
```

Parameters

hwndPage
Handle to the page that has reverted to the previously saved state.

Return Values

No return value.

Remarks

The property sheet disables the Apply button if no other pages have registered changes with the property sheet.

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 prsht.h.

Property Sheet Macros

PropSheet_AddPage

Adds a new page to the end of an existing property sheet. You can use this macro or send the **PSM_ADDPAGE** message explicitly.

```
BOOL PropSheet_AddPage(
    HWND hPropSheetDlg,
    HPROPSHEETPAGE hpage,
);
```

Parameters

hPropSheetDlg
Handle to the property sheet.

hpage
Handle to the page to add. The page must have been created by a previous call to the **CreatePropertySheetPage** function.

Return Values

Returns TRUE if successful, FALSE otherwise.

Remarks

The new page should be no larger than the largest page currently in the property sheet because the property sheet is not resized to fit the new page.

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 prsht.h.

PropSheet_Apply

Simulates the selection of the Apply button, indicating that one or more pages have changed and the changes need to be validated and recorded. You can use this macro or send the **PSM_APPLY** message explicitly.

```
BOOL PropSheet_Apply(
    HWND hPropSheetDlg
);
```

Parameters

hPropSheetDlg
Handle to the property sheet.

Return Values

Returns TRUE if all pages successfully applied the changes, or FALSE otherwise.

Remarks

The property sheet sends the **PSN_KILLACTIVE** notification message to the current page. If the current page returns FALSE, the property sheet sends the **PSN_APPLY** notification message to all pages.

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 prsht.h.

PropSheet_CancelToClose

Used when changes made since the most recent **PSN_APPLY** notification cannot be canceled. You can also send a **PSM_CANCELTOCLOSE** message explicitly.

```
VOID PropSheet_CancelToClose(
    HWND hPropSheetDlg
);
```

Parameters

hPropSheetDlg
Handle to the property sheet.

Return Values

No return value.

Remarks

PSM_CANCELTOCLOSE disables the Cancel button and changes the text of the OK button to "Close." You can use this macro or send the PSM_CANCELTOCLOSE message explicitly.

Most property sheets wait to perform irreversible changes until a PSN_APPLY notification is received. However, in some circumstances, a property sheet may make irreversible changes outside the standard PSN_APPLY/PSN_RESET sequence. One example is a property sheet that contains an Edit button that is used to display a

subdialog box for editing a property. When the user clicks OK to submit the change, the property sheet page has several options:

- It can record the changes but wait until it receives a PSN_APPLY notification to apply them. This is the preferred approach.
- It can apply the changes immediately after exiting the subdialog box, but remember the original settings. Those settings can be used to restore the original state if a PSN_RESET notification is received.
- It can apply the changes immediately and not attempt to restore the original settings when it receives a PSN_RESET notification. This approach is not recommended, but may be necessary if the changes are too far-reaching for the other two options to be practical.

For the third option, applications should send a PSM_CANCELTOCLOSE message to the property sheet. It indicates to the user that the changes made with the subdialog box cannot be reversed by clicking the Cancel button.

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 prsht.h.

PropSheet_Changed

Informs a property sheet that information in a page has changed. You can use this macro or send the **PSM_CHANGED** message explicitly.

```
BOOL PropSheet_Changed(
    HWND hPropSheetDlg,
    HWND hwndPage
);
```

Parameters

hPropSheetDlg
Handle to the property sheet.

hwndPage
Handle to the page that has changed.

Return Values

No return value.

Remarks

The property sheet enables the Apply button.

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 prsht.h.

PropSheet_GetCurrentPageHwnd

Retrieves a handle to the window of the current page of a property sheet. You can use this macro or send the **PSM_GETCURRENTPAGEHWND** message explicitly.

```
HWND PropSheet_GetCurrentPageHwnd(
    HWND hDlg
);
```

Parameters

hDlg
Handle to the property sheet.

Return Values

Returns a handle to the window of the current property sheet page.

Remarks

Use the **PropSheet_GetCurrentPageHwnd** macro with modeless property sheets to determine when to destroy the dialog box. When the user clicks the OK or Cancel button, **PropSheet_GetCurrentPageHwnd** returns NULL, and you can then use the **DestroyWindow** function to destroy the dialog box.

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 prsht.h.

See Also

PropertySheet

PropSheet_GetTabControl

Retrieves the handle to the tab control of a property sheet. You can use this macro or send the **PSM_GETTABCONTROL** message explicitly.

```
HWND PropSheet_GetTabControl(
    HWND hPropSheetDlg
);
```

Parameters

hPropSheetDlg
Handle to the property sheet.

Return Values

Returns the handle to the tab control.

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 prsht.h.

PropSheet_HwndToIndex

Takes a window handle of the property sheet page and returns its zero-based index. You can use this macro or send the **PSM_HWNDTOINDEX** message explicitly.

```
int PropSheet_HwndToIndex(
  HWND hPropSheetDlg,
  HWND hPageDlg
);
```

Parameters

hPropSheetDlg
Handle to the property sheet's window.

hPageDlg
Handle to the page's window.

Return Values

Returns the zero-based index of the property sheet page specified by *hPageDlg* if successful. Otherwise, it returns –1.

Requirements

Version 5.80 and later of Comctl32.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 prsht.h.

See Also

PropSheet_GetCurrentPageHwnd, **GetParent**

PropSheet_IdToIndex

Takes the resource identifier (ID) of a property sheet page and returns its zero-based index. You can use this macro or send the **PSM_IDTOINDEX** message explicitly.

```
int PropSheet_IdToIndex(
  HWND hPropSheetDlg,
  int iPageID
);
```

Parameters

hPropSheetDlg
Handle to the property sheet window.

iPageID
Resource ID of the page.

Return Values

Returns the zero-based index of the property sheet page specified by *iPageID* if successful. Otherwise, it returns –1.

Requirements

Version 5.80 and later of Comctl32.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 prsht.h.

See Also

PropSheet_IndexToId

PropSheet_IndexToHwnd

Takes the index of a property sheet page and returns its window handle. You can use this macro or send the **PSM_INDEXTOHWND** message explicitly.

```
HWND PropSheet_IndexToHwnd(
  HWND hPropSheetDlg,
  int iPageIndex
);
```

Parameters

hPropSheetDlg
Handle to the property sheet page's window.

iPageIndex
Zero-based index of the page.

Return Values

Returns the handle to the property sheet page's window specified by *iPageIndex* if successful. Otherwise, it returns zero.

Requirements

Version 5.80 and later of Comctl32.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 prsht.h.

See Also

PropSheet_HwndToIndex

PropSheet_IndexToId

Takes the index of a property sheet page and returns its resource identifier (ID). You can use this macro or send the **PSM_INDEXTOID** message explicitly.

```
int PropSheet_IndexToId(
  HWND hPropSheetDlg,
  int iPageIndex
);
```

Parameters

hPropSheetDlg
Handle to the property sheet.

iPageIndex
Zero-based index of the page.

Return Values

Returns the resource ID of the property sheet page specified by *iPageIndex* if successful. Otherwise, it returns zero.

Requirements

Version 5.80 and later of Comctl32.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 prsht.h.

See Also

PSM_INDEXTOID

PropSheet_IndexToPage

Takes the index of a property sheet page and returns its HPROPSHEETPAGE handle. You can use this macro or send the **PSM_INDEXTOPAGE** message explicitly.

```
HPROPSHEETPAGE PropSheet_IndexToPage(
  HWND hPropSheetDlg,
  int iPageIndex
);
```

Parameters

hPropSheetDlg
Handle to the property sheet window.

iPageIndex
Zero-based ndex of the page.

Return Values

Returns the HPROPSHEETPAGE handle of the property sheet page specified by *iPageIndex* if successful. Otherwise, it returns zero.

Requirements

Version 5.80 and later of Comctl32.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 prsht.h.

See Also

PropSheet_PageToIndex

PropSheet_InsertPage

Inserts a new page into an existing property sheet. It can be inserted either at a specified index or after a specified page. You can use this macro or send the **PSM_INSERTPAGE** message explicitly.

```
BOOL PropSheet_InsertPage (
    HWND hPropSheetDlg,
    int index,
    HPROPSHEETPAGE hpage
);

- or -

BOOL PropSheet_InsertPage (
    HWND hPropSheetDlg,
    HPROPSHEETPAGE hpageInsertAfter,
    HPROPSHEETPAGE hpage
);
```

Parameters

hPropSheetDlg
Handle to the property sheet.

wParam
Location where the page is to be inserted. Set *wParam* to NULL to make the new page the first page. To specify where the new page is to be inserted, you can either pass an index or an existing page's HPROPSHEETPAGE handle.

index
If *wParam* is less than MAXUSHORT, it specifies the zero-based index for the new page. For example, to make the inserted page the third one on the property sheet, set *index* to 2. To make it the first page, set *index* to 0. If *index* has a value greater than the number of pages and less than MAXUSHORT, the page will be appended.

hpageInsertAfter
If you set *wParam* to an existing page's HPROPSHEETPAGE handle, the new page will be inserted after it.

hpage
Handle to the page to be inserted. It must first be created by a call to the **CreatePropertySheetPage** function.

Return Values

Returns a nonzero value if the page was successfully inserted. Otherwise, it returns zero.

Remarks

The pages after the insertion point are shifted to the right to accommodate the new page.

The property sheet is not resized to fit the new page. Do not make the new page larger than the property sheet's largest page.

Version 5.80 and later of Comctl32.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 prsht.h.

PropSheet_IsDialogMessage

Passes a message to a property sheet dialog box and indicates whether the dialog box processed the message. You can use this macro or send the **PSM_ISDIALOGMESSAGE** message explicitly.

```
BOOL PropSheet_IsDialogMessage(
    HWND hDlg,
    LPMSG pMsg
);
```

Parameters

hDlg
Handle to the property sheet.

pMsg
Address of an **MSG** structure that contains the message to be checked.

Return Values

Returns TRUE if the macro has been processed, or FALSE if the macro has not been processed.

Remarks

Your message loop should use the **PSM_ISDIALOGMESSAGE** message with modeless property sheets to pass messages to the property sheet dialog box.

If the return value indicates that the **PSM_ISDIALOGMESSAGE** message was processed, it must not be passed to the **TranslateMessage** or **DispatchMessage** function.

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 prsht.h.

See Also

PropertySheet

PropSheet_PageToIndex

Takes the HPROPSHEETPAGE handle of a property sheet page and returns its zero-based index. You can use this macro or send the **PSM_PAGETOINDEX** message explicitly.

```
int PropSheet_PageToIndex(
  HWND hPropSheetDlg,
  HPROPSHEETPAGE hPage,
);
```

Parameters

hPropSheetDlg
Handle to the property sheet.

hPage
HPROPSHEETPAGE handle to the property sheet page.

Return Values

Returns the zero-based index of the property sheet page specified by *hPage* if successful. Otherwise, it returns –1.

Requirements

Version 5.80 and later of Comctl32.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 prsht.h.

See Also

CreatePropertySheetPage, **PropSheet_IndexToPage**

PropSheet_PressButton

Simulates the selection of a property sheet button. You can use this macro or send the **PSM_PRESSBUTTON** message explicitly.

```
BOOL PropSheet_PressButton(
    HWND hPropSheetDlg,
    int iButton
);
```

Parameters

hPropSheetDlg
Handle to the property sheet.

iButton
Index of the button to select. This parameter can be one of the following values:

PSBTN_APPLYNOW	Selects the Apply button.
PSBTN_BACK	Selects the Back button.
PSBTN_CANCEL	Selects the Cancel button.
PSBTN_FINISH	Selects the Finish button.
PSBTN_HELP	Selects the Help button.
PSBTN_NEXT	Selects the Next button.
PSBTN_OK	Selects the OK button.

Return Values

No return value.

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 prsht.h.

PropSheet_QuerySiblings

Causes a property sheet to send the **PSM_QUERYSIBLINGS** message to each of its pages. You can use this macro or send the **PSM_QUERYSIBLINGS** message explicitly.

```
int PropSheet_QuerySiblings(
    HWND hPropSheetDlg,
    WPARAM param1,
    LPARAM param2
);
```

Parameters

hPropSheetDlg
Handle to the property sheet.

param1
First application-defined parameter.

param2
Second application-defined parameter.

Return Values

Returns the nonzero value from a page in the property sheet, or zero if no page returns a nonzero value.

Remarks

If a page returns a nonzero value, the property sheet does not send the message to subsequent pages.

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 prsht.h.

PropSheet_RebootSystem

Indicates the system needs to be restarted for the changes to take effect. You can use this macro or send the **PSM_REBOOTSYSTEM** message explicitly.

```
VOID PropSheet_RebootSystem(
    HWND hPropSheetDlg
);
```

Parameters

hPropSheetDlg
Handle to the property sheet.

Return Values

No return value.

Remarks

An application should send the PSM_REBOOTSYSTEM message only in response to the **PSN_APPLY** or **PSN_KILLACTIVE** notification message.

This macro causes the **PropertySheet** function to return the ID_PSREBOOTSYSTEM value, but only if the user clicks the OK button to close the property sheet. It is the application's responsibility to reboot the system, which can be done by using the **ExitWindowsEx** function.

This macro supersedes all **PropSheet_RebootSystem** macros that precede or follow it.

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 prsht.h.

See Also

PSM_RESTARTWINDOWS

PropSheet_RemovePage

Removes a page from a property sheet. You can use this macro or send the **PSM_REMOVEPAGE** message explicitly.

```
VOID PropSheet_RemovePage(
    HWND hPropSheetDlg,
    int index,
```

```
    HPROPSHEETPAGE hpage
);
```

Parameters

hPropSheetDlg
Handle to the property sheet.

index and *hpage*
Zero-based index of the page and the handle to the page to remove, respectively. An application can specify the index or the handle, or both. If both are specified, *hpage* takes precedence.

Return Values

No return value.

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 prsht.h.

PropSheet_RestartWindows

Sends a **PSM_RESTARTWINDOWS** message indicating that Windows needs to be restarted for changes to take effect. You can use this macro or send the PSM_RESTARTWINDOWS message explicitly.

```
VOID PropSheet_RestartWindows(
    HWND hPropSheetDlg
);
```

Parameters

hPropSheetDlg
Handle to the property sheet.

Return Values

No return value.

Remarks

An application should send the PSM_RESTARTWINDOWS message only in response to the **PSN_APPLY** or **PSN_KILLACTIVE** notification message.

The **PSM_RESTARTWINDOWS** message causes the **PropertySheet** function to return the ID_PSRESTARTWINDOWS value, but only if the user clicks the OK button to close

the property sheet. It is the application's responsibility to restart Windows, which can be done by using the **ExitWindowsEx** function.

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 prsht.h.

PropSheet_SetCurSel

Activates the specified page in a property sheet. You can use this macro or send the **PSM_SETCURSEL** message explicitly.

```
BOOL PropSheet_SetCurSel(
    HWND hPropSheetDlg,
    HPROPSHEETPAGE hpage,
    int Index
);
```

Parameters

hPropSheetDlg
Handle to the property sheet.

index and *hpage*
Zero-based index of the page and the handle to the page to activate, respectively. An application can specify the index or the handle, or both. If both are specified, *hpage* takes precedence.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

The window that is losing the activation receives the **PSN_KILLACTIVE** notification message, and the window that is gaining the activation receives the **PSN_SETACTIVE** notification message.

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 prsht.h.

PropSheet_SetCurSelByID

Activates the specified page in a property sheet based on the resource identifier of the page. You can use this macro or send the **PSM_SETCURSELID** message explicitly.

```
BOOL PropSheet_SetCurSelByID(
    HWND hPropSheetDlg,
    int id
);
```

Parameters

hPropSheetDlg
Handle to the property sheet.

id
Resource identifier of the page to activate.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

The window that is losing the activation receives the **PSN_KILLACTIVE** notification message, and the window that is gaining the activation receives the **PSN_SETACTIVE** notification message.

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 prsht.h.

PropSheet_SetFinishText

Sets the text of the Finish button in a wizard, shows and enables the button, and hides the Next and Back buttons. You can use this macro or send the **PSM_SETFINISHTEXT** message explicitly.

```
VOID PropSheet_SetFinishText(
    HWND hPropSheetDlg,
    LPTSTR lpszText
);
```

Parameters

hPropSheetDlg
Window handle of the wizard.

lpszText
Address of the new text for the Finish button.

Return Values

No return value.

Remarks

This macro causes the DM_SETDEFID message to be sent to the property sheet dialog box. The *wParam* parameter specifies the identifier of the Finish button.

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 prsht.h.

PropSheet_SetHeaderSubTitle

Sets the subtitle text for the header of a wizard's interior page. You can use this macro or send the **PSM_SETHEADERSUBTITLE** message explicitly.

```
VOID PropSheet_SetHeaderSubTitle(
  HWND hWizardDlg,
  int iPageIndex,
  LPCSTR pszHeaderSubTitle
);
```

Parameters

hWizardDlg
Handle to the wizard's window.

iPageIndex
Zero-based index of the page.

pszHeaderSubTitle
New header subtitle.

Return Values

No return value.

Remarks

If you specify the current page, it will immediately be repainted to display the new subtitle.

Requirements

Version 5.80 and later of Comctl32.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 prsht.h.

See Also

PropSheet_HwndToIndex, **PropSheet_IdToIndex**, **PropSheet_PageToIndex**

PropSheet_SetHeaderTitle

Sets the title text for the header of a wizard's interior page. You can use this macro or send the **PSM_SETHEADERTITLE** message explicitly.

```
int PropSheet_SetHeaderTitle(
  HWND hWizardDlg,
  int iPageIndex,
  LPCSTR pszHeaderTitle
);
```

Parameters

hWizardDlg
Handle to the wizard's window.

iPageIndex
Zero-based index of the page.

pszHeaderTitle
New header title.

Return Values

No return value.

Remarks

If you specify the current page, it will immediately be repainted to display the new title.

Requirements

Version 5.80 and later of Comctl32.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 prsht.h.

PropSheet_SetTitle

Sets the title of a property sheet. You can use this macro or send the **PSM_SETTITLE** message explicitly.

```
VOID PropSheet_SetTitle(
    HWND hPropSheetDlg,
    DWORD dwStyle,
    LPTSTR lpszText
);
```

Parameters

hPropSheetDlg
Handle to the property sheet.

dwStyle
Flag that indicates whether to include the prefix "Properties for" with the specified title string. If *dwStyle* is the PSH_PROPTITLE value, the prefix is included. Otherwise, the prefix is not used.

lpszText
Address of a buffer that contains the title string. If the high-order word of this parameter is NULL, the property sheet loads the string resource specified in the low-order word.

Return Values

No return value.

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 prsht.h.

PropSheet_SetWizButtons

Enables or disables the Back, Next, and Finish buttons in a wizard by posting a **PSM_SETWIZBUTTONS** message. You can use this macro or send the PSM_SETWIZBUTTONS message explicitly.

```
VOID PropSheet_SetWizButtons(
    HWND  hPropSheetDlg,
    DWORD dwFlags
);
```

Parameters

hPropSheetDlg
Handle to the property sheet.

dwFlags
A value that specifies which wizard buttons are enabled. You can combine one or more of the following flags:

PSWIZB_BACK	Enable the Back button. If this flag is not set, the Back button is displayed as disabled.
PSWIZB_DISABLEDFINISH	Display a disabled Finish button.
PSWIZB_FINISH	Display an enabled Finish button.
PSWIZB_NEXT	Enable the Next button. If this flag is not set, the Next button is displayed as disabled.

Return Values

No return value.

Remarks

This macro uses **PostMessage** to send the **PSM_SETWIZBUTTONS** message. If your notification handler calls **PropSheet_SetWizButtons**, do not do anything that will affect window focus until after the handler returns. For example, if you call **MessageBox** immediately after calling **PropSheet_SetWizButtons**, the message box will receive focus. Since messages sent with **PostMessage** are not delivered until they reach the head of the message queue, the **PSM_SETWIZBUTTONS** message will not be delivered until after the wizard has lost focus to the message box. As a result, the property sheet will not be able to properly set the focus for the buttons.

Wizards display either three or four buttons below each page. This message is used to specify which buttons are enabled. Wizards normally display Back, Cancel, and either a Next or a Finish button. You typically enable only the Next button for the welcome page, Next and Back for interior pages, and Back and Finish for the completion page. The Cancel button is always enabled. Normally, setting PSWIZB_FINISH or PSWIZB_DISABLEDFINISH replaces the Next button with a Finish button. To display Next and Finish buttons simultaneously, set the PSH_WIZARDHASFINISH FLAG in the

dwFlags member of the wizard's **PROPSHEETHEADER** structure when you create the wizard. Every page will then display all four buttons.

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 prsht.h.

PropSheet_UnChanged

Informs a property sheet that information in a page has reverted to the previously saved state. You can use this macro or send the **PSM_UNCHANGED** message explicitly.

```
VOID PropSheet_UnChanged(hPropSheetDlg, hwndPage)
    HWND hPropSheetDlg,
    HWND hwndPage
);
```

Parameters

hPropSheetDlg
Handle to the property sheet.

hwndPage
Handle to the page that has reverted to the previously saved state.

Return Values

No return value.

Remarks

The property sheet disables the Apply Now button if no other pages have registered changes with the property sheet.

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 prsht.h.

Property Sheet Notifications

PSN_APPLY

Indicates that the user clicked the OK, Close, or Apply button and wants all changes to take effect. This notification is sent in the form of a **WM_NOTIFY** message.

```
PSN_APPLY
    lppsn = (LPPSHNOTIFY) lParam;
```

Parameters

lppsn
Address of a **PSHNOTIFY** structure that contains information about the notification.

Return Values

Return PSNRET_INVALID_NOCHANGEPAGE to prevent the changes from taking effect and to return the focus to the page. Return PSNRET_NOERROR to accept the changes and allow the property sheet to be destroyed. To set the return value, the dialog box procedure for the page must use the **SetWindowLong** function with the DWL_MSGRESULT value, and the dialog box procedure must return TRUE.

Remarks

The **lParam** member of the **PSHNOTIFY** structure pointed to by *lppsn* will be TRUE if the user clicked the OK button. It will also be TRUE if the PSM_CANCELTOCLOSE message has been sent and the user clicked the Close button. It will be FALSE if the user clicked the Apply button. The **PSHNOTIFY** structure contains an **NMHDR** structure as its first member, **hdr**. The **hwndFrom** member of this **NMHDR** structure contains the handle to the property sheet.

Do not call the **EndDialog** function when processing this notification.

The property sheet is destroyed if the user clicks the OK button and the application returns the PSNRET_NOERROR value in response to this notification.

To receive this notification, a page must set the DWL_MSGRESULT value to FALSE in response to the **PSN_KILLACTIVE** notification message.

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 prsht.h.

PSN_GETOBJECT

Sent by a property sheet to request a drop target object when the cursor passes over one of the tab control's buttons.

```
PSN_GETOBJECT
    lpnmon = (LPNMOBJECTNOTIFY) lParam;
```

Parameters

lpnmon

Address of an **NMOBJECTNOTIFY** structure that, on entry, contains information about the notification. If this notification is processed, you must insert object information into this structure.

Return Values

The application processing this notification must return zero.

Remarks

To provide an object, an application must set values in some members of the **NMOBJECTNOTIFY** structure at *lpnmon*. The **pObject** member must be set to a valid object pointer, and the **hResult** member must be set to a success flag. To comply with COM standards, always increment the object's reference count when providing an object pointer.

If an application does not provide an object, it must set **pObject** to NULL and **hResult** to a failure flag.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

PSN_HELP

Notifies a page that the user has clicked the Help button. This notification message is sent in the form of a **WM_NOTIFY** message.

```
PSN_HELP
    lppsn = (LPPSHNOTIFY) lParam;
```

Parameters

lppsn

Address of a **PSHNOTIFY** structure that contains information about the notification. This structure contains an **NMHDR** structure as its first member, **hdr**. The **hwndFrom** member of this **NMHDR** structure contains the handle to the property sheet. The**lParam** member of the **PSHNOTIFY** structure does not contain any information.

Return Values

No return value.

Remarks

An application should display Help information for the page.

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 prsht.h.

PSN_KILLACTIVE

Notifies a page that it is about to lose activation either because another page is being activated or the user has clicked the OK button. This notification message is sent in the form of a **WM_NOTIFY** message.

```
PSN_KILLACTIVE
    lppsn = (LPPSHNOTIFY) lParam;
```

Parameters

lppsn

Address of a **PSHNOTIFY** structure that contains information about the notification. This structure contains an **NMHDR** structure as its first member, **hdr**. The **hwndFrom** member of this **NMHDR** structure contains the handle to the property sheet. The **lParam** member of the **PSHNOTIFY** structure does not contain any information.

Return Values

Returns TRUE to prevent the page from losing the activation, or FALSE to allow it.

Remarks

An application should validate the information the user has typed.

To set the return value, the dialog box procedure for the page must use the **SetWindowLong** function with the DWL_MSGRESULT value, and the dialog box procedure must return TRUE.

If the dialog box procedure sets DWL_MSGRESULT to TRUE, it should display a message box to explain the problem to the user.

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 prsht.h.

PSN_QUERYCANCEL

Indicates the user clicked the Cancel button. This notification message is sent in the form of a **WM_NOTIFY** message.

```
PSN_QUERYCANCEL
    lppsn = (LPPSHNOTIFY) lParam;
```

Parameters

lppsn
Address of a **PSHNOTIFY** structure that contains information about the notification. This structure contains an **NMHDR** structure as its first member, **hdr**. The **hwndFrom** member of this **NMHDR** structure contains the handle to the property sheet. The **lParam** member of the **PSHNOTIFY** structure does not contain any information.

Return Values

Returns TRUE to prevent the cancel operation, or FALSE to allow it.

Remarks

A property sheet page can use this notification message to ask the user to verify the cancel operation.

To set the return value, the dialog box procedure for the page must use the **SetWindowLong** function with the DWL_MSGRESULT value, and the dialog box procedure must return TRUE.

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 prsht.h.

PSN_QUERYINITIALFOCUS

Sent by a property sheet to provide a property sheet page an opportunity to specify which dialog box control should receive the initial focus. This notification is sent in the form of a **WM_NOTIFY** message.

```
PSN_QUERYINITIALFOCUS
    lppsn = (LPPSHNOTIFY) lParam;
```

Parameters

lppsn

Pointer to a **PSHNOTIFY** structure. Cast the **lParam** member of this structure to an **HWND** type, to retrieve the handle of the control that will be given focus by default. The structure contains an **NMHDR** structure as its first member, **hdr**. The **hwndFrom** member of this **NMHDR** structure contains the handle to the property sheet.

Return Values

To specify which control should receive focus, return the control's handle. Otherwise, return zero and focus will go to the default control. To set the return value, the dialog box procedure must call the **SetWindowLong** function with a DWL_MSGRESULT value and return TRUE.

Example

This code fragment implements a simple handler for PSN_QUERYINITIALFOCUS. It requests that initial focus be given to the Location control (IDC_LOCATION).

```
case PSN_QUERYINITIALFOCUS :
    SetWindowLong(hDlg,DWL_MSGRESULT, (LPARAM)GetDlgItem(hDlg, IDC_LOCATION));
    return TRUE;
...
```

Remarks

An application must not call the **SetFocus** function while handling this notification. Return the handle of the control that should receive focus, and the property sheet manager will handle the focus change.

The PSN_QUERYINITIALFOCUS notification is not sent if the property sheet manager determines that no control on the page should receive focus.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.

PSN_RESET

Notifies a page that the property sheet is about to be destroyed. This notification message is sent in the form of a **WM_NOTIFY** message.

```
PSN_RESET
    lppsn = (LPPSHNOTIFY) lParam;
```

Parameters

lppsn
Address of a **PSHNOTIFY** structure that contains information about the notification.

Return Values

No return value.

Remarks

All changes made since the last PSN_APPLY notification are canceled. The **lParam** member of the **PSHNOTIFY** structure pointed to by *lppsn* will be set to TRUE if the user clicked the "X" button in the upper-right corner of the property sheet. It will be FALSE if the user clicked the Cancel button. The **PSHNOTIFY** structure contains an **NMHDR** structure as its first member, **hdr**. The **hwndFrom** member of this **NMHDR** structure contains the handle to the property sheet.

An application can use this notification message as an opportunity to perform cleanup operations.

Do not call the **EndDialog** function when processing this notification.

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 prsht.h.

PSN_SETACTIVE

Notifies a page that it is about to be activated. This notification message is sent in the form of a **WM_NOTIFY** message.

```
PSN_SETACTIVE
    lppsn = (LPPSHNOTIFY) lParam;
```

Parameters

lppsn

Address of a **PSHNOTIFY** structure that contains information about the notification. This structure contains an **NMHDR** structure as its first member, **hdr**. The **hwndFrom** member of this **NMHDR** structure contains the handle to the property sheet. The **lParam** member of the **PSHNOTIFY** structure does not contain any information.

Return Values

Returns zero to accept the activation, or –1 to activate the next or the previous page (depending on whether the user clicked the Next or Back button). To set the activation to a particular page, return the resource identifier of the page.

Remarks

The PSN_SETACTIVE notification message is sent before the page is visible. An application can use this notification to initialize data in the page.

To set the return value, the dialog box procedure for the page must use the **SetWindowLong** function with the DWL_MSGRESULT value, and the dialog box procedure must return TRUE.

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 prsht.h.

See Also

NMHDR

PSN_TRANSLATEACCELERATOR

Notifies a property sheet that a keyboard message has been received. It provides the page an opportunity to do private keyboard accelerator translation. This notification is sent in the form of a WM_NOTIFY message.

```
PSN_TRANSLATEACCELERATOR
    lppsn = (LPPSHNOTIFY) lParam;
```

Parameters

lppsn

A pointer to a **PSHNOTIFY** structure that contains information about the notification. This structure contains an **NMHDR** structure as its first member, **hdr**. The **hwndFrom** member of the **NMHDR** structure contains the handle to the property sheet. The **lParam** member of the **PSHNOTIFY** structure is a pointer to the message's **MSG** structure. It can be cast to an **LPMSG** type, to get access to the parameters of the message to be translated.

Return Values

Return PSNRET_MESSAGEHANDLED to indicate that no further processing is necessary. Return PSNRET_NOERROR to request normal processing.

Remarks

To set the return value, the dialog box procedure for the page must use the **SetWindowLong** function with the DWL_MSGRESULT value. The dialog box procedure must return TRUE.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.

PSN_WIZBACK

Notifies a page that the user has clicked the Back button in a wizard. This notification message is sent in the form of a **WM_NOTIFY** message.

```
PSN_WIZBACK
    lppsn = (LPPSHNOTIFY) lParam;
```

Parameters

lppsn

Address of a **PSHNOTIFY** structure that contains information about the notification. This structure contains an **NMHDR** structure as its first member, **hdr**. The **hwndFrom** member of the **NMHDR** structure contains the handle to the property sheet. The **lParam** member of the **PSHNOTIFY** structure does not contain any information.

Return Values

Return 0 to allow the wizard to go to the previous page. Return –1 to prevent the wizard from changing pages. To display a particular page, return its dialog resource identifier.

Remarks

To set the return value, the dialog box procedure for the page must call the **SetWindowLong** function with the DWL_MSGRESULT value and return TRUE. For example:

```
case PSN_WIZBACK :
    SetWindowLong(hDlg, DWL_MSGRESULT, 0);
    break;
case PSN_WIZNEXT :
...
```

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 prsht.h.

See Also

PSN_WIZNEXT

PSN_WIZFINISH

Notifies a page that the user has clicked the Finish button in a wizard. This notification message is sent in the form of a **WM_NOTIFY** message.

```
PSN_WIZFINISH
    lppsn = (LPPSHNOTIFY) lParam;
```

Parameters

lppsn

Address of a **PSHNOTIFY** structure that contains information about the notification. The first member of this structure, **hdr**, is an **NMHDR** structure. The **hwndFrom** member of this **NMHDR** structure contains the handle to the property sheet. The **lParam** member of the **PSHNOTIFY** structure does not contain any information.

Return Values

- Return TRUE to prevent the wizard from finishing.

- Version 5.80. Return a window handle to prevent the wizard from finishing. The wizard will set the focus to that window. The window must be owned by the wizard page.
- Return FALSE to allow the wizard to finish.

Remarks

To set the return value, the dialog box procedure for the page must use the **SetWindowLong** function with the DWL_MSGRESULT value, and the dialog box procedure must return TRUE.

Version 5.80. If your application returns TRUE to prevent a wizard from finishing, it has no control over which window on the page receives focus. Applications that need to stop a wizard from finishing should normally do so by returning the handle of the window on wizard page that is to receive focus.

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 prsht.h.

PSN_WIZNEXT

Notifies a page that the user has clicked the Next button in a wizard. This notification message is sent in the form of a **WM_NOTIFY** message.

```
PSN_WIZNEXT
    lppsn = (LPPSHNOTIFY) lParam;
```

Parameters

lppsn

Address of a **PSHNOTIFY** structure that contains information about the notification. This structure contains an **NMHDR** structure as its first member, **hdr**. The **hwndFrom** member of the **NMHDR** structure contains the handle to the property sheet. The **lParam** member of the **PSHNOTIFY** structure does not contain any information.

Return Values

Return 0 to allow the wizard to go to the next page. Return –1 to prevent the wizard from changing pages. To display a particular page, return its dialog resource identifier.

Remarks

To set the return value, the dialog box procedure for the page must call the **SetWindowLong** function with the DWL_MSGRESULT value and return TRUE. For example:

```
case PSN_WIZNEXT :
    SetWindowLong(hDlg, DWL_MSGRESULT, 0);
    break;
case PSN_WIZBACK :
...
```

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 prsht.h.

See Also

PSN_WIZBACK

Property Sheet Structures

PROPSHEETHEADER

Defines the frame and pages of a property sheet.

```
typedef struct _PROPSHEETHEADER {
    DWORD dwSize;
    DWORD dwFlags;
    HWND  hwndParent;
    HINSTANCE hInstance;
    union {
        HICON hIcon;
        LPCTSTR pszIcon;
        };
    LPCTSTR pszCaption;
    UINT nPages;
    union {
        UINT nStartPage;
        LPCTSTR pStartPage;
        };
    union {
```

(continued)

(continued)

```
        LPCPROPSHEETPAGE ppsp;
        HPROPSHEETPAGE FAR *phpage;
        };
    PFNPROPSHEETCALLBACK pfnCallback;

#if (_WIN32_IE >= 0x0500)
    union {
        HBITMAP hbmWatermark;
        LPCTSTR pszbmWatermark;
        };
    HPALETTE hplWatermark;
    union {
        HBITMAP hbmHeader;
        LPCSTR pszbmHeader;
        };
#endif
} PROPSHEETHEADER, FAR *LPPROPSHEETHEADER;
```

Members

dwSize

Size, in bytes, of this structure. The property sheet manager uses this member to determine which version of the **PROPSHEETHEADER** structure you are using. For more information, see the *Remarks*.

dwFlags

Flags that indicate which options to use when creating the property sheet page. This member can be a combination of the following values:

PSH_DEFAULT
: Uses the default meaning for all structure members.

PSH_HASHELP
: Permits property sheet pages to display a Help button. You must also set the PSP_HASHELP flag in the page's **PROPSHEETPAGE** structure when the page is created. If any of the initial property sheet pages enable a Help button, PSH_HASHELP will be set automatically. If none of the initial pages enable a Help button, you must explicitly set PSH_HASHELP if you want to have Help buttons on any pages that might be added later.

PSH_HEADER
: Version 5.80. Indicates that a header bitmap will be used with a Wizard97 wizard. You must also set the PSH_WIZARD97 flag. The header bitmap is obtained from the **pszbmHeader** member, unless the PSH_USEHBMHEADER flag is also set. In that case, the header bitmap is obtained from the **hbmHeader** member.

PSH_MODELESS
: Causes the **PropertySheet** function to create the property sheet as a modeless dialog box instead of as a modal dialog box. When this flag is set, **PropertySheet**

returns immediately after the dialog box is created, and the return value from **PropertySheet** is the window handle to the property sheet dialog box.

PSH_NOAPPLYNOW
Removes the Apply button.

PSH_NOCONTEXTHELP
Version 5.80. Removes the context-sensitive Help button ("?"), which is usually present on the caption bar of property sheets. This flag is not valid for wizards. See *Property Sheets* for a discussion of how to remove the caption bar help button for earlier versions of the common controls.

PSH_PROPSHEETPAGE
Uses the **ppsp** member and ignores the **phpage** member when creating the pages for the property sheet.

PSH_PROPTITLE
Displays the string "Properties for," followed by the string specified by the **pszCaption** member, in the title bar of the property sheet.

PSH_RTLREADING
Reverses the direction in which **pszCaption** is displayed. Normal windows display all text, including **pszCaption**, left-to-right (LTR). For languages such as Hebrew or Arabic, that read right-to-left (RTL), a window can be *mirrored* and all text will be displayed RTL. If PSP_RTLREADING is set, **pszCaption** will instead read RTL in a normal parent window, and LTR in a mirrored parent window.

PSH_STRETCHWATERMARK
Stretches the watermark in Internet Explorer 4.0-compatible Wizard97-style wizards.

Note This style flag is only included to provide backward compatibility for certain applications. Its use is not recommended and it is only supported by common controls versions 4.0 and 4.01. With common controls version 5.80 and later, this flag is ignored.

PSH_USECALLBACK
Calls the function specified by the **pfnCallback** member when initializing the property sheet defined by this structure.

PSH_USEHBMHEADER
Version 5.80. Obtains the header bitmap from the **hbmHeader** member instead of the **pszbmHeader** member. You must also set PSH_WIZARD97 and PSH_HEADER.

PSH_USEHBMWATERMARK
Version 5.80. Obtains the watermark bitmap from the **hbmWatermark** member instead of the **pszbmWatermark** member. You must also set PSH_WIZARD97 and PSH_WATERMARK.

PSH_USEHICON
Uses **hIcon** as the small icon in the title bar of the property sheet dialog box.

PSH_USEHPLWATERMARK
Version 5.80. Uses the **HPALETTE** structure pointed to by the **hplWatermark** member instead of the default palette to draw the watermark bitmap and/or header bitmap for a Wizard97 wizard. You must also set PSH_WIZARD97, and PSH_WATERMARK or PSH_HEADER.

PSH_USEICONID
Uses **pszIcon** as the name of the icon resource to load and use as the small icon in the title bar of the property sheet dialog box.

PSH_USEPAGELANG
Version 5.80. Specifies that the language for the property sheet will be taken from the first page's resource.

PSH_USEPSTARTPAGE
Uses the **pStartPage** member instead of the **nStartPage** member when displaying the initial page of the property sheet.

PSH_WATERMARK
Version 5.80. Specifies that a watermark bitmap will be used with a Wizard97 wizard. You must also set the PSH_WIZARD97 flag. The watermark bitmap is obtained from the **pszbmWatermark** member, unless PSH_USEHBMWATERMARK is set. In that case, the header bitmap is obtained from the **hbmWatermark** member.

PSH_WIZARD
Creates a wizard property sheet.

PSH_WIZARD97
Version 5.80. Creates a Wizard97-style property sheet that allows a header and/or watermark bitmap to be displayed in the background.

PSH_WIZARDCONTEXTHELP
Adds a context-sensitive Help button ("?"), which is usually absent from the caption bar of a wizard. This flag is not valid for regular property sheets.

PSH_WIZARDHASFINISH
Always displays the Finish button on the wizard. You must also set either PSH_WIZARD or PSH_WIZARD97.

PSH_WIZARD_LITE
Version 5.80. Uses the Wizard-lite style. This style is similar in appearance to PSH_WIZARD97, but it is implemented much like PSH_WIZARD. There are few restrictions on how the pages are formatted. For instance, there are no enforced borders, and the PSH_WIZARD_LITE style does not paint the watermark and header bitmaps for you the way Wizard97 does.

hwndParent
Handle to the property sheet's owner window.

hInstance
Handle to the instance from which to load the icon or title string resource. If the **pszIcon** or **pszCaption** member identifies a resource to load, this member must be specified.

hIcon
Handle to the icon to use as the small icon in the title bar of the property sheet dialog box. If the **dwFlags** member does not include PSH_USEHICON, this member is ignored. This member is declared as a union with **pszIcon**.

pszIcon
Icon resource to use as the small icon in the title bar of the property sheet dialog box. This member can specify either the identifier of the icon resource or the address of the string that specifies the name of the icon resource. If the **dwFlags** member does not include PSH_USEICONID, this member is ignored. This member is declared as a union with **hIcon**.

pszCaption
Title of the property sheet dialog box. This member can specify either the identifier of a string resource or the address of a string that specifies the title. If the **dwFlags** member includes PSH_PROPTITLE, the string "Properties for" is inserted at the beginning of the title. This field is ignored for wizards.

nPages
Number of elements in the **phpage** array.

nStartPage
Zero-based index of the initial page that appears when the property sheet dialog box is created. This member is declared as a union with **pStartPage**.

pStartPage
Name of the initial page that appears when the property sheet dialog box is created. This member can specify either the identifier of a string resource or the address of a string that specifies the name. This member is declared as a union with **nStartPage**.

ppsp
Address of an array of **PROPSHEETPAGE** structures that define the pages in the property sheet. If the **dwFlags** member does not include PSH_PROPSHEETPAGE, this member is ignored. Note that the **PROPSHEETPAGE** structure is variable in size. Applications that parse the array pointed to by **ppsp** must take the size of each page into account. This member is declared as a union with **phpage**.

phpage
Address of an array of handles to the property sheet pages. Each handle must have been created by a previous call to the **CreatePropertySheetPage** function. If the **dwFlags** member includes PSH_PROPSHEETPAGE, **phpage** is ignored and should be set to NULL. When the **PropertySheet** function returns, any HPROPSHEETPAGE handles in the **phpage** array will have been destroyed. This member is declared as a union with **ppsp**.

pfnCallback
Address of an application-defined callback function that is called when the property sheet is initialized. For more information about the callback function, see the description of the **PropSheetProc** function. If the **dwFlags** member does not include PSH_USECALLBACK, this member is ignored.

hbmWatermark
Version 5.80. Handle to the watermark bitmap. If the **dwFlags** member does not include PSH_USEHBMWATERMARK, this member is ignored.

pszbmWatermark
Version 5.80. Bitmap resource to use as the watermark. This member can specify either the identifier of the bitmap resource or the address of the string that specifies the name of the bitmap resource. If the **dwFlags** member includes PSH_USEHBMWATERMARK, this member is ignored.

hplWatermark
Version 5.80. **HPALETTE** structure used for drawing the watermark bitmap and/or header bitmap. If the **dwFlags** member does not include PSH_USEHPLWATERMARK, this member is ignored.

hbmHeader
Version 5.80. Handle to the header bitmap. If the **dwFlags** member does not include PSH_USEHBMHEADER, this member is ignored.

pszbmHeader
Version 5.80. Bitmap resource to use as the header. This member can specify either the identifier of the bitmap resource or the address of the string that specifies the name of the bitmap resource. If the **dwFlags** member includes PSH_USEHBMHEADER, this member is ignored.

Remarks

If the user chooses a setting such as Large Fonts, which enlarges the dialog box, the watermark that is painted on the start and finish pages will be enlarged as well. The size and position of the original bitmap will remain the same. The additional area will be filled with the color of the pixel in the upper-left corner of the bitmap.

Note that several members of this structure are only supported for Comctl32.dll versions 4.71 and later. You can enable these members by including one of the following in your header:

```
#define _WIN32_IE 0x0400 // For version 4.71
```

or

```
#define _WIN32_IE 0x0500 // For version 5.80
```

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 Comctl32.dll, which expect a smaller structure. This includes all systems with Microsoft Windows 95 or Microsoft Windows NT 4.0 that do not have Internet Explorer version 4.0 or later installed. You can run your application on pre-4.71 versions of Comctl32.dll by defining the appropriate version number. However, this may cause problems if your application also needs to run on systems with more recent versions.

You can remain compatible with all Comctl32.dll versions by using the current header files and setting the size of the PROPSHEETHEADER structure appropriately. Before

you initialize the structure, use the **DllGetVersion** function to determine which Comctl32.dll version is installed on the system. If it is version 4.71 or greater, use:

```
psh.dwSize = sizeof(PROPSHEETHEADER);
```

to initialize the dwSize member. For earlier versions, the size of the pre-4.71 structure is given by the PROPSHEETHEADER_V1_SIZE constant. Use:

```
psh.dwSize = PROPSHEETHEADER_V1_SIZE;
```

The three wizard styles, PSH_WIZARD, PSH_WIZARD97, and PSH_WIZARD_LITE, are mutually incompatible. Only one of these style flags should be set.

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 prsht.h.

PROPSHEETPAGE

This structure defines a page in a property sheet.

```
typedef struct _PROPSHEETPAGE {
    DWORD dwSize;
    DWORD dwFlags;
    HINSTANCE hInstance;
    union {
        LPCSTR pszTemplate;
        LPCDLGTEMPLATE pResource;
        };
    union {
        HICON hIcon;
        LPCSTR pszIcon;
        };
    LPCSTR pszTitle;
    DLGPROC pfnDlgProc;
    LPARAM lParam;
    LPFNPSPCALLBACK pfnCallback;
    UINT FAR * pcRefParent;

#if (_WIN32_IE >= 0x0500)
    LPCTSTR pszHeaderTitle;
    LPCTSTR pszHeaderSubTitle;
#endif
} PROPSHEETPAGE, FAR *LPPROPSHEETPAGE;
```

Members

dwSize

Size, in bytes, of this structure. The property sheet manager uses this member to determine which version of the **PROPSHEETHEADER** structure you are using.

dwFlags

Flags that indicate which options to use when creating the property sheet page. This member can be a combination of the following values:

PSP_DEFAULT

Uses the default meaning for all structure members.

PSP_DLGINDIRECT

Creates the page from the dialog box template in memory pointed to by the **pResource** member. The **PropertySheet** function assumes that the template that is in memory is not write-protected. A read-only template will cause an exception in some versions of Microsoft Windows.

PSP_HASHELP

Enables the property sheet Help button when the page is active.

PSP_HIDEHEADER

Version 5.80. Causes the wizard property sheet to hide the header area when the page is selected. If a watermark has been provided, it will be painted on the left side of the page. This flag should be set for welcome and completion pages, and omitted for interior pages.

PSP_PREMATURE

Version 4.71. Causes the page to be created when the property sheet is created. If this flag is not specified, the page will not be created until it is selected the first time.

PSP_RTLREADING

Reverses the direction in which **pszTitle** is displayed. Normal windows display all text, including **pszTitle**, left-to-right (LTR). For languages such as Hebrew or Arabic, that read right-to-left (RTL), a window can be *mirrored* and all text will be displayed RTL. If PSP_RTLREADING is set, **pszTitle** will instead read RTL in a normal parent window, and LTR in a mirrored parent window.

PSP_USECALLBACK

Calls the function specified by the **pfnCallback** member when creating or destroying the property sheet page defined by this structure.

PSP_USEHEADERSUBTITLE

Version 5.80. Displays the string pointed to by the **pszHeaderSubTitle** member as the subtitle of the header area of a Wizard97 page. To use this flag, you must also set the PSH_WIZARD97 flag in the **dwFlags** member of the associated **PROPSHEETHEADER** structure. The PSP_USEHEADERSUBTITLE flag is ignored if PSP_HIDEHEADER is set.

PSP_USEHEADERTITLE

Version 5.80. Displays the string pointed to by the **pszHeaderTitle** member as the title in the header of a Wizard97 interior page. You must also set the

PSH_WIZARD97 flag in the **dwFlags** member of the associated **PROPSHEETHEADER** structure. The PSP_USEHEADERTITLE flag is ignored if PSP_HIDEHEADER is set.

PSP_USEHICON
Uses **hIcon** as the small icon on the tab for the page.

PSP_USEICONID
Uses **pszIcon** as the name of the icon resource to load and use as the small icon on the tab for the page.

PSP_USEREFPARENT
Maintains the reference count specified by the **pcRefParent** member for the lifetime of the property sheet page created from this structure.

PSP_USETITLE
Uses the **pszTitle** member as the title of the property sheet dialog box instead of the title stored in the dialog box template.

hInstance
Handle to the instance from which to load an icon or string resource. If the **pszIcon**, **pszTitle pszHeaderTitle**, or **pszHeaderSubTitle** members identifies a resource to load, **hInstance** must be specified.

pszTemplate
Dialog box template to use to create the page. This member can specify either the resource identifier of the template or the address of a string that specifies the name of the template. If the PSP_DLGINDIRECT flag in the **dwFlags** member is set, **pszTemplate** is ignored. This member is declared as a union with **pResource**.

pResource
Address of a dialog box template in memory. The **PropertySheet** function assumes that the template is not write-protected. A read-only template will cause an exception in some versions of Windows. To use this member, you must set the PSP_DLGINDIRECT flag in the **dwFlags** member. This member is declared as a union with **pszTemplate**.

hIcon
Handle to the icon to use as the icon in the tab of the page. If the **dwFlags** member does not include PSP_USEHICON, this member is ignored. This member is declared as a union with **pszIcon**.

pszIcon
Icon resource to use as the icon in the tab of the page. This member can specify either the identifier of the icon resource or the address of the string that specifies the name of the icon resource. To use this member, you must set the PSP_USEICONID flag in the **dwFlags** member. This member is declared as a union with **hIcon**.

pszTitle
Title of the property sheet dialog box. This title overrides the title specified in the dialog box template. This member can specify either the identifier of a string resource or the address of a string that specifies the title. To use this member, you must set the PSP_USETITLE flag in the **dwFlags** member.

pfnDlgProc
Address of the dialog box procedure for the page. Because the pages are created as modeless dialog boxes, the dialog box procedure must not call the **EndDialog** function.

lParam
When the page is created, a copy of the page's **PROPSHEETPAGE** structure is passed to the dialog box procedure with a WM_INITDIALOG message. The **lParam** member is provided to allow you to pass application-specific information to the dialog box procedure. It has no effect on the page itself. For more information, see *Property Sheet Creation.*

pfnCallback
Address of an application-defined callback function that is called when the page is created and when it is about to be destroyed. For more information about the callback function, see **PropSheetPageProc**. To use this member, you must set the PSP_USECALLBACK flag in the **dwFlags** member.

pcRefParent
Address of the reference count value. To use this member, you must set the PSP_USEREFPARENT flag in the **dwFlags** member.

Note When a property sheet page is created, the value pointed to by **pcRefParent** is incremented. You create a property sheet page implicitly by setting the PSH_PROPSHEETPAGE flag in the **dwFlags** member of **PROPSHEETHEADER** and calling the **PropertySheet** function. You can do it explicitly by using the **CreatePropertySheetPage** function. When a property sheet page is destroyed, the value pointed to by the **pcRefParent** member is decremented. This takes place automatically when the property sheet is destroyed. You can explicitly destroy a property sheet page by using the **DestroyPropertySheetPage** function.

---

pszHeaderTitle
Version 5.80. Title of the header area. To use this member, you must:

- Set the PSP_USEHEADERTITLE flag in the **dwFlags** member.
- Set the PSH_WIZARD97 flag in the **dwFlags** member of the page's **PROPSHEETHEADER** structure.
- Make sure that the PSP_HIDEHEADER flag in the **dwFlags** member is not set.

pszHeaderSubTitle
Version 5.80. Subtitle of the header area. To use this member, you must:

- Set the PSP_USEHEADERSUBTITLE flag in the **dwFlags** member.
- Set the PSH_WIZARD97 flag in the **dwFlags** member of the page's **PROPSHEETHEADER** structure.
- Make sure that the PSP_HIDEHEADER flag in the **dwFlags** member is not set.

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 prsht.h.

PSHNOTIFY

Contains information for the property sheet notification messages.

```
typedef struct _PSHNOTIFY {
    NMHDR hdr;
    LPARAM lParam;
} PSHNOTIFY, FAR *LPPSHNOTIFY;
```

Members

hdr

Address of an **NMHDR** structure that contains additional information about the notification.

lParam

Additional information about this notification. To determine what, if any, information is contained in this member, see the description of the particular notification message.

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 prsht.h.

CHAPTER 22

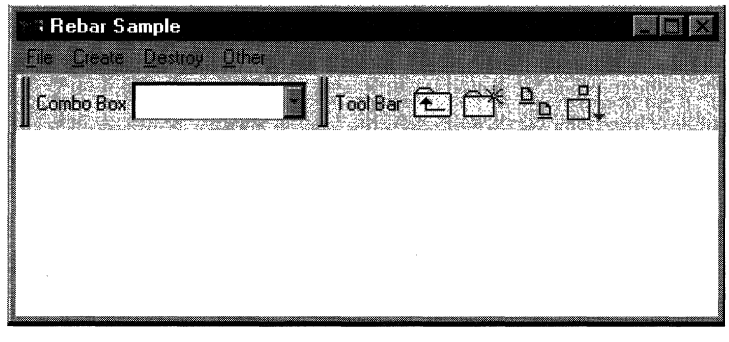
Rebar Controls

Rebar controls act as containers for child windows. An application assigns child windows, which are often other controls, to a rebar control band. Rebar controls contain one or more bands, and each band can have any combination of a gripper bar, a bitmap, a text label, and a child window. However, bands cannot contain more than one child window.

About Rebar Controls

A rebar control displays the child window over a specified background bitmap. As you dynamically reposition a rebar control band, the rebar control manages the size and position of the child window assigned to that band.

The following illustration shows a rebar control that has two bands. One contains a combo box, and the other contains a transparent tool bar control.



Note The rebar control is implemented in versions 4.70 and later of Comctl32.dll.

Rebar Bands and Child Windows

An application defines a rebar band's traits by using the **RB_INSERTBAND** and **RB_SETBANDINFO** messages. These messages accept the address of a **REBARBANDINFO** structure as the *lParam* parameter. The **REBARBANDINFO** structure members define the traits of a given band. To set a band's traits, set the **cbSize** member to indicate the size of the structure, in bytes. Then, set the **fMask** member to indicate which structure members your application is filling.

To assign a child window to a band, include the RBBIM_CHILD flag in the **fMask** member of the **REBARBANDINFO** structure, and then set the **hwndChild** member to

the child window's handle. Applications can set the minimum allowable width and height of a child window in the **cxMinChild** and **cyMinChild** members, respectively.

When a rebar control is destroyed, it destroys any child windows assigned to the bands within it. To prevent the control from destroying child windows assigned to its bands, remove the bands by sending the **RB_DELETEBAND** message, and then reset the parent to another window with the **SetParent** function before destroying the rebar control.

The Rebar Control User Interface

All rebar control bands can be resized, except those that use the RBBS_FIXEDSIZE style. To resize or change the order of bands within the control, click and drag a band's gripper bar. The rebar control automatically resizes and repositions child windows assigned to its bands. Additionally, you can toggle the size of a band by clicking on the band text, if there is any.

The Rebar Control Image List

If an application is using an image list with a rebar control, it must send the **RB_SETBARINFO** message before adding bands to the control. This message accepts the address of a **REBARINFO** structure as the *lParam* parameter. Before sending the message, prepare the **REBARINFO** structure by setting the **cbSize** member to the size of the structure, in bytes. Then, if the rebar control is going to display images on the bands, set the **fMask** member to the RBIM_IMAGELIST flag and assign an image list handle to the **himl** member. If the rebar will not use band images, set **fMask** to zero.

Rebar Control Message Forwarding

A rebar control forwards all **WM_NOTIFY** window messages to its parent window. Additionally, a rebar control forwards any messages sent to it from windows assigned to its bands, like **WM_CHARTOITEM**, **WM_COMMAND**, and others.

Custom Draw Support

Rebar controls support custom draw functionality. For more information, see *About Custom Draw.*

Using Rebar Controls

This section gives sample code that demonstrates how to implement a rebar control.

Creating a Rebar Control

An application creates a rebar control by calling the **CreateWindowEx** function, specifying REBARCLASSNAME as the window class. The application must first register

the window class by calling the **InitCommonControlsEx** function, while specifying the ICC_COOL_CLASSES bit in the accompanying **INITCOMMONCONTROLSEX** structure.

The following sample creates a rebar control with two bands—one that contains a combo box and another that contains a *tool bar*. The sample includes the **RBS_VARHEIGHT** style to allow the control to use variable band height. After creating the rebar control, CreateRebar creates the child windows with calls to two application-defined functions, **CreateComboBox** and **CreateToolbar**. Before adding each band, CreateRebar initializes the **cbSize** member of the **REBARBANDINFO** structure, as required by the **RB_INSERTBAND** message. Then it sets the value of the structure's **fMask** member to reflect which members contain valid data. CreateRebar sets the **cyMinChild** member for each band to allow for the height of the control within it. The **cxMinChild** member is zero to allow the user to completely hide the control within a given band:

```
HWND WINAPI CreateRebar(HWND hwndOwner)
{
   REBARINFO     rbi;
   REBARBANDINFO rbBand;
   RECT          rc;
   HWND   hwndCB, hwndTB, hwndRB;
   DWORD  dwBtnSize;
   INITCOMMONCONTROLSEX icex;

   icex.dwSize = sizeof(INITCOMMONCONTROLSEX);
   icex.dwICC  = ICC_COOL_CLASSES|ICC_BAR_CLASSES;
   InitCommonControlsEx(&icex);

   hwndRB = CreateWindowEx(WS_EX_TOOLWINDOW,
                           REBARCLASSNAME,
                           NULL,
                           WS_CHILD|WS_VISIBLE|WS_CLIPSIBLINGS|
                           WS_CLIPCHILDREN|RBS_VARHEIGHT|CCS_NODIVIDER,
                           0,0,0,0,
                           hwndOwner,
                           NULL,
                           g_hinst,
                           NULL);
   if(!hwndRB)
      return NULL;

   // Initialize and send the REBARINFO structure.
   rbi.cbSize = sizeof(REBARINFO);  // Required when using this struct.
   rbi.fMask  = 0;
   rbi.himl   = (HIMAGELIST)NULL;
   if(!SendMessage(hwndRB, RB_SETBARINFO, 0, (LPARAM)&rbi))
      return NULL;
```

(continued)

(continued)

```
    // Initialize structure members that both bands will share.
    rbBand.cbSize = sizeof(REBARBANDINFO);  // Required
    rbBand.fMask  = RBBIM_COLORS | RBBIM_TEXT | RBBIM_BACKGROUND |
                    RBBIM_STYLE | RBBIM_CHILD  | RBBIM_CHILDSIZE |
                    RBBIM_SIZE;
    rbBand.fStyle = RBBS_CHILDEDGE | RBBS_FIXEDBMP;
    rbBand.hbmBack= LoadBitmap(g_hinst, MAKEINTRESOURCE(IDB_BACKGRND));

    // Create the combo-box control to be added.
    hwndCB = CreateComboBox(hwndRB);

    // Set values unique to the band with the combo box.
    GetWindowRect(hwndCB, &rc);
    rbBand.lpText     = "Combo Box";
    rbBand.hwndChild  = hwndCB;
    rbBand.cxMinChild = 0;
    rbBand.cyMinChild = rc.bottom - rc.top;
    rbBand.cx         = 200;

    // Add the band that has the combo box.
    SendMessage(hwndRB, RB_INSERTBAND, (WPARAM)-1, (LPARAM)&rbBand);

    // Create the tool bar control to be added.
    hwndTB = CreateToolbar(hwndOwner, dwStyle);

    // Get the height of the tool bar.
    dwBtnSize = SendMessage(hwndTB, TB_GETBUTTONSIZE, 0,0);

    // Set values unique to the band with the tool bar.
    rbBand.lpText     = "Tool Bar";
    rbBand.hwndChild  = hwndTB;
    rbBand.cxMinChild = 0;
    rbBand.cyMinChild = HIWORD(dwBtnSize);
    rbBand.cx         = 250;

    // Add the band that has the tool bar.
    SendMessage(hwndRB, RB_INSERTBAND, (WPARAM)-1, (LPARAM)&rbBand);

    return (hwndRB);
}
```

Rebar Control Styles

Rebar controls support a variety of control styles in addition to standard window styles.

RBS_AUTOSIZE — **Version 4.71.** The rebar control automatically will change the layout of the bands when the size or position of the control changes. An **RBN_AUTOSIZE** notification will be sent when this occurs.

RBS_BANDBORDERS — **Version 4.70.** The rebar control displays narrow lines to separate adjacent bands.

RBS_DBLCLKTOGGLE — **Version 4.71.** The rebar band will toggle its maximized or minimized state when the user double-clicks on the band. Without this style, the maximized or minimized state is toggled when the user single-clicks on the band.

RBS_FIXEDORDER — **Version 4.70.** The rebar control always displays bands in the same order. You can move bands to different rows, but the band order is static.

RBS_REGISTERDROP — **Version 4.71.** The rebar control generates **RBN_GETOBJECT** notification messages when an object is dragged over a band in the control. To receive the **RBN_GETOBJECT** notifications, initialize OLE with a call to **OleInitialize** or **CoInitialize**.

RBS_TOOLTIPS — **Version 4.70.** Not yet supported.

RBS_VARHEIGHT — **Version 4.70.** The rebar control displays bands at the minimum required height, when possible. Without this style, the rebar control displays all bands at the same height, using the height of the tallest visible band to determine the height of other bands.

RBS_VERTICALGRIPPER — **Version 4.71.** The size grip will be displayed vertically instead of horizontally in a vertical rebar control. This style is ignored for rebar controls that do not have the **CCS_VERT** style.

Rebar Control Reference

Rebar Control Messages

RB_BEGINDRAG

Puts the rebar control in drag-and-drop mode. This message does not cause a **RBN_BEGINDRAG** notification to be sent.

```
RB_BEGINDRAG
    wParam = (WPARAM)(UINT) uBand;
    lParam = (LPARAM)(DWORD)dwPos;
```

Parameters

uBand

Zero-based index of the band that the drag-and-drop operation will affect.

dwPos

DWORD value that contains the starting mouse coordinates. The horizontal coordinate is contained in the LOWORD and the vertical coordinate is contained in the HIWORD. If you pass (DWORD)–1, the rebar control will use the position of the mouse the last time the control's thread called **GetMessage** or **PeekMessage**.

Return Values

The return value for this message is not used.

Remarks

The **RB_BEGINDRAG**, **RB_DRAGMOVE**, and **RB_ENDDRAG** messages allow you to implement an **IDropTarget** interface for a rebar control. You send the **RB_BEGINDRAG** message in response to **IDropTarget::DragEnter**, send the **RB_DRAGMOVE** message in response to **IDropTarget::DragOver**, and the **RB_ENDDRAG** message in response to **IDropTarget::Drop** and **IDropTarget::DragLeave**.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

RB_DELETEBAND

Deletes a band from a rebar control.

```
RB_DELETEBAND
    wParam = (WPARAM)(UINT) uBand;
    lParam = 0;
```

Parameters

uBand

Zero-based index of the band to be deleted.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RB_DRAGMOVE

Updates the drag position in the rebar control after a previous **RB_BEGINDRAG** message.

```
RB_DRAGMOVE
    wParam = 0;
    lParam = (LPARAM)(DWORD)dwPos;
```

Parameters

dwPos

DWORD value that contains the new mouse coordinates. The horizontal coordinate is contained in the LOWORD and the vertical coordinate is contained in the HIWORD. If you pass (DWORD)–1, the rebar control will use the position of the mouse the last time the control's thread called **GetMessage** or **PeekMessage**.

Return Values

The return value for this message is not used.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

See Also

RB_ENDDRAG

RB_ENDDRAG

Terminates the rebar control's drag-and-drop operation. This message does not cause an **RBN_ENDDRAG** notification to be sent.

```
RB_ENDDRAG
    wParam = 0;
    lParam = 0;
```

Return Values

The return value for this message is not used.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

See Also

RB_BEGINDRAG, **RB_DRAGMOVE**

RB_GETBANDBORDERS

Retrieves the borders of a band. The result of this message can be used to calculate the usable area in a band.

```
RB_GETBANDBORDERS
    wParam = (WPARAM)(UINT) uBand;
    lParam = (LPARAM)(LPRECT) lprc;
```

Parameters

uBand

Zero-based index of the band for which the borders will be retrieved.

lprc

Address of a **RECT** structure that will receive the band borders. If the rebar control has the **RBS_BANDBORDERS** style, each member of this structure will receive the number of pixels, on the corresponding side of the band, that constitute the border. If the rebar control does not have the **RBS_BANDBORDERS** style, only the **left** member of this structure receives valid information.

Return Values

The return value is not used.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RB_GETBANDCOUNT

Retrieves the count of bands currently in the rebar control.

```
RB_GETBANDCOUNT
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a UINT value that represents the number of bands assigned to the control.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RB_GETBANDINFO

Retrieves information about a specified band in a rebar control.

```
RB_GETBANDINFO
    wParam = (WPARAM)(UINT) uBand;
    lParam = (LPARAM)(LPREBARBANDINFO) lprbbi;
```

Parameters

uBand
Zero-based index of the band for which the information will be retrieved.

lprbbi
Address of a **REBARBANDINFO** structure that will receive the requested band information. Before sending this message, you must set the **cbSize** member of this structure to the size of the **REBARBANDINFO** structure and set the **fMask** parameter to the items you want to retrieve. Additionally, you must set the **cch** member of the **REBARBANDINFO** structure to the size of the **lpText** buffer when **RBBIM_TEXT** is specified.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

See Also

RB_SETBANDINFO

RB_GETBARHEIGHT

Retrieves the height of the rebar control.

```
RB_GETBARHEIGHT
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a UINT value that represents the height, in pixels, of the control.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RB_GETBARINFO

Retrieves information about the rebar control and the image list it uses.

```
RB_GETBARINFO
    wParam = 0;
    lParam = (LPARAM)(LPREBARINFO) lprbi;
```

Parameters

lprbi
Address of a **REBARINFO** structure that will receive the rebar control information. You must set the **cbSize** member of this structure to sizeof(REBARINFO) before sending this message.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RB_GETBKCOLOR

Retrieves a rebar control's default background color.

```
RB_GETBKCOLOR
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a **COLORREF** value that represent the current default background color.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

See Also

RB_SETBKCOLOR

RB_GETCOLORSCHEME

Retrieves the color scheme information from the rebar control.

```
RB_GETCOLORSCHEME
    wParam = 0;
    lParam = (LPARAM)(LPCOLORSCHEME) lpcs;
```

Parameters

lpcs
Address of a **COLORSCHEME** structure that will receive the color scheme information. You must set the **dwSize** member of this structure to **sizeof(COLORSCHEME)** before sending this message.

Return Values

Returns nonzero if successful, or zero otherwise.

Version 4.71 and later of Comctl32.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 commctrl.h.

RB_SETCOLORSCHEME

RB_GETDROPTARGET

Retrieves a rebar control's **IDropTarget** interface pointer.

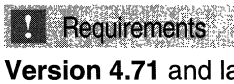
```
RB_GETDROPTARGET
    wParam = 0;
    lParam = (IDropTarget**)ppDropTarget;
```

Parameters

ppDropTarget
Address of an **IDropTarget** pointer that receives the interface pointer. It is the caller's responsibility to call **Release** on this pointer when it is no longer needed.

Return Values

The return value for this message is not used.



Version 4.71 and later of Comctl32.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 commctrl.h.

RB_GETPALETTE

Retrieves the rebar control's current palette.

```
RB_GETPALETTE
    wParam = 0;
    lParam = 0;
```

Return Values

Returns an **HPALETTE** that specifies the rebar control's current palette.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

See Also

RB_SETPALETTE

RB_GETRECT

Retrieves the bounding rectangle for a given band in a rebar control.

```
RB_GETRECT
    wParam = (WPARAM)(INT) iBand;
    lParam = (LPARAM)(LPRECT) lprc;
```

Parameters

iBand
Zero-based index of a band in the rebar control.

lprc
Address of a **RECT** structure that will receive the bounds of the rebar band.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RB_GETROWCOUNT

Retrieves the number of rows of bands in a rebar control.

```
RB_GETROWCOUNT
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a UINT value that represents the number of band rows in the control.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RB_GETROWHEIGHT

Retrieves the height of a specified row in a rebar control.

```
RB_GETROWHEIGHT
    wParam = (WPARAM)(UINT) uRow;
    lParam = 0;
```

Parameters

uRow
Zero-based index of a band. The height of the row that contains the specified band will be retrieved.

Return Values

Returns a UINT value that represents the row height, in pixels.

Remarks

To retrieve the number of rows in a rebar control, use the **RB_GETROWCOUNT** message.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RB_GETTEXTCOLOR

Retrieves a rebar control's default text color.

```
RB_GETTEXTCOLOR
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a **COLORREF** value that represent the current default text color.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

See Also

RB_SETTEXTCOLOR

RB_GETTOOLTIPS

Retrieves the handle to any tooltip control associated with the rebar control.

```
RB_GETTOOLTIPS
    wParam = 0;
    lParam = 0;
```

Return Values

Returns an HWND value that is the handle to the tooltip control associated with the rebar control, or zero if no tooltip control is associated with the rebar control.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

RB_GETUNICODEFORMAT

Retrieves the UNICODE character format flag for the control.

```
RB_GETUNICODEFORMAT
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the UNICODE format flag for the control. If this value is nonzero, the control is using UNICODE characters. If this value is zero, the control is using ANSI characters.

Remarks

See the remarks for **CCM_GETUNICODEFORMAT** for a discussion of this message.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

See Also

RB_SETUNICODEFORMAT

RB_HITTEST

Determines which portion of a rebar band is at a given point on the screen, if a rebar band exists at that point.

```
RB_HITTEST
    wParam = 0;
    lParam = (LPARAM)(LPRBHITTESTINFO) lprbht;
```

Parameters

lprbht

Address of an **RBHITTESTINFO** structure. Before sending the message, the **pt** member of this structure must be initialized to the point that will be hit tested, in client coordinates.

Return Values

Returns the zero-based index of the band at the given point, or –1 if no rebar band was at the point.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RB_IDTOINDEX

Converts a band identifier to a band index in a rebar control.

```
RB_IDTOINDEX
    wParam = (WPARAM)(UINT) uBandID;
    lParam = 0;
```

Parameters

uBandID

The application-defined identifier of the band in question. This is the value that was passed in the **wID** member of the **REBARBANDINFO** structure when the band was inserted.

Return Values

Returns the zero-based band index if successful, or –1 otherwise. If duplicate band identifiers exist, the first one is returned.

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RB_INSERTBAND

Inserts a new band in a rebar control.

```
RB_INSERTBAND
    wParam = (WPARAM)(UINT) uIndex;
    lParam = (LPARAM)(LPREBARBANDINFO) lprbbi;
```

Parameters

uIndex

Zero-based index of the location where the band will be inserted. If you set this parameter to –1, the control will add the new band at the last location.

lprbbi

Address of a **REBARBANDINFO** structure that defines the band to be inserted. You must set the **cbSize** member of this structure to sizeof(REBARBANDINFO) before sending this message.

Return Values

Returns nonzero if successful, or zero otherwise.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RB_MAXIMIZEBAND

Resizes a band in a rebar control to either its ideal or largest size.

```
RB_MAXIMIZEBAND
    wParam = (WPARAM)(UINT) uBand;
    lParam = (LPARAM)(BOOL) fIdeal;
```

Parameters

uBand
Zero-based index of the band to be maximized.

fIdeal
Indicates if the ideal width of the band should be used when the band is maximized. If this value is nonzero, the ideal width will be used. If this value is zero, the band will be made as large as possible.

Return Values

The return value is not used.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RB_MINIMIZEBAND

Resizes a band in a rebar control to its smallest size.

```
RB_MINIMIZEBAND
    wParam = (WPARAM)(UINT) uBand;
    lParam = 0;
```

Parameters

uBand
Zero-based index of the band to be minimized.

Return Values

The return value is not used.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RB_MOVEBAND

Moves a band from one index to another.

```
RB_MOVEBAND
    wParam = (WPARAM)(UINT) iFrom;
    lParam = (LPARAM)(UINT) iTo;
```

Parameters

iFrom
Zero-based index of the band to be moved.

iTo
Zero-based index of the new band position.

Return Values

Returns nonzero if successful, or zero otherwise.

Remarks

This message most likely will change the index of other bands in the rebar control. If a band is moved from index 6 to index 0, all of the bands in between will have their index incremented by one.

iTo must never be greater than the number of bands minus one. The number of bands can be obtained with the **RB_GETBANDCOUNT** message.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

RB_PUSHCHEVRON

Sent to a rebar control to programmatically push a chevron.

```
RB_PUSHCHEVRON
    wParam = (WPARAM)(UINT) uBand;
    lParam = (LPARAM) lAppValue;
```

Parameters

uBand
Zero-based index of the band whose chevron is to be pushed.

lAppValue
An application-defined, 32-bit value. It will be passed back to the application as the **lParamNM** member of the **NMREBARCHEVRON** structure that is passed with the **RBN_CHEVRONPUSHED** notification.

Return Values

The return value for this notification is not used.

Remarks

When this message is sent, the rebar control will send the application an **RBN_CHEVRONPUSHED** notification, prompting it to display the chevron menu.

Requirements

Version 5.80 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 or Windows NT 4 with Internet Explorer 5 or later.
Windows 95/98: Requires Windows 98 or Windows 95 with Internet Explorer 5 or later.
Windows CE: Unsupported.
Header: Declared in commctrl.h.

RB_SETBANDINFO

Sets characteristics of an existing band in a rebar control.

```
RB_SETBANDINFO
    wParam = (WPARAM)(UINT) uBand;
    lParam = (LPARAM)(LPREBARBANDINFO) lprbbi;
```

Parameters

uBand
Zero-based index of the band to receive the new settings.

lprbbi
Address of a **REBARBANDINFO** structure that defines the band to be modified and the new settings. Before sending this message, you must set the **cbSize** member of this structure to the sizeof(REBARBANDINFO) structure. Additionally, you must set the **cch** member of the **REBARBANDINFO** structure to the size of the **lpText** buffer when **RBBIM_TEXT** is specified.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RB_SETBARINFO

Sets the characteristics of a rebar control.

```
RB_SETBARINFO
    wParam = 0;
    lParam = (LPARAM)(LPREBARINFO) lprbi;
```

Parameters

lprbi
Address of a **REBARINFO** structure that contains the information to be set. You must set the **cbSize** member of this structure to sizeof(REBARINFO) before sending this message.

Return Values

Returns nonzero if successful, or zero otherwise.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RB_SETBKCOLOR

Sets a rebar control's default background color.

```
RB_SETBKCOLOR
    wParam = 0;
    lParam = (LPARAM)(COLORREF)clrBk;
```

Parameters

clrBk

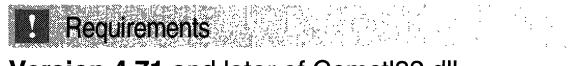
COLORREF value that represents the new default background color.

Return Values

Returns a **COLORREF** value that represents the previous default background color.

Remarks

The rebar control's default background color is used to draw the background in a rebar control and all bands that are added after this message has been sent. The default background color for a particular band can be overridden when a band is added or modified by setting the **RBBIM_COLORS** flag in **fMask** and setting **clrBack** in the **REBARBANDINFO** structure.



Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

See Also

RB_GETBKCOLOR

RB_SETCOLORSCHEME

Sets the color scheme information for the rebar control.

```
RB_SETCOLORSCHEME
    wParam = 0;
    lParam = (LPARAM)(LPCOLORSCHEME) lpcs;
```

Parameters

lpcs
Address of a **COLORSCHEME** structure that contains the color scheme information.

Return Values

The return value for this message is not used.

Remarks

The rebar control uses the color scheme information when drawing the three-dimensional elements in the control and bands.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

See Also

RB_GETCOLORSCHEME

RB_SETPALETTE

Sets the rebar control's current palette.

```
RB_SETPALETTE
    wParam = 0;
    lParam = (LPARAM)(HPALETTE) hpal;
```

Parameters

hpal

An **HPALETTE** that specifies the new palette that the rebar control will use.

Return Values

Returns an **HPALETTE** that specifies the rebar control's previous palette.

Remarks

It is the responsibility of the application sending this message to delete the HPALETTE passed in this message (see **DeleteObject**). The rebar control will not delete an HPALETTE set with this message.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

See Also

RB_GETPALETTE

RB_SETPARENT

Sets a rebar control's parent window.

```
RB_SETPARENT
    wParam = (WPARAM)(HWND) hwndParent;
    lParam = 0;
```

Parameters

hwndParent

Handle to the parent window to be set.

Return Values

Returns the handle to the previous parent window, or NULL if there is no previous parent.

Remarks

The rebar control sends notification messages to the window you specify with this message. This message does not actually change the parent of the rebar control.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RB_SETTEXTCOLOR

Sets a rebar control's default text color.

```
RB_SETTEXTCOLOR
    wParam = 0;
    lParam = (LPARAM)(COLORREF)clrText;
```

Parameters

clrText
COLORREF value that represents the new default text color.

Return Values

Returns a **COLORREF** value that represents the previous default text color.

Remarks

The rebar control's default text color is used to draw the text in a rebar control and all bands that are added after this message has been sent. The default text color for a particular band can be overridden when a band is added or modified by setting the **RBBIM_COLORS** flag in **fMask** and setting **clrBack** in the **REBARBANDINFO** structure.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

See Also

RB_GETTEXTCOLOR

RB_SETTOOLTIPS

Associates a tooltip control with the rebar control.

```
RB_SETTOOLTIPS
    wParam = (WPARAM)(HWND) hwndToolTip;
    lParam = 0;
```

Parameters

hwndToolTip
Handle to the tooltip control to be set.

Return Values

The return value for this message is not used.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

RB_SETUNICODEFORMAT

Sets the UNICODE character format flag for the control. This message allows you to change the character set used by the control at run time, instead of having to re-create the control.

```
RB_SETUNICODEFORMAT
    wParam = (WPARAM)(BOOL)fUnicode;
    lParam = 0;
```

Parameters

fUnicode

Determines the character set that is used by the control. If this value is nonzero, the control will use UNICODE characters. If this value is zero, the control will use ANSI characters.

Return Values

Returns the previous UNICODE format flag for the control.

Remarks

See the remarks for **CCM_SETUNICODEFORMAT** for a discussion of this message.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

See Also

RB_GETUNICODEFORMAT

RB_SHOWBAND

Shows or hides a given band in a rebar control.

```
RB_SHOWBAND
    wParam = (WPARAM)(INT) iBand;
    lParam = (LPARAM)(BOOL) fShow;
```

Parameters

iBand

Zero-based index of a band in the rebar control.

fShow

Boolean value that indicates if the band should be shown or hidden. If this value is nonzero, the band will be shown. Otherwise, the band will be hidden.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RB_SIZETORECT

Attempts to find the best layout of the bands for the given rectangle.

```
RB_SIZETORECT
   wParam = 0;
   lParam = (LPARAM)(LPRECT) prc;
```

Parameters

prc

Address of a **RECT** structure that specifies the rectangle to which the rebar control should be sized.

Return Values

Returns nonzero if a layout change occurred, or zero otherwise.

Remarks

The rebar bands will be arranged and wrapped as necessary to fit the rectangle.

Bands that have the **RBBS_VARIABLEHEIGHT** style will be resized as evenly as possible to fit the rectangle.

The height of a horizontal rebar or the width of a vertical rebar can change, depending on the new layout.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

Rebar Control Notifications

NM_CUSTOMDRAW (rebar)

Sent by the rebar control to notify its parent window about drawing operations. This notification is sent in the form of a **WM_NOTIFY** message.

```
NM_CUSTOMDRAW
    lpNMCustomDraw = (LPNMCUSTOMDRAW) lParam;
```

Parameters

lpNMCustomDraw
Address of an **NMCUSTOMDRAW** structure that contains information about the drawing operation. The **dwItemSpec** member of this structure contains the identifier of the band being drawn. The **lItemlParam** member of this structure contains the lParam of the band being drawn.

Return Values

The value your application can return depends on the current drawing stage. The **dwDrawStage** member of the associated **NMCUSTOMDRAW** structure holds a value that specifies the drawing stage. You must return one of the following values.

When **dwDrawStage** equals CDDS_PREPAINT:

CDRF_DODEFAULT
The control will draw itself. It will not send any additional **NM_CUSTOMDRAW** messages for this paint cycle.

CDRF_NOTIFYITEMDRAW
The control will notify the parent of any item-related drawing operations. It will send **NM_CUSTOMDRAW** notification messages before and after drawing items.

CDRF_NOTIFYITEMERASE
The control will notify the parent when an item will be erased. It will send **NM_CUSTOMDRAW** notification messages before and after erasing items.

CDRF_NOTIFYPOSTERASE
The control will notify the parent after erasing an item.

CDRF_NOTIFYPOSTPAINT
The control will notify the parent after painting an item.

CDRF_NOTIFYSUBITEMDRAW
Version 4.71. The control will notify the parent when a list-view subitem is being drawn.

When **dwDrawStage** equals CDDS_ITEMPREPAINT:

CDRF_NEWFONT
Your application specified a new font for the item; the control will use the new font. For more information on changing fonts, see *Changing Fonts and Colors*.

CDRF_SKIPDEFAULT
Your application drew the item manually. The control will not draw the item.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 4.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

See Also

Using Custom Draw

NM_NCHITTEST (rebar)

Sent by a rebar control when the control receives a **WM_NCHITTEST** message. This notification message is sent in the form of a **WM_NOTIFY** message.

```
NM_NCHITTEST
    lpnmmouse = (LPNMMOUSE) lParam;
```

Parameters

lpnmmouse
Address of an **NMMOUSE** structure that contains information about the notification. The *dwItemSpec* member contains the band index over which the hit-test message occurred, and the *pt* member contains the mouse coordinates of the hit-test message.

Return Values

Return zero to allow the rebar to perform default processing of the hit-test message, or return one of the HT* values documented under **WM_NCHITTEST** to override the default hit-test processing.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

NM_RELEASEDCAPTURE (rebar)

Notifies a rebar control's parent window that the control is releasing mouse capture. This notification is sent in the form of a **WM_NOTIFY** message.

```
NM_RELEASEDCAPTURE
    lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh
Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The control ignores the return value from this notification.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

RBN_AUTOSIZE

Sent by a rebar control created with the **RBS_AUTOSIZE** style when the rebar automatically resizes itself. This notification message is sent in the form of a **WM_NOTIFY** message.

```
RBN_AUTOSIZE
    lpnmas = (LPNMRBAUTOSIZE) lParam;
```

Parameters

lpnmas

Address of an **NMRBAUTOSIZE** structure that contains information about the resize operation.

Return Values

The return value for this notification is not used.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RBN_BEGINDRAG

Sent by a rebar control when the user begins dragging a band. This notification message is sent in the form of a **WM_NOTIFY** message.

```
RBN_BEGINDRAG
    lpnmrb = (LPNMREBAR)lParam;
```

Parameters

lpnmrb

Address of an **NMREBAR** structure that contains information about the notification.

Return Values

Return zero to allow the rebar to continue the drag operation, or nonzero to quit the drag operation.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RBN_CHEVRONPUSHED

Sent by a rebar control when a chevron is pushed. This notification message is sent in the form of a **WM_NOTIFY** message.

```
RBN_CHEVRONPUSHED
    lpnm = (NMREBARCHEVRON) lParam;
```

Parameters

lpnm
Pointer to the the band's **NMREBARCHEVRON** structure.

Return Values

The return value for this notification is not used.

Remarks

When an application receives this notification, it is responsible for displaying a popup menu with items for each hidden tool. Use the **rc** member of the **NMREBARCHEVRON** structure to position the menu.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.

RBN_CHILDSIZE

Sent by a rebar control when a band's child window is resized. This notification message is sent in the form of a **WM_NOTIFY** message.

```
RBN_CHILDSIZE
    lprbcs = (LPNMREBARCHILDSIZE)lParam;
```

Parameters

lprbcs

Address of an **NMREBARCHILDSIZE** structure that contains information about the notification.

Return Values

The return value for this notification is not used.

Version 4.71 and later of Comctl32.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 commctrl.h.

RBN_DELETEDBAND

Sent by a rebar control after a band has been deleted. This notification message is sent in the form of a **WM_NOTIFY** message.

```
RBN_DELETEDBAND
    lpnmrb = (LPNMREBAR)lParam;
```

Parameters

lpnmrb

Address of an **NMREBAR** structure that contains information about the notification.

Return Values

The return value for this notification is not used.

Version 4.71 and later of Comctl32.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 commctrl.h.

RBN_DELETINGBAND

Sent by a rebar control when a band is about to be deleted. This notification message is sent in the form of a **WM_NOTIFY** message.

```
RBN_DELETINGBAND
    lpnmrb = (LPNMREBAR)lParam;
```

Parameters

lpnmrb
Address of an **NMREBAR** structure that contains information about the notification.

Return Values

The return value for this notification is not used.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

RBN_ENDDRAG

Sent by a rebar control when the user stops dragging a band. This notification message is sent in the form of a **WM_NOTIFY** message.

```
RBN_ENDDRAG
    lpnmrb = (LPNMREBAR)lParam;
```

Parameters

lpnmrb
Address of an **NMREBAR** structure that contains information about the notification.

Return Values

The return value for this notification is not used.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RBN_GETOBJECT

Sent by a rebar control created with the **RBS_REGISTERDROP** style when an object is dragged over a band in the control. This notification message is sent in the form of a **WM_NOTIFY** message.

```
RBN_GETOBJECT
    lpnmon = (LPNMOBJECTNOTIFY) lParam;
```

Parameters

lpnmon
Address of an **NMOBJECTNOTIFY** structure that contains information about the band that the object is dragged over and also receives the data provided by the receiving application in response to this message.

Return Values

The return value for this notification must be zero.

Remarks

To receive the RBN_GETOBJECT notification, initialize OLE with a call to **OleInitialize** or **CoInitialize**.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

RBN_HEIGHTCHANGE

Sent by a rebar control when its height has changed. This notification message is sent in the form of a **WM_NOTIFY** message.

```
RBN_HEIGHTCHANGE
    lpnmhdr = (LPNMHDR) lParam;
```

Parameters

lpnmhdr

Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The return value for this notification is not used.

Remarks

Rebar controls that use the CCS_VERT style send this notification message when their width changes.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

RBN_LAYOUTCHANGED

Sent by a rebar control when the user changes the layout of the control's bands. This notification message is sent in the form of a **WM_NOTIFY** message.

```
RBN_LAYOUTCHANGED
    lpnmhdr = (LPNMHDR) lParam;
```

Parameters

lpnmhdr

Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The return value for this notification is not used.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

Rebar Control Structures

NMRBAUTOSIZE

Contains information used in handling the **RBN_AUTOSIZE** notification messages.

```
typedef struct tagNMRBAUTOSIZE {
    NMHDR hdr;
    BOOL fChanged;
    RECT rcTarget;
    RECT rcActual;
} NMRBAUTOSIZE, *LPNMRBAUTOSIZE;
```

Members

hdr
NMHDR structure that contains additional information about the notification message.

fChanged
Member that indicates if the size or layout of the rebar control has changed (nonzero if a change occurred, or zero otherwise).

rcTarget
RECT structure that contains the rectangle to which the rebar control tried to size itself.

rcActual
RECT structure that contains the rectangle to which the rebar control actually sized itself.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

NMREBAR

Contains information used in handling various rebar notification messages.

```
typedef struct tagNMREBAR {
    NMHDR hdr;
    DWORD dwMask;
    UINT uBand;
    UINT fStyle;
    UINT wID;
    LPARAM lParam;
} NMREBAR, *LPNMREBAR;
```

Members

hdr
NMHDR structure that contains additional information about the notification message.

dwMask
Set of flags that define which members of this structure contain valid information. This can be one or more of the following values:

RBNM_ID	The **wID** member contains valid information.
RBNM_LPARAM	The **lParam** member contains valid information.
RBNM_STYLE	The **fStyle** member contains valid information.

uBand
Zero-based index of the band affected by the notification. This will be –1 if no band is affected.

fStyle
The style of the band. This is one or more of the RBBS_ styles detailed in the **fStyle** member of the **REBARBANDINFO** structure. This member is valid only if **dwMask** contains RBNM_STYLE.

wID
Application-defined identifier of the band. This member is only valid if **dwMask** contains RBNM_ID.

lParam
Application-defined, 32-bit value associated with the band. This member is valid only if **dwMask** contains RBNM_LPARAM.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

NMREBARCHEVRON

Contains information used in handling the **RBN_CHEVRONPUSHED** notification message.

```
typedef struct tagNMREBARCHEVRON{
    NMHDR hdr;
    UINT uBand;
    UINT wID;
    LPARAM lParam;
    RECT rc;
    LPARAM lParamNM;
} NMREBARCHEVRON, *LPNMREBARCHEVRON;
```

Members

hdr
NMHDR structure that contains additional information about the notification message.

uBand
Index of the band sending the notification.

wID
Application-defined identifier for the band.

lParam
Application-defined, 32-bit value associated with the band.

rc
RECT structure that defines the area covered by the chevron.

lParamNM
Application-defined, 32-bit value. If the **RBN_CHEVRONPUSHED** notification was sent as a result of an **RB_PUSHCHEVRON** message, this member will contain the message's *lAppValue* value. Otherwise, it will be set to zero.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.

NMREBARCHILDSIZE

Contains information used in handling the **RBN_CHILDSIZE** notification message.

```
typedef struct tagNMREBARCHILDSIZE{
    NMHDR hdr;
    UINT uBand;
    UINT wID;
    RECT rcChild;
    RECT rcBand;
} NMREBARCHILDSIZE, *LPNMREBARCHILDSIZE;
```

Members

hdr
NMHDR structure that contains additional information about the notification message.

uBand
Zero-based index of the band affected by the notification. This will be –1 if no band is affected.

wID
Application-defined identifier of the band.

rcChild
RECT structure that contains the new size of the child window. This member can be changed during the notification to modify the child window's position and size.

rcBand
RECT structure that contains the new size of the band.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

RBHITTESTINFO

Contains information specific to a hit test operation. This structure is used with the **RB_HITTEST** message.

```
typedef struct _RB_HITTESTINFO {
    POINT pt;
    UINT  flags;
    int   iBand;
} RBHITTESTINFO, FAR *LPRBHITTESTINFO;
```

Members

pt

POINT structure that describes the point to be hit-tested, in client coordinates.

flags

Member that receives a flag value indicating the rebar band's component located at the point described by **pt**. This member will be one of the following:

RBHT_CAPTION	The point was in the rebar band's caption.
RBHT_CHEVRON	The point was in the rebar band's chevron (versions 5.80 and greater).
RBHT_CLIENT	The point was in the rebar band's client area.
RBHT_GRABBER	The point was in the rebar band's gripper.
RBHT_NOWHERE	The point was not in a rebar band.

iBand

Member that receives the rebar band's index at the point described by **pt**. This value will be the zero-based index of the band, or –1 if no band was at the hit-tested point.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

REBARBANDINFO

Contains information that defines a band in a rebar control.

```
typedef struct tagREBARBANDINFO{
    UINT        cbSize;
```

```
    UINT        fMask;
    UINT        fStyle;
    COLORREF    clrFore;
    COLORREF    clrBack;
    LPTSTR      lpText;
    UINT        cch;
    int         iImage;
    HWND        hwndChild;
    UINT        cxMinChild;
    UINT        cyMinChild;
    UINT        cx;
    HBITMAP     hbmBack;
    UINT        wID;
#if (_WIN32_IE >= 0x0400)
    UINT        cyChild;
    UINT        cyMaxChild;
    UINT        cyIntegral;
    UINT        cxIdeal;
    LPARAM      lParam;
    UINT        cxHeader;
#endif
} REBARBANDINFO, FAR *LPREBARBANDINFO;
```

Members

cbSize

Size of this structure, in bytes. Your application must fill this member before sending any messages that use the address of this structure as a parameter.

fMask

Flags that indicate which members of this structure are valid or must be filled. This value can be a combination of the following:

Flag	Description
RBBIM_BACKGROUND	The **hbmBack** member is valid or must be filled.
RBBIM_CHILD	The **hwndChild** member is valid or must be filled.
RBBIM_CHILDSIZE	The **cxMinChild**, **cyMinChild**, **cyChild**, **cyMaxChild**, and cyIntegral members are valid or must be filled.
RBBIM_COLORS	The **clrFore** and **clrBack** members are valid or must be filled.
RBBIM_HEADERSIZE	**Version 4.71**. The **cxHeader** member is valid or must be filled.
RBBIM_IDEALSIZE	**Version 4.71**. The **cxIdeal** member is valid or must be filled.
RBBIM_ID	The **wID** member is valid or must be filled.

(continued)

(continued)

Flag	Description
RBBIM_IMAGE	The **iImage** member is valid or must be filled.
RBBIM_LPARAM	**Version 4.71**. The **lParam** member is valid or must be filled.
RBBIM_SIZE	The **cx** member is valid or must be filled.
RBBIM_STYLE	The **fStyle** member is valid or must be filled.
RBBIM_TEXT	The **lpText** member is valid or must be filled.

fStyle
Flags that specify the band style. This value can be a combination of the following:

Flag	Description
RBBS_BREAK	The band is on a new line.
RBBS_CHILDEDGE	The band has an edge at the top and bottom of the child window.
RBBS_FIXEDBMP	The background bitmap does not move when the band is resized.
RBBS_FIXEDSIZE	The band cannot be sized. With this style, the sizing grip is not displayed on the band.
RBBS_GRIPPERALWAYS	**Version 4.71**. The band will always have a sizing grip, even if it is the only band in the rebar.
RBBS_HIDDEN	The band will not be visible.
RBBS_NOGRIPPER	**Version 4.71**. The band will never have a sizing grip, even if there is more than one band in the rebar.
RBBS_USECHEVRON	**Version 5.80.** Show a chevron button if the band is smaller than **cxIdeal**.
RBBS_VARIABLEHEIGHT	**Version 4.71**. The band can be resized by the rebar control; **cyIntegral** and **cyMaxChild** affect how the rebar will resize the band.

clrFore and **clrBack**
Band foreground and background colors. If **hbmBack** specifies a background bitmap, these members are ignored. By default, the band will use the background color of the rebar control set with the **RB_SETBKCOLOR** message. If a background color is specified here, then this background color will be used, instead.

lpText
Address of a buffer that contains the display text for the band. If band information is being requested from the control and **RBBIM_TEXT** is specified in **fMask**, this member must be initialized to the address of the buffer that will receive the text.

cch
Size of the buffer at **lpText**, in bytes. If information is not being requested from the control, this member is ignored.

iImage
Zero-based index of any image that should be displayed in the band. The image list is set using the **RB_SETBARINFO** message.

hwndChild
Handle to the child window contained in the band, if any.

cxMinChild
Minimum width of the child window, in pixels. The band cannot be sized smaller than this value.

cyMinChild
Minimum height of the child window, in pixels. The band cannot be sized smaller than this value.

cx
Length of the band, in pixels.

hbmBack
Handle to a bitmap that is used as the background for this band.

wID
UINT value that the control uses to identify this band for custom draw notification messages. This value may be used for additional purposes in the future.

cyChild
Version 4.71. Initial height of the band, in pixels. This member is ignored unless the **RBBS_VARIABLEHEIGHT** style is specified.

cyMaxChild
Version 4.71. Maximum height of the band, in pixels. This member is ignored unless the **RBBS_VARIABLEHEIGHT** style is specified.

cyIntegral
Version 4.71. Step value by which the band can grow or shrink, in pixels. If the band is resized, it will be resized in steps specified by this value. This member is ignored unless the **RBBS_VARIABLEHEIGHT** style is specified.

cxIdeal
Version 4.71. Ideal width of the band, in pixels. If the band is maximized to the ideal width (see **RB_MAXIMIZEBAND**), the rebar control will attempt to make the band this width.

lParam
Version 4.71. 32-bit, application-defined value.

cxHeader
Version 4.71. Size of the band's header, in pixels. The band header is the area between the edge of the band and the edge of the child window. This is the area where band text and images are displayed, if they are specified. If this value is

specified, it will override the normal header dimensions that the control calculates for the band.

Remarks

The **cxMinChild**, **cyMinChild**, and **cx** members provide information on dimensions relative to the orientation of the control. That is, for a horizontal rebar control, **cxMinChild** and **cx** are horizontal measurements, and **cyMinChild** is a vertical measurement. However, if the control uses the **CCS_VERT** style, **cxMinChild** and **cx** are vertical measurements, and **cyMinChild** is a horizontal measurement.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

REBARINFO

Contains information that describes rebar control characteristics.

```
typedef struct tagREBARINFO{
    UINT        cbSize;
    UINT        fMask;
    HIMAGELIST  himl;
} REBARINFO, FAR *LPREBARINFO;
```

Members

cbSize

Size of this structure, in bytes. Your application must fill this member before sending any messages that use the address of this structure as a parameter.

fMask

Flag values that describe characteristics of the rebar control. Currently, rebar controls support only one value:

RBIM_IMAGELIST	The **himl** member is valid or must be filled.

himl

Handle to an image list. The rebar control will use the specified image list to obtain images.

Requirements

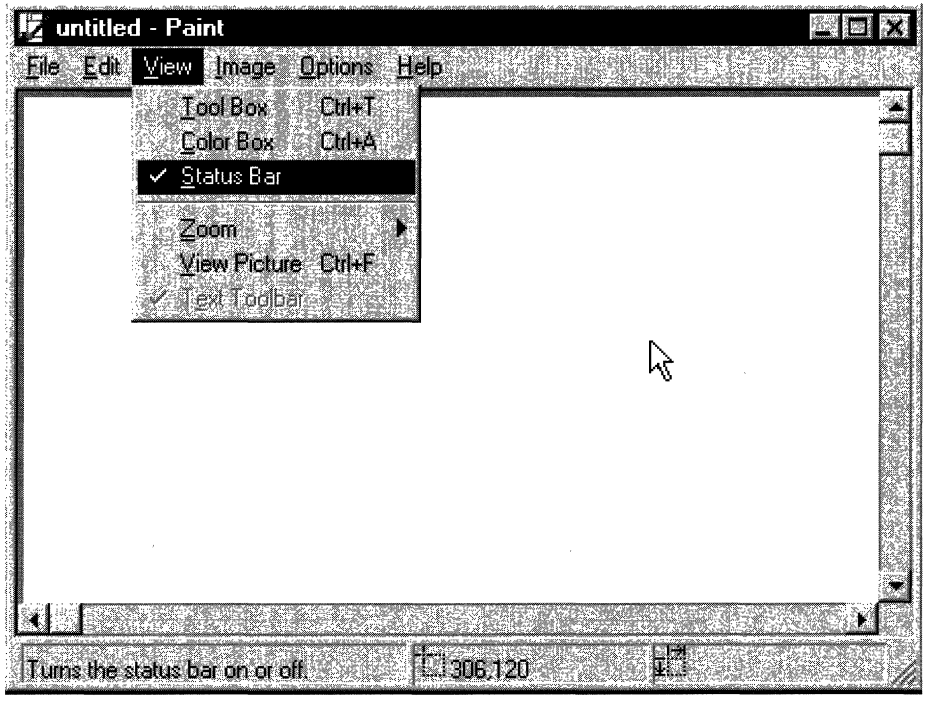
Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

CHAPTER 23

Status Bars

A *status bar* is a horizontal window at the bottom of a parent window in which an application can display various kinds of status information. The status bar can be divided into parts to display more than one type of information. The following illustration shows the status bar in the Microsoft Windows Paint application. The status bar is the bar at the bottom of the window that contains Help text and coordinate information.



Using Status Bars

You can create a status bar by using the **CreateStatusWindow** function or by using the **CreateWindowEx** function and specifying the STATUSCLASSNAME window class. To ensure that the common control dynamic-link library (DLL) is loaded, use the **InitCommonControls** function first. After you create the status bar, you can divide it into parts, set the text for each part, and control the appearance of the window by using status-bar messages.

Types and Styles

The default position of a status bar is along the bottom of the parent window, but you can specify the **CCS_TOP** style to have it appear at the top of the parent window's client area.

You can specify the **SBARS_SIZEGRIP** style to include a sizing grip at the right end of the status bar.

Note Combining the **CCS_TOP** and **SBARS_SIZEGRIP** styles is not recommended, because the resulting sizing grip is not functional.

Size and Height

The window procedure for the status bar automatically sets the initial size and position of the window, ignoring the values specified in the **CreateWindowEx** function. The width is the same as that of the parent window's client area. The height is based on the metrics of the font that is currently selected into the status bar's device context and on the width of the window's borders.

The window procedure automatically adjusts the size of the status bar whenever it receives a **WM_SIZE** message. Typically, when the size of the parent window changes, the parent sends a **WM_SIZE** message to the status bar.

An application can set the minimum height of a status bar's drawing area by sending the window an **SB_SETMINHEIGHT** message, specifying the minimum height, in pixels. The drawing area does not include the window's borders. A minimum height is useful for drawing in an owner-drawn status bar. For more information, see **Owner-Drawn Status Bars** later in this chapter.

You retrieve the widths of the borders of a status bar by sending the window an **SB_GETBORDERS** message. The message includes the address of a three-element array that receives the widths.

Multiple-Part Status Bars

A status bar can have many different parts, each displaying a different line of text. You divide a status bar into parts by sending the window an **SB_SETPARTS** message, specifying the number of parts to create and the address of an integer array. The array contains one element for each part, and each element specifies the client coordinate of the right edge of a part.

A status bar can have a maximum of 256 parts, although applications typically use far fewer than that. You retrieve a count of the parts in a status bar, as well as the coordinate of the right edge of each part, by sending the window an **SB_GETPARTS** message.

Status-Bar Text Operations

You set the text of any part of a status bar by sending the **SB_SETTEXT** message, specifying the zero-based index of a part, an address of the string to draw in the part, and the technique for drawing the string. The drawing technique determines whether the text has a border and, if it does, the style of the border. It also determines whether the parent window is responsible for drawing the text. For more information, see the *Owner-Drawn Status Bars* section below.

By default, text is left-aligned within the specified part of a status bar. You can embed tab characters (\ t) in the text to center or right-align it. Text to the right of a single tab character is centered, and text to the right of a second tab character is right-aligned.

To retrieve text from a status bar, use the **SB_GETTEXTLENGTH** and **SB_GETTEXT** messages.

If your application uses a status bar that has only one part, you can use the WM_SETTEXT, WM_GETTEXT, and WM_GETTEXTLENGTH messages to perform text operations. These messages deal only with the part that has an index of zero, allowing you to treat the status bar much like a static text control.

To display a line of status without creating a status bar, use the **DrawStatusText** function. The function uses the same techniques to draw the status as the window procedure for the status bar, but it does not automatically set the size and position of the status information. When calling the function, you must specify the size and position of the status information, as well as the device context of the window in which to draw it.

Owner-Drawn Status Bars

You can define individual parts of a status bar to be owner-drawn parts. Using this technique gives you more control than you would otherwise have over the appearance of the window part. For example, you can display a bitmap rather than text or draw text using a different font.

To define a window part as owner-drawn, send the **SB_SETTEXT** message to the status bar, specifying the part and the SBT_OWNERDRAW drawing technique. When SBT_OWNERDRAW is specified, the *lParam* parameter is a 32-bit, application-defined value that the application can use when drawing the part. For example, you can specify a font handle, a bitmap handle, an address of a string, and so on.

When a status bar needs to draw an owner-drawn part, it sends the **WM_DRAWITEM** message to the parent window. The *wParam* parameter of the message is the child window identifier of the status bar, and the *lParam* parameter is the address of a **DRAWITEMSTRUCT** structure. The parent window uses the information in the structure to draw the part. For an owner-drawn part of a status bar, **DRAWITEMSTRUCT** contains the following information:

Member	Description
CtlType	Undefined; do not use.
CtlID	Child window identifier of the status bar.
itemID	Zero-based index of the part to be drawn.
itemAction	Undefined; do not use.
itemState	Undefined; do not use.
hwndItem	Handle to the status bar.
hDC	Handle to the device context of the status bar.
rcItem	Coordinates of the window part to be drawn. The coordinates are relative to the upper-left corner of the status bar.
itemData	Application-defined 32-bit value specified in the *lParam* parameter of the **SB_SETTEXT** message.

Simple-Mode Status Bars

You put a status bar in "simple mode" by sending it an **SB_SIMPLE** message. A simple-mode status bar displays only one part. When the text of the window is set, the window is invalidated, but it is not redrawn until the next **WM_PAINT** message. Waiting for the message reduces screen flicker by minimizing the number of times the window is redrawn. A simple-mode status bar is useful for displaying Help text for menu items while the user is scrolling through the menu.

The string that a status bar displays while in simple mode is maintained separately from the strings that it displays while in nonsimple mode. This means you can put the window in simple mode, set its text, and switch back to nonsimple mode without the nonsimple-mode text being changed.

When setting the text of a simple-mode status bar, you can specify any drawing technique except SBT_OWNERDRAW. A simple-mode status bar does not support owner drawing.

Default Status-Bar Message Processing

This section describes the messages handled by the window procedure for the predefined STATUSCLASSNAME class.

Message	Default processing
WM_CREATE	Initializes the status bar.
WM_DESTROY	Frees resources allocated for the status bar.
WM_GETFONT	Returns the handle to the current font with which the status bar draws its text.

Message	Default processing
WM_GETTEXT	Copies the text from the first part of a status bar to a buffer. It returns a 32-bit value that specifies the length, in characters, of the text and the technique used to draw the text.
WM_GETTEXTLENGTH	Returns a 32-bit value that specifies the length, in characters, of the text in the first part of a status bar and the technique used to draw the text.
WM_NCHITTEST	Returns the HTBOTTOMRIGHT value if the mouse cursor is in the sizing grip, causing the system to display the sizing cursor. If the mouse cursor is not in the sizing grip, the status bar passes this message to the **DefWindowProc** function.
WM_PAINT	Paints the invalid region of the status bar. If the *wParam* parameter is non-NULL, the control assumes that the value is an HDC and paints using that device context.
WM_SETFONT	Selects the font handle into the device context for the status bar.
WM_SETTEXT	Copies the specified text into the first part of a status bar, using the default drawing operation (specified as zero). It returns TRUE if successful, or FALSE otherwise.
WM_SIZE	Resizes the status bar based on the current width of the parent window's client area and the height of the current font of the status bar.

Status-Bar Example

The following example demonstrates how to create a status bar that has a sizing grip and divide the window into four equal parts based on the width of the parent window's client area:

```
// DoCreateStatusBar - creates a status bar and divides it into
//     the specified number of parts.
// Returns the handle to the status bar.
// hwndParent - parent window for the status bar.
// nStatusID - child window identifier.
// hinst - handle to the application instance.
// nParts - number of parts into which to divide the status bar.
HWND DoCreateStatusBar(HWND hwndParent, int nStatusID,
    HINSTANCE hinst, int nParts)
{
HWND hwndStatus;
RECT rcClient;
```

(continued)

(continued)

```
    HLOCAL hloc;
    LPINT lpParts;
    int i, nWidth;

    // Ensure that the common control DLL is loaded.
    InitCommonControls();

    // Create the status bar.
    hwndStatus = CreateWindowEx(
        0,                          // no extended styles
        STATUSCLASSNAME,            // name of status-bar class
        (LPCTSTR) NULL,             // no text when first created
        SBARS_SIZEGRIP |            // includes a sizing grip
        WS_CHILD,                   // creates a child window
        0, 0, 0, 0,                 // ignores size and position
        hwndParent,                 // handle to parent window
        (HMENU) nStatusID,          // child window identifier
        hinst,                      // handle to application instance
        NULL);                      // no window creation data

    // Get the coordinates of the parent window's client area.
    GetClientRect(hwndParent, &rcClient);

    // Allocate an array for holding the right-edge coordinates.
    hloc = LocalAlloc(LHND, sizeof(int) * nParts);
    lpParts = LocalLock(hloc);

    // Calculate the right-edge coordinate for each part, and
    // copy the coordinates to the array.
    nWidth = rcClient.right / nParts;
    for (i = 0; i < nParts; i++) {
        lpParts[i] = nWidth;
        nWidth += nWidth;
    }

    // Tell the status bar to create the window parts.
    SendMessage(hwndStatus, SB_SETPARTS, (WPARAM) nParts,
        (LPARAM) lpParts);

    // Free the array, and return.
    LocalUnlock(hloc);
    LocalFree(hloc);
    return hwndStatus;
    }
```

Status-Bar Updates in Internet Explorer

Status-bar controls in Microsoft Internet Explorer support the following new features:

Icon Support
Icons now can be displayed in a status bar. The **SB_SETICON** message is used to set the icon.

Tooltip Support
The status bar now supports tooltips. To enable tooltips, the SBT_TOOLTIPS style must be set when the status bar is created. The **SB_SETTIPTEXT** and **SB_GETTIPTEXT** messages are used to set and get the tooltip text, respectively. The tooltip for a part will only be displayed if the part has an icon and no text or if all of the text cannot be displayed inside the part. Tooltips are not supported in simple mode.

Enhanced Simple-Mode Support
The **SB_ISSIMPLE** message has been added to determine if a status bar is in simple mode. The **SBN_SIMPLEMODECHANGE** notification has been added to inform the owner that the simple mode has changed.

Background Color Support
The **SB_SETBKCOLOR** message has been added to allow the background color of a status bar to be modified.

Status-Bar Styles

Status-bar controls support the following style, in addition to standard window styles:

SBARS_SIZEGRIP	The status-bar control will include a sizing grip at the right end of the status bar. A sizing grip is similar to a sizing border; it is a rectangular area that the user can click and drag to resize the parent window.
SBARS_TOOLTIPS	**Version 5.80.** Identical to SBT_TOOLTIPS. Use this flag for versions 5.00 and later.
SBT_TOOLTIPS	**Version 4.71.** Use this style to enable tooltips.

Status-Bar Reference

Status-Bar Functions

CreateStatusWindow

Creates a status window, which is typically used to display the status of an application. The window generally appears at the bottom of the parent window, and it contains the specified text.

```
HWND CreateStatusWindow(
  LONG style,
  LPCTSTR lpszText,
  HWND hwndParent,
  UINT wID
);
```

Parameters

style
Window styles for the status window. This parameter must include the WS_CHILD style and should include the WS_VISIBLE style.

lpszText
Address of a null-terminated string that specifies the status text for the first part.

hwndParent
Handle to the parent window.

wID
Control identifier for the status window. The window procedure uses this value to identify messages it sends to the parent window.

Return Values

Returns the handle for the status window if successful, or NULL otherwise. To get extended error information, call **GetLastError**.

Remarks

The **CreateStatusWindow** function calls the **CreateWindow** function to create the window. It passes the parameters to **CreateWindow** without modification and sets the position, width, and height parameters to default values.

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 commctrl.h.
Import Library: comctl32.lib.

DrawStatusText

The **DrawStatusText** function draws the specified text in the style of a status window with borders.

```
void DrawStatusText(
  HDC hdc,
  LPRECT lprc,
  LPCTSTR pszText,
  UINT uFlags
);
```

Parameters

hdc
Handle to the display context for the window.

lprc
Pointer to a **RECT** structure that contains the position, in client coordinates, of the rectangle in which the text is drawn. The function draws the borders just inside the edges of the specified rectangle.

pszText
Pointer to a null-terminated string that specifies the text to display. Tab characters in the string determine whether the string is left-aligned, right-aligned, or centered.

uFlags
Text drawing flags. This parameter can be a combination of these values:

SBT_NOBORDERS	Prevents borders from being drawn around the specified text.
SBT_POPOUT	Draws highlighted borders that make the text stand out.
SBT_RTLREADING	Indicates that the string pointed to by *pszText* will be displayed in the opposite direction to the text in the parent window.

Return Values

This function does not return a value.

Remarks

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, the *pszText* string will be displayed in the same direction as the text in its parent window. If

SBT_RTLREADING is set, the *pszText* string will read in the opposite direction from the text in the parent 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 commctrl.h.
Import Library: comctl32.lib.

MenuHelp

Processes **WM_MENUSELECT** and **WM_COMMAND** messages, and displays Help text about the current menu in the specified status window.

```
void MenuHelp(
  UINT uMsg,
  WPARAM wParam,
  LPARAM lParam,
  HMENU hMainMenu,
  HINSTANCE hInst,
  HWND hwndStatus,
  LPUINT lpwIDs
);
```

Parameters

uMsg
Message being processed. This can be either **WM_MENUSELECT** or **WM_COMMAND**.

wParam
wParam of the message specified in *uMsg*.

lParam
lParam of the message specified in *uMsg*.

hMainMenu
Handle to the application's main menu.

hInst
Handle to the module that contains the string resources.

hwndStatus
Handle to the status window.

lpwIDs
Address of an array that contains pairs of string resource identifiers and menu handles. The function searches the array for the handle to the selected menu and, if found, uses the corresponding resource identifier to load the appropriate Help string.

Return Values

This function does not return a value.

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 commctrl.h.
Import Library: comctl32.lib.

Status-Bar Messages

SB_GETBORDERS

Retrieves the current widths of the horizontal and vertical borders of a status window.

```
SB_GETBORDERS
    wParam = 0;
    lParam = (LPARAM) (LPINT) aBorders;
```

Parameters

aBorders

Address of an integer array that has three elements. The first element receives the width of the horizontal border, the second receives the width of the vertical border, and the third receives the width of the border between rectangles.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

The borders determine the spacing between the outside edge of the window and the rectangles within the window that contain text. The borders also determine the spacing between rectangles.

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 commctrl.h.

SB_GETICON

Retrieves the icon for a part in a status bar.

```
SB_GETICON
    wParam = (WPARAM)(INT)iPart;
    lParam = 0;
```

Parameters

iPart

Zero-based index of the part that contains the icon to be retrieved. If this parameter is –1, the status bar is assumed to be a **Simple Mode** status bar.

Return Values

Returns the handle to the icon if successful, or NULL otherwise.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

SB_GETPARTS

Retrieves a count of the parts in a status window. The message also retrieves the coordinate of the right edge of the specified number of parts.

```
SB_GETPARTS
    wParam = (WPARAM) nParts;
    lParam = (LPARAM) (LPINT) aRightCoord;
```

Parameters

nParts

Number of parts for which to retrieve coordinates. If this parameter is greater than the number of parts in the window, the message retrieves coordinates for existing parts only.

aRightCoord

Address of an integer array that has the same number of elements as parts specified by *nParts*. Each element in the array receives the client coordinate of the right edge of the corresponding part. If an element is set to –1, the position of the right edge for that

part extends to the right edge of the window. To retrieve the current number of parts, set this parameter to zero.

Return Values

Returns the number of parts in the window if successful, or zero otherwise.

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 commctrl.h.

SB_GETRECT

Retrieves the bounding rectangle of a part in a status window.

```
SB_GETRECT
    wParam = (WPARAM) iPart;
    lParam = (LPARAM) (LPRECT) lprc;
```

Parameters

iPart
Zero-based index of the part whose bounding rectangle is to be retrieved.

lprc
Address of a **RECT** structure that receives the bounding rectangle.

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: Requires version 1.0 or later.
Header: Declared in commctrl.h.

SB_GETTEXT

The SB_GETTEXT message retrieves the text from the specified part of a status window.

```
SB_GETTEXT
   wParam = (WPARAM) iPart;
   lParam = (LPARAM) (LPSTR) szText;
```

Parameters

iPart

Zero-based index of the part from which to retrieve text.

szText

Pointer to the buffer that receives the text. This parameter is a null-terminated string.

Return Values

Returns a 32-bit value that consists of two 16-bit values. The low word specifies the length, in characters, of the text. The high word specifies the type of operation used to draw the text. The type can be one of the following values:

0	The text is drawn with a border to appear lower than the plane of the window.
SBT_NOBORDERS	The text is drawn without borders.
SBT_POPOUT	The text is drawn with a border to appear higher than the plane of the window.
SBT_RTLREADING	The text will be displayed in the opposite direction to the text in the parent window.

If the text has the SBT_OWNERDRAW drawing type, this message returns the 32-bit value associated with the text instead of the length and operation type.

Remarks

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). If SBT_RTLREADING is set, the *szText* string will read in the opposite direction from the text in the parent 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 commctrl.h.

See Also

SB_SETTEXT

SB_GETTEXTLENGTH

The SB_GETTEXTLENGTH message retrieves the length, in characters, of the text from the specified part of a status window.

```
SB_GETTEXTLENGTH
    wParam = (WPARAM) iPart;
    lParam = 0;
```

Parameters

iPart

Zero-based index of the part from which to retrieve text.

Return Values

Returns a 32-bit value that consists of two 16-bit values. The low word specifies the length, in characters, of the text. The high word specifies the type of operation used to draw the text. The type can be one of the following values:

0	The text is drawn with a border to appear lower than the plane of the window.
SBT_NOBORDERS	The text is drawn without borders.
SBT_OWNERDRAW	The text is drawn by the parent window.
SBT_POPOUT	The text is drawn with a border to appear higher than the plane of the window.
SBT_RTLREADING	The text will be displayed in the opposite direction to the text in the parent window.

Remarks

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). If SBT_RTLREADING is set, the specified status window text will read in the opposite direction from the text in the parent 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 commctrl.h.

SB_GETTIPTEXT

Retrieves the tooltip text for a part in a status bar. The status bar must have been created with the SBT_TOOLTIPS style to enable tooltips.

```
SB_GETTIPTEXT
    wParam = MAKEWPARAM(iPart, nSize);
    lParam = (LPARAM)(LPCTSTR)lpszTooltip;
```

Parameters

iPart
Zero-based index of the part that will receive the tooltip text.

nSize
Size of the buffer at *lpszTooltip*, in characters.

lpszTooltip
Address of a character buffer that receives the tooltip text.

Return Values

The return value is not used.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

See Also

Status-Bar Updates in Internet Explorer

SB_GETUNICODEFORMAT

Retrieves the UNICODE character format flag for the control.

```
SB_GETUNICODEFORMAT
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the UNICODE format flag for the control. If this value is nonzero, the control is using UNICODE characters. If this value is zero, the control is using ANSI characters.

Remarks

See the remarks for **CCM_GETUNICODEFORMAT** for a discussion of this message.

Requirements

Version 4.00 or later of Comctl32.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 commctrl.h.

See Also

SB_SETUNICODEFORMAT

SB_ISSIMPLE

Checks a status-bar control to determine if it is in simple mode.

```
SB_ISSIMPLE
   wParam = 0;
   lParam = 0;
```

Return Values

Returns nonzero if the status-bar control is in simple mode, or zero otherwise.

Requirements

Version 4.70 or later of Comctl32.dll

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

SB_SETBKCOLOR

Sets the background color in a status bar.

```
SB_SETBKCOLOR
    wParam = 0;
    lParam = (LPARAM)(COLORREF)clrBk;
```

Parameters

clrBk

COLORREF value that specifies the new background color. Specify the CLR_DEFAULT value to cause the status bar to use its default background color.

Return Values

Returns the previous background color, or CLR_DEFAULT if the background color is the default color.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

SB_SETICON

Sets the icon for a part in a status bar.

```
SB_SETICON
    wParam = (WPARAM)(INT)iPart;
    lParam = (LPARAM)(HICON)hIcon;
```

Parameters

iPart

Zero-based index of the part that will receive the icon. If this parameter is –1, the status bar is assumed to be a simple status bar.

hIcon

Handle to the icon to be set. If this value is NULL, the icon is removed from the part.

Return Values

Returns nonzero if successful, or zero otherwise.

Remarks

The status bar will not destroy the icon. It is the calling application's responsibility to keep track of and destroy any icons.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

SB_SETMINHEIGHT

Sets the minimum height of a status window's drawing area.

```
SB_SETMINHEIGHT
    wParam = (WPARAM) minHeight;
    lParam = 0;
```

Parameters

minHeight
Minimum height, in pixels, of the window.

Return Values

No return value.

Remarks

The minimum height is the sum of *wParam* and twice the width, in pixels, of the vertical border of the status window. An application must send the **WM_SIZE** message to the status window to redraw the window. The *wParam* and *lParam* parameters of the **WM_SIZE** message should be set to zero.

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 commctrl.h.

SB_SETPARTS

Sets the number of parts in a status window and the coordinate of the right edge of each part.

```
SB_SETPARTS
    wParam = (WPARAM) nParts;
    lParam = (LPARAM) (LPINT) aWidths;
```

Parameters

nParts

Number of parts to set (cannot be greater than 256).

aWidths

Pointer to an integer array. The number of elements is specified in *nParts*. Each element specifies the position, in client coordinates, of the right edge of the corresponding part. If an element is –1, the right edge of the corresponding part extends to the border of the window.

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: Requires version 1.0 or later.
Header: Declared in commctrl.h.

SB_SETTEXT

The SB_SETTEXT message sets the text in the specified part of a status window.

```
SB_SETTEXT
    wParam = (WPARAM) iPart | uType;
    lParam = (LPARAM) (LPSTR) szText;
```

Parameters

iPart

Zero-based index of the part to set. If this parameter is set to SB_SIMPLEID, the status window is assumed to be a simple window with only one part.

uType

Type of drawing operation. This parameter can be one of the following values:

0	The text is drawn with a border to appear lower than the plane of the window.
SBT_NOBORDERS	The text is drawn without borders.
SBT_OWNERDRAW	The text is drawn by the parent window.
SBT_POPOUT	The text is drawn with a border to appear higher than the plane of the window.
SBT_RTLREADING	The text will be displayed in the opposite direction to the text in the parent window.

szText
Pointer to a null-terminated string that specifies the text to set. If *uType* is SBT_OWNERDRAW, this parameter represents 32 bits of data. The parent window must interpret the data and draw the text when it receives the WM_DRAWITEM message. The text for each part is limited to 127 characters.

Return Values
Returns TRUE if successful, or FALSE otherwise.

Remarks
The message invalidates the portion of the window that has changed, causing it to display the new text when the window next receives the WM_PAINT message.

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). If SBT_RTLREADING is set, the *szText* string will read in the opposite direction from the text in the parent window.

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 commctrl.h.

SB_GETTEXT

SB_SETTIPTEXT

Sets the tooltip text for a part in a status bar. The status bar must have been created with the SBT_TOOLTIPS style to enable tooltips.

```
SB_SETTIPTEXT
   wParam = (WPARAM)(INT)iPart;
   lParam = (LPARAM)(LPCTSTR)lpszTooltip;
```

Parameters

iPart
Zero-based index of the part that will receive the tooltip text.

lpszTooltip
Address of a character buffer that contains the new tooltip text.

Return Values

The return value is not used.

Remarks

See *Status-Bar Updates in Internet Explorer* for further information.

This tooltip text is displayed in two situations:

- When the corresponding pane in the status bar contains only an icon.
- When the corresponding pane in the status bar contains text that is truncated due to the size of the pane.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

SB_SETUNICODEFORMAT

Sets the UNICODE character format flag for the control. This message allows you to change the character set used by the control at run time, instead of having to re-create the control.

```
SB_SETUNICODEFORMAT
   wParam = (WPARAM)(BOOL)fUnicode;
   lParam = 0;
```

Parameters

fUnicode

Determines the character set that is used by the control. If this value is nonzero, the control will use UNICODE characters. If this value is zero, the control will use ANSI characters.

Return Values

Returns the previous UNICODE format flag for the control.

Remarks

See the remarks for **CCM_SETUNICODEFORMAT** for a discussion of this message.

Requirements

Version 4.00 or later of Comctl32.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 commctrl.h.

See Also

SB_GETUNICODEFORMAT

SB_SIMPLE

Specifies whether a status window displays simple text or displays all window parts set by a previous **SB_SETPARTS** message.

```
SB_SIMPLE
    wParam = (WPARAM) (BOOL) fSimple;
    lParam = 0;
```

Parameters

fSimple

Display type flag. If this parameter is TRUE, the window displays simple text. If it is FALSE, it displays multiple parts.

Return Values

The return value is not used.

Remarks

If the status window is being changed from nonsimple to simple, or vice versa, the window is immediately redrawn.

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 commctrl.h.

Status-Bar Notifications

NM_CLICK (status bar)

Notifies a status-bar control's parent window that the user has clicked the left mouse button within the control. NM_CLICK is sent in the form of a **WM_NOTIFY** message.

```
NM_CLICK
    lpnm = (LPNMMOUSE) lParam;
```

Parameters

lpnm

Address of an **NMMOUSE** structure that contains additional information about this notification message. The **dwItemSpec** member contains the index of the section that was clicked.

Return Values

Return TRUE to indicate that the mouse click was handled and suppress default processing by the system. Return FALSE to allow default processing of the click.

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in commctrl.h.

NM_DBLCLK (status bar)

Notifies a status-bar control's parent window that the user has double-clicked the left mouse button within the control. NM_DBLCLK is sent in the form of a **WM_NOTIFY** message.

```
NM_DBLCLK
    lpnm = (LPNMMOUSE) lParam;
```

Parameters

lpnm

Address of an **NMMOUSE** structure that contains additional information about this notification message. The **dwItemSpec** member contains the index of the section that was clicked.

Return Values

Return TRUE to indicate that the mouse click was handled and suppress default processing by the system. Return FALSE to allow default processing of the click.

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 commctrl.h.

NM_RCLICK (status bar)

Notifies a status-bar control's parent window that the user has clicked the right mouse button within the control. This notification is sent in the form of a **WM_NOTIFY** message.

```
NM_RCLICK
    lpnm = (LPNMMOUSE) lParam;
```

Parameters

lpnm

Address of an **NMMOUSE** structure that contains additional information about this notification message. The **dwItemSpec** member contains the index of the section that was clicked.

Return Value

Return TRUE to indicate that the mouse click was handled and suppress default processing by the system. Return FALSE to allow default processing of the click.

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 commctrl.h.

NM_RDBLCLK (status bar)

Notifies a status-bar control's parent window that the user has double-clicked the right mouse button within the control. NM_RDBLCLK is sent in the form of a **WM_NOTIFY** message.

```
NM_RDBLCLK
    lpnm = (LPNMMOUSE) lParam;
```

Parameters

lpnm
Address of an **NMMOUSE** structure that contains additional information about this notification message. The **dwItemSpec** member contains the index of the section that was clicked.

Return Values

Return TRUE to indicate that the mouse click was handled and suppress default processing by the system. Return FALSE to allow default processing of the click.

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 commctrl.h.

SBN_SIMPLEMODECHANGE

Sent by a status-bar control when the simple mode changes due to a **SB_SIMPLE** message. This notification is sent in the form of a **WM_NOTIFY** message.

```
SBN_SIMPLEMODECHANGE
    lpnmh = (NMHDR*) lParam;
```

Parameters

lpnmh
Address of an **NMHDR** structure that contains information about the notification.

Return Values

The return value is ignored by the status bar.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

CHAPTER 24

Tab Controls

A *tab control* is analogous to the dividers in a notebook or the labels in a file cabinet. By using a tab control, an application can define multiple *pages* for the same area of a window or dialog box. Each page consists of a certain type of information or a group of controls that the application displays when the user selects the corresponding tab.

About Tab Controls

You can create a tab control by calling the **CreateWindowEx** function, specifying the WC_TABCONTROL window class. This window class is registered when the common controls dynamic-link library (DLL) is loaded. To ensure that the DLL is loaded, use the **InitCommonControls** function.

You send messages to a tab control to add tabs and otherwise affect the control's appearance and behavior. Each message has a corresponding macro that you can use instead of sending the message explicitly. You cannot disable an individual tab in a tab control. However, you can disable a tab control in a property sheet by disabling the corresponding page.

About Tab Control Styles

You can apply certain characteristics to tab controls by specifying tab control styles when the control is created. For example, you can specify the alignment and general appearance of the tabs in a tab control.

You can cause the tabs to look like buttons by specifying the **TCS_BUTTONS** style. Tabs in this type of tab control should serve the same function as button controls; that is, clicking a tab should carry out a command instead of displaying a page. Because the display area in a button tab control is typically not used, no border is drawn around it.

You can cause a tab to receive the input focus when clicked by specifying the **TCS_FOCUSONBUTTONDOWN** style. This style is typically used only with the **TCS_BUTTONS** style. You can specify that a tab does not receive input focus when clicked by using the **TCS_FOCUSNEVER** style.

By default, a tab control displays only one row of tabs. If not all tabs can be shown at once, the tab control displays an up-down control so that the user can scroll additional tabs into view. You can cause a tab control to display multiple rows of tabs, if necessary, by specifying the **TCS_MULTILINE** style. With this style, all tabs can be displayed at once. The tabs are left-aligned within each row unless you specify the **TCS_RIGHTJUSTIFY** style. In this case, the width of each tab is increased so that each row of tabs fills the entire width of the tab control.

A tab control automatically sizes each tab to fit its icon, if any, and its label. To give all tabs the same width, you can specify the **TCS_FIXEDWIDTH** style. The control sizes all the tabs to fit the widest label, or you can assign a specific width and height by using the **TCM_SETITEMSIZE** message. Within each tab, the control centers the icon and label, placing the icon to the left of the label. You can force the icon to the left, leaving the label centered, by specifying the **TCS_FORCEICONLEFT** style. You can left-align both the icon and label by using the **TCS_FORCELABELLEFT** style. You cannot use the **TCS_FIXEDWIDTH** style with the **TCS_RIGHTJUSTIFY** style.

You can specify that the parent window draws the tabs in the control by using the **TCS_OWNERDRAWFIXED** style. For more information, see *Owner-Drawn Tabs*.

You can specify that a tab control will create a tooltip control by using the **TCS_TOOLTIPS** style. For more information about this, see *Tab Control Tooltips*.

Tabs and Tab Attributes

Each tab in a tab control consists of an icon, a label, and application-defined data. This information is specified by a **TCITEM** structure. You can add tabs to a tab control, get the number of tabs, retrieve and set the contents of a tab, and delete tabs. Tabs are identified by their zero-based index.

To add tabs to a tab control, use the **TCM_INSERTITEM** message, specifying the position of the item and the address of a **TCITEM** structure. You can retrieve and set the contents of an existing tab by using the **TCM_GETITEM** and **TCM_SETITEM** messages. For each tab, you can specify an icon, a label, or both. You can also specify application-defined data to associate with the tab.

You can retrieve the current number of tabs by using the **TCM_GETITEMCOUNT** message, delete a tab by using the **TCM_DELETEITEM** message, and delete all tabs in a tab control by using the **TCM_DELETEALLITEMS** message.

You can associate application-defined data with each tab. For example, you might save information about each page with its corresponding tab. By default, a tab control allocates four extra bytes per tab for application-defined data. You can change the number of extra bytes per tab by using the **TCM_SETITEMEXTRA** message. You can only use this message when the tab control is empty.

The application-defined data is specified by the **lParam** member of the **TCITEM** structure. If you use more than 4 bytes of application-defined data, you need to define your own structure and use it instead of **TCITEM**. You can retrieve and set application-defined data the same way you retrieve and set other information about a tab—by using the **TCM_GETITEM** and **TCM_SETITEM** messages.

The first member of your structure must be a **TCITEMHEADER** structure, and the remaining members must specify application-defined data. **TCITEMHEADER** is identical to **TCITEM**, except it does not have the **lParam** member. The difference between the size of your structure and the size of **TCITEMHEADER** should equal the number of extra bytes per tab.

Display Area

The *display area* of a tab control is the area in which an application displays the current page. Typically, an application creates a child window or dialog box, setting the window size and position to fit the display area. Given the window rectangle for a tab control, you can calculate the bounding rectangle of the display area by using the **TCM_ADJUSTRECT** message.

Sometimes the display area must be a particular size—for example, the size of a modeless child dialog box. Given the bounding rectangle for the display area, you can use **TCM_ADJUSTRECT** to calculate the corresponding window rectangle for the tab control.

Tab Selection

When the user selects a tab, a tab control sends its parent window notification messages in the form of **WM_NOTIFY** messages. The **TCN_SELCHANGING** notification message is sent before the selection changes, and the **TCN_SELCHANGE** notification message is sent after the selection changes.

You can process **TCN_SELCHANGING** to save the state of the outgoing page. You can return TRUE to prevent the selection from changing. For example, you might not want to switch away from a child dialog box in which a control has an invalid setting.

You must process **TCN_SELCHANGE** to display the incoming page in the display area. This might simply entail changing the information displayed in a child window. More often, each page consists of a child window or dialog box. In this case, an application might process this notification by destroying or hiding the outgoing child window or dialog box and by creating or showing the incoming child window or dialog box.

You can retrieve and set the current selection by using the **TCM_GETCURSEL** and **TCM_SETCURSEL** messages.

Tab Control Image Lists

Each tab can have an icon associated with it, which is specified by an index in the image list for the tab control. When a tab control is created, it has no image list associated with it. An application can create an image list by using the **ImageList_Create** function and then assign it to a tab control by using the **TCM_SETIMAGELIST** message.

You can add images to a tab control's image list just as you would to any other image list. However, an application should remove images by using the **TCM_REMOVEIMAGE** message instead of the **ImageList_Remove** function. This message ensures that each tab remains associated with the same image as before.

Destroying a tab control does not destroy an image list that is associated with it. You must destroy the image list separately. This is useful if you want to assign the same image list to multiple tab controls.

To retrieve the handle to the image list currently associated with a tab control, you can use the **TCM_GETIMAGELIST** message.

Tab Size and Position

Each tab in a tab control has a size and position. You can set the size of tabs, retrieve the bounding rectangle of a tab, or determine which tab is at a specified position.

For fixed-width and owner-drawn tab controls, you can set the exact width and height of tabs by using the **TCM_SETITEMSIZE** message. In other tab controls, each tab's size is calculated based on the icon and label for the tab. The tab control includes space for a border and an additional margin. You can set the thickness of the margin by using the **TCM_SETPADDING** message.

You can determine the current bounding rectangle for a tab by using the **TCM_GETITEMRECT** message. You can determine which tab, if any, is at a specified location by using the **TCM_HITTEST** message.

In a tab control with the **TCS_MULTILINE** style, you can determine the current number of rows of tabs by using the **TCM_GETROWCOUNT** message.

Owner-Drawn Tabs

If a tab control has the **TCS_OWNERDRAWFIXED** style, the parent window must paint tabs by processing the **WM_DRAWITEM** message. The tab control sends this message whenever a tab needs to be painted. The *lParam* parameter specifies the address of a **DRAWITEMSTRUCT** structure, which contains the index of the tab, its bounding rectangle, and the device context (DC) in which to draw.

By default, the **itemData** member of **DRAWITEMSTRUCT** contains the value of the **lParam** member of the **TCITEM** structure. However, if you change the amount of application-defined data per tab, **itemData** contains the address of the data instead. You can change the amount of application-defined data per tab by using the **TCM_SETITEMEXTRA** message.

To specify the size of items in a tab control, the parent window must process the **WM_MEASUREITEM** message. Because all tabs in an owner-drawn tab control are the same size, this message is sent only once. There is no tab control style for owner-drawn tabs of varying size. You can also set the width and height of tabs by using the **TCM_SETITEMSIZE** message.

Tab Control Tooltips

You can use a tooltip control to provide a brief description of each tab in a tab control. A tab control that has the **TCS_TOOLTIPS** style creates a tooltip control when it is created and destroys the tooltip control when it is destroyed. You can also create a tooltip control and assign it to a tab control.

If you use a tooltip control with a tab control, the parent window must process the **TTN_NEEDTEXT** notification message to provide a description of each tab on request.

To use the same tooltip control with more than one tab control, create the tooltip control yourself and assign it to the tab control by using the **TCM_SETTOOLTIPS** message. You can get the handle to a tab control's current tooltip control by using the **TCM_GETTOOLTIPS** message. If you create your own tooltip control, you should *not* use the **TCS_TOOLTIPS** style. For more information about tooltip controls, see *Tooltip Controls*.

Default Tab Control Message Processing

This section describes the message processing performed by a tab control. Messages specific to tab controls are discussed in other sections of this documentation.

Message	Processing performed
WM_CAPTURECHANGED	Does nothing if the tab control released the mouse capture itself. If another window captured the mouse and a button is held down, the command releases the button.
WM_CREATE	Allocates and initializes an internal data structure. The control creates a tooltip control if the **TCS_TOOLTIPS** style is specified.
WM_DESTROY	Frees resources allocated during WM_CREATE processing.
WM_GETDLGCODE	Returns a combination of the DLGC_WANTARROWS and DLGC_WANTCHARS values.
WM_GETFONT	Returns the handle to the font used for labels.
WM_KEYDOWN	Processes direction keys and changes the selection, if appropriate.
WM_KILLFOCUS	Invalidates the tab that has the focus so it will be repainted to reflect an unfocused state.
WM_LBUTTONDOWN	Forwards the message to the tooltip control, if any, and changes the selection if the user is clicking a tab. If the user is clicking a button, the control redraws the button to give a sunken appearance and captures the mouse. If the user is clicking either a tab or button and the **TCS_FOCUSONBUTTONDOWN** style is specified, the control sets the focus to itself.
WM_LBUTTONUP	Releases the mouse if a button was pressed. If the cursor is over the button and is being held down, the control changes the selection accordingly and redraws the button.

(continued)

(continued)

Message	Processing performed
WM_MOUSEMOVE	Forwards the message to the tooltip control, if any. If the **TCS_BUTTONS** style is specified and the mouse button is being held down after clicking, the control may also redraw the affected button to give it a raised or sunken appearance.
WM_NOTIFY	Forwards notification messages sent by the tooltip control.
WM_PAINT	Draws a border around the display area (unless the **TCS_BUTTONS** style is specified) and paints any tabs that intersect the invalid rectangle. For each tab, it draws the body of the tab (or sends a **WM_DRAWITEM** message to the parent window) and then draws a border around the tab. If the *wParam* parameter is non-NULL, the control assumes that the value is an HDC and paints using that device context.
WM_RBUTTONDOWN	Sends an **NM_RCLICK** notification message to the parent window.
WM_SETFOCUS	Invalidates the tab that has the focus so that it will be repainted to reflect a focused state.
WM_SETFONT	Sets the font used for labels.
WM_SETREDRAW	Sets the state of an internal flag that determines whether the control is repainted when items are inserted and deleted, when the font is changed, and so on.
WM_SIZE	Recalculates the positions of tabs and may invalidate part of the tab control to force repainting of some or all tabs.

Using Tab Controls

This section provides two examples that use tab controls. The first example demonstrates how to use a tab control to switch between multiple pages of text in an application's main window. The second example demonstrates how to use a tab control to switch between multiple pages of controls in a dialog box.

Creating a Tab Control

The example in this section demonstrates how to create a tab control and display it in the client area of the application's main window. The application displays a third window (a static control) in the display area of the tab control. The parent window positions and sizes the tab control and static control when it processes the **WM_SIZE** message.

There are seven tabs, one for each day of the week. When the user selects a tab, the application displays the name of the corresponding day in the static control. The following global variables are used in this example.

```
// Global variables

      HINSTANCE g_hinst;    // handle to application
                            // instance
      char g_achTemp[256];  // temporary buffer for
                            // strings
      HWND g_hwndMain;      // main application window
      HWND g_hwndTab;       // tab control
      HWND g_hwndDisplay;   // handle to static control
                    //   in tab control's display area
```

The following function creates the tab control and adds a tab for each day of the week. The names of the days are defined as string resources, consecutively numbered starting with IDS_FIRSTDAY (defined in the application's header file). Both the parent window and the tab control must have the WS_CLIPSIBLINGS window style. The application's initialization function calls this function after creating the main window.

```
// DoCreateTabControl - creates a tab control, sized to fit the
      //     specified parent window's client area, and
      //     adds some tabs.
      // Returns the handle to the tab control.
      // hwndParent - parent window (the application's
      //     main window).

   HWND WINAPI DoCreateTabControl(HWND hwndParent)
   {
   RECT rcClient;
   HWND hwndTab;
   TCITEM tie;
   int i;

   // Get the dimensions of the parent window's client
   // area, and create a tab control child window of
   // that size.
   GetClientRect(hwndParent, &rcClient);
   InitCommonControls();
   hwndTab = CreateWindow(
      WC_TABCONTROL, "",
      WS_CHILD | WS_CLIPSIBLINGS | WS_VISIBLE,
      0, 0, rcClient.right, rcClient.bottom,
      hwndParent, NULL, g_hinst, NULL
```

(continued)

(continued)

```
    );
if (hwndTab == NULL)
    return NULL;

// Add tabs for each day of the week.
tie.mask = TCIF_TEXT | TCIF_IMAGE;
tie.iImage = -1;
tie.pszText = g_achTemp;

for (i = 0; i < 7; i++) {
    LoadString(g_hinst, IDS_FIRSTDAY + i,
            g_achTemp, sizeof(g_achTemp));
    if (TabCtrl_InsertItem(hwndTab, i, &tie) == -1) {
        DestroyWindow(hwndTab);
        return NULL;
    }
}
return hwndTab;
}
```

The following function creates the static control that occupies the tab control's display area. The application's initialization function calls this function after creating the main window and the tab control.

```
    // DoCreateDisplayWindow - creates a child window
    //     (a static control) to occupy the tab
    //     control's display area.
    // Returns the handle to the static control.
    // hwndParent - parent window (the application's
    //     main window).

HWND WINAPI DoCreateDisplayWindow(HWND hwndParent)
{
HWND hwndStatic = CreateWindow("STATIC", "",
    WS_CHILD | WS_VISIBLE | WS_BORDER,
    0, 0, CW_USEDEFAULT, CW_USEDEFAULT,
    hwndParent, NULL, g_hinst, NULL);

return hwndStatic;
}
```

Following are the relevant portions of the application's window procedure. The application processes the **WM_SIZE** message to position and size the tab control and the static control. To determine the appropriate position and size for the static control,

this example sends the tab control a **TCM_ADJUSTRECT** message (by using the **TabCtrl_AdjustRect** macro).

When a tab is selected, the tab control sends a **WM_NOTIFY** message, specifying the **TCN_SELCHANGE** notification message. The application processes this notification message by setting the text of the static control.

```
    // MainWindowProc - processes the message for the
    //     main window class.
    // The return value depends on the message.
    // hwnd - handle to the window.
    // uMsg - identifier for the message.
    // wParam - message-specific parameter.
    // lParam - message-specific parameter.

LRESULT CALLBACK MainWindowProc(
    HWND hwnd,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam
    )
{
switch (uMsg) {
    case WM_SIZE: {
            HDWP hdwp;
            RECT rc;

            // Calculate the display rectangle,
            // assuming the tab control is the size of
            // the client area.
            SetRect(&rc, 0, 0,
                    LOWORD(lParam), HIWORD(lParam));
            TabCtrl_AdjustRect(g_hwndTab, FALSE, &rc);

            // Size the tab control to fit the client
            // area.
            hdwp = BeginDeferWindowPos(2);
            DeferWindowPos(hdwp, g_hwndTab, NULL, 0, 0,
                LOWORD(lParam), HIWORD(lParam),
                SWP_NOMOVE | SWP_NOZORDER
                );

            // Position and size the static control to
            // fit the tab control's display area, and
            // make sure the static control is in front
            // of the tab control.
```

(continued)

(continued)

```
                DeferWindowPos(hdwp,
rc.top,                 g_hwndDisplay, HWND_TOP, rc.left,
rc.top, 0               rc.right - rc.left, rc.bottom -
                    );
                EndDeferWindowPos(hdwp);
            }
            break;

        case WM_NOTIFY:
            switch (HIWORD(wParam)) {
                case 0:
                    .
                    .   // menu command processing
                    .

                case TCN_SELCHANGE: {
(g_hwndTab);            int iPage = TabCtrl_GetCurSel
+ iPage,                LoadString(g_hinst, IDS_FIRSTDAY
                            g_achTemp, sizeof(g_achTemp));
WM_SETTEXT, 0           SendMessage(g_hwndDisplay,
                            (LPARAM) g_achTemp);
                }
                break;
            }
            break;

            .
            .   // additional message processing
            .

        default:
            return DefWindowProc(hwnd, uMsg, wParam,
lParam);
    }
    return 0;
}
```

Creating a Tabbed Dialog Box

The example in this section demonstrates how to create a dialog box that uses tabs to provide multiple pages of controls. The main dialog box is a modal dialog box. Each page of controls is defined by a dialog box template that has the WS_CHILD style. When a tab is selected, a modeless dialog box is created for the incoming page and the dialog box for the outgoing page is destroyed.

Note In many cases, you can implement multiple-page dialog boxes more easily by using property sheets. For more information about property sheets, see *Property Sheets.*

The template for the main dialog box simply defines two button controls. When processing the WM_INITDIALOG message, the dialog box procedure creates a tab control and loads the dialog template resources for each of the child dialog boxes.

The information is saved in an application-defined structure called DLGHDR. A pointer to this structure is associated with the dialog box window by using the **SetWindowLong** function. The structure is defined in the application's header file, as follows:

```
#define C_PAGES 3

typedef struct tag_dlghdr {
HWND hwndTab;       // tab control
HWND hwndDisplay;   // current child dialog box
RECT rcDisplay;     // display rectangle for the tab
                    // control
DLGTEMPLATE *apRes[C_PAGES];
} DLGHDR;
```

The following function processes the WM_INITDIALOG message for the main dialog box. The function allocates the DLGHDR structure, loads the dialog template resources for the child dialog boxes, and creates the tab control.

The size of each child dialog box is specified by the **DLGTEMPLATE** structure. The function examines the size of each dialog box and uses the macro for the **TCM_ADJUSTRECT** message to calculate an appropriate size for the tab control. Then it sizes the dialog box and positions the two buttons accordingly. This example sends **TCM_ADJUSTRECT** by using the **TabCtrl_AdjustRect** macro.

```
VOID WINAPI OnTabbedDialogInit(HWND hwndDlg)
{
DLGHDR *pHdr = (DLGHDR *) LocalAlloc(LPTR,
sizeof(DLGHDR));
DWORD dwDlgBase = GetDialogBaseUnits();
int cxMargin = LOWORD(dwDlgBase) / 4;
int cyMargin = HIWORD(dwDlgBase) / 8;
```

(continued)

(continued)

```
    TCITEM tie;
    RECT rcTab;
    HWND hwndButton;
    RECT rcButton;
    int i;

    // Save a pointer to the DLGHDR structure.
    SetWindowLong(hwndDlg, GWL_USERDATA, (LONG) pHdr);

    // Create the tab control.
    InitCommonControls();
    pHdr->hwndTab = CreateWindow(
        WC_TABCONTROL, "",
        WS_CHILD | WS_CLIPSIBLINGS | WS_VISIBLE,
        0, 0, 100, 100,
        hwndDlg, NULL, g_hinst, NULL
        );
    if (pHdr->hwndTab == NULL) {
        // handle error
    }

    // Add a tab for each of the three child dialog boxes.
    tie.mask = TCIF_TEXT | TCIF_IMAGE;
    tie.iImage = -1;
    tie.pszText = "First";
    TabCtrl_InsertItem(pHdr->hwndTab, 0, &tie);
    tie.pszText = "Second";
    TabCtrl_InsertItem(pHdr->hwndTab, 1, &tie);
    tie.pszText = "Third";
    TabCtrl_InsertItem(pHdr->hwndTab, 2, &tie);

    // Lock the resources for the three child dialog boxes.
    pHdr->apRes[0] = DoLockDlgRes
(MAKEINTRESOURCE(DLG_FIRST));
    pHdr->apRes[1] = DoLockDlgRes
(MAKEINTRESOURCE(DLG_SECOND));
    pHdr->apRes[2] = DoLockDlgRes
(MAKEINTRESOURCE(DLG_THIRD));
    // Determine the bounding rectangle for all child
    // dialog boxes.
    SetRectEmpty(&rcTab);
    for (i = 0; i < C_PAGES; i++) {
        if (pHdr->apRes[i]->cx > rcTab.right)
            rcTab.right = pHdr->apRes[i]->cx;
```

```
            if (pHdr->apRes[i]->cy > rcTab.bottom)
                rcTab.bottom = pHdr->apRes[i]->cy;
        }
        rcTab.right = rcTab.right * LOWORD(dwDlgBase) / 4;
        rcTab.bottom = rcTab.bottom * HIWORD(dwDlgBase) / 8;

        // Calculate how large to make the tab control, so
        // the display area can accommodate all the child
        // dialog boxes.
        TabCtrl_AdjustRect(pHdr->hwndTab, TRUE, &rcTab);
        OffsetRect(&rcTab, cxMargin - rcTab.left,
                cyMargin - rcTab.top);

        // Calculate the display rectangle.
        CopyRect(&pHdr->rcDisplay, &rcTab);
        TabCtrl_AdjustRect(pHdr->hwndTab, FALSE,
&pHdr->rcDisplay);

        // Set the size and position of the tab control,
        // buttons, and dialog box.
        SetWindowPos(pHdr->hwndTab, NULL, rcTab.left,
rcTab.top,
                rcTab.right - rcTab.left, rcTab.bottom -
rcTab.top,
                SWP_NOZORDER);

        // Move the first button below the tab control.
        hwndButton = GetDlgItem(hwndDlg, BTN_CLOSE);
        SetWindowPos(hwndButton, NULL,
                rcTab.left, rcTab.bottom + cyMargin, 0, 0,
                SWP_NOSIZE | SWP_NOZORDER);

        // Determine the size of the button.
        GetWindowRect(hwndButton, &rcButton);
        rcButton.right -= rcButton.left;
        rcButton.bottom -= rcButton.top;

        // Move the second button to the right of the first.
        hwndButton = GetDlgItem(hwndDlg, BTN_TEST);
        SetWindowPos(hwndButton, NULL,
            rcTab.left + rcButton.right + cxMargin,
            rcTab.bottom + cyMargin, 0, 0,
            SWP_NOSIZE | SWP_NOZORDER);

        // Size the dialog box.
```

(continued)

(continued)

```
    SetWindowPos(hwndDlg, NULL, 0, 0,
        rcTab.right + cyMargin +
        2 * GetSystemMetrics(SM_CXDLGFRAME),
        rcTab.bottom + rcButton.bottom + 2 * cyMargin +
        2 * GetSystemMetrics(SM_CYDLGFRAME) +
        GetSystemMetrics(SM_CYCAPTION),
        SWP_NOMOVE | SWP_NOZORDER);

    // Simulate selection of the first item.
    OnSelChanged(hwndDlg);
    }

        // DoLockDlgRes - loads and locks a dialog
        // template resource.
        // Returns the address of the locked resource.
        // lpszResName - name of the resource

    DLGTEMPLATE * WINAPI DoLockDlgRes(LPCSTR lpszResName)
    {
    HRSRC hrsrc = FindResource(NULL, lpszResName,
RT_DIALOG);
    HGLOBAL hglb = LoadResource(g_hinst, hrsrc);
    return (DLGTEMPLATE *) LockResource(hglb);
    }
```

The following function processes the **TCN_SELCHANGE** notification message for the main dialog box. The function destroys the dialog box for the outgoing page, if any. Then it uses the **CreateDialogIndirect** function to create a modeless dialog box for the incoming page.

```
        // OnSelChanged - processes the TCN_SELCHANGE
        //     notification.
        // hwndDlg - handle to the parent dialog box.

    VOID WINAPI OnSelChanged(HWND hwndDlg)
    {
        DLGHDR *pHdr = (DLGHDR *) GetWindowLong(
        hwndDlg, GWL_USERDATA);
    int iSel = TabCtrl_GetCurSel(pHdr->hwndTab);

    // Destroy the current child dialog box, if any.
    if (pHdr->hwndDisplay != NULL)
        DestroyWindow(pHdr->hwndDisplay);
```

```
    // Create the new child dialog box.
    pHdr->hwndDisplay = CreateDialogIndirect(g_hinst,
        pHdr->apRes[iSel], hwndDlg, ChildDialogProc);
}
```

The following function processes the WM_INITDIALOG message for each of the child dialog boxes. You cannot specify the position of a dialog box created using the **CreateDialogIndirect** function. This function uses the **SetWindowPos** function to position the child dialog within the tab control's display area.

```
    // OnChildDialogInit - Positions the child dialog
    //     box to fall within the display area of the
    //     tab control.

VOID WINAPI OnChildDialogInit(HWND hwndDlg)
{
HWND hwndParent = GetParent(hwndDlg);
DLGHDR *pHdr = (DLGHDR *) GetWindowLong(
    hwndParent, GWL_USERDATA);
SetWindowPos(hwndDlg, HWND_TOP,
    pHdr->rcDisplay.left, pHdr->rcDisplay.top,
    0, 0, SWP_NOSIZE);
}
```

Tab Control Updates in Internet Explorer

Tab controls in Microsoft Internet Explorer support the following new features.

Item States
Tab control items now support an item state to support the **TCM_DESELECTALL** message. Additionally, the **TCITEM** structure supports item state values. See *Tab Control Item States* for more information.

Extended Styles
Tab controls now support extended styles that allow the controls to have enhanced capabilities. See *Tab Control Extended Styles* for more information.

Structures Renamed
All structures used with tab controls have been renamed to conform to current naming conventions, while maintaining backward compatibility. For example, the **TC_ITEM** structure is now named **TCITEM**.

Tab Control Styles

Current tab control styles are supported, and the following styles have been added:

TCS_BOTTOM	**Version 4.70**. Tabs appear at the bottom of the control. This value equals TCS_RIGHT.
TCS_BUTTONS	Tabs appear as buttons, and no border is drawn around the display area.
TCS_FIXEDWIDTH	All tabs are the same width. This style cannot be combined with the TCS_RIGHTJUSTIFY style.
TCS_FLATBUTTONS	**Version 4.71**. Selected tabs appear as being indented into the background while other tabs appear as being on the same plane as the background. This style only affects tab controls with the TCS_BUTTONS style.
TCS_FOCUSNEVER	The tab control does not receive the input focus when clicked.
TCS_FOCUSONBUTTONDOWN	The tab control receives the input focus when clicked.
TCS_FORCEICONLEFT	Icons are aligned with the left edge of each fixed-width tab. This style can only be used with the TCS_FIXEDWIDTH style.
TCS_FORCELABELLEFT	Labels are aligned with the left edge of each fixed-width tab; that is, the label is displayed immediately to the right of the icon instead of being centered. This style can only be used with the TCS_FIXEDWIDTH style, and it implies the TCS_FORCEICONLEFT style.
TCS_HOTTRACK	**Version 4.70**. Items under the pointer are automatically highlighted. You can check whether or not hot tracking is enabled by calling **SystemParametersInfo**.
TCS_MULTILINE	Multiple rows of tabs are displayed, if necessary, so all tabs are visible at once.
TCS_MULTISELECT	**Version 4.70**. Multiple tabs can be selected by holding down CTRL when clicking. This style must be used with the TCS_BUTTONS style.
TCS_OWNERDRAWFIXED	The parent window is responsible for drawing tabs.
TCS_RAGGEDRIGHT	Rows of tabs will not be stretched to fill the entire width of the control. This style is the default.
TCS_RIGHT	**Version 4.70**. Tabs appear vertically on the right side of controls that use the TCS_VERTICAL style. This value equals TCS_BOTTOM.

TCS_RIGHTJUSTIFY	The width of each tab is increased, if necessary, so that each row of tabs fills the entire width of the tab control. This window style is ignored unless the TCS_MULTILINE style is also specified.
TCS_SCROLLOPPOSITE	**Version 4.70**. Unneeded tabs scroll to the opposite side of the control when a tab is selected.
TCS_SINGLELINE	Only one row of tabs is displayed. The user can scroll to see more tabs, if necessary. This style is the default.
TCS_TABS	Tabs appear as tabs, and a border is drawn around the display area. This style is the default.
TCS_TOOLTIPS	The tab control has a tooltip control associated with it.
TCS_VERTICAL	**Version 4.70**. Tabs appear at the left side of the control, with tab text displayed vertically. This style is valid only when used with the TCS_MULTILINE style. To make tabs appear on the right side of the control, also use the TCS_RIGHT style.

Remarks

The following styles can be modified after the control is created:

- TCS_BOTTOM
- TCS_BUTTONS
- TCS_FIXEDWIDTH
- TCS_FLATBUTTONS
- TCS_FORCEICONLEFT
- TCS_FORCELABELLEFT
- TCS_MULTILINE
- TCS_OWNERDRAWFIXED
- TCS_RAGGEDRIGHT
- TCS_RIGHT
- TCS_VERTICAL

Tab Control Extended Styles

The tab control now supports extended styles. These styles are manipulated using the **TCM_GETEXTENDEDSTYLE** and **TCM_SETEXTENDEDSTYLE** messages and should not be confused with extended window styles which are passed to **CreateWindowEx**.

Value	Description
TCS_EX_FLATSEPARATORS	**Version 4.71**. The tab control will draw separators between the tab items. This extended style only affects tab controls that have the **TCS_BUTTONS** and **TCS_FLATBUTTONS** styles. By default, creating the tab control with the **TCS_FLATBUTTONS** style sets this extended style. If you do not require separators, you should remove this extended style after creating the control.
TCS_EX_REGISTERDROP	**Version 4.71**. The tab control generates **TCN_GETOBJECT** notification messages to request a drop target object when an object is dragged over the tab items in the control. The application must call **CoInitialize** or **OleInitialize** before setting this style.

Tab Control Item States

Tab control items now support an item state to support the **TCM_DESELECTALL** message. Additionally, the **TCITEM** structure supports item state values.

Value	Description
TCIS_BUTTONPRESSED	**Version 4.70**. The tab control item is selected. This state is only meaningful if the **TCS_BUTTONS** style flag has been set.
TCIS_HIGHLIGHTED	**Version 4.71**. The tab control item is highlighted, and the tab and text are drawn using the current highlight color. When using high-color, this will be a true interpolation, not a dithered color.

Tab Control Reference

Tab Control Messages

TCM_ADJUSTRECT

Calculates a tab control's display area given a window rectangle, or calculates the window rectangle that would correspond to a specified display area. You can send this message explicitly or by using the **TabCtrl_AdjustRect** macro.

```
TCM_ADJUSTRECT
    wParam = (WPARAM) (BOOL) fLarger;
    lParam = (LPARAM) (LPRECT) prc;
```

Parameters

fLarger

Operation to perform. If this parameter is TRUE, *prc* specifies a display rectangle and receives the corresponding window rectangle. If this parameter is FALSE, *prc* specifies a window rectangle and receives the corresponding display area.

prc

Address of a **RECT** structure that specifies the given rectangle and receives the calculated rectangle.

Return Values

No return value.

Remarks

This message only applies to tab controls that are at the top. It does not apply to tab controls that are on the sides or bottom.


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 commctrl.h.

TCM_DELETEALLITEMS

Removes all items from a tab control. You can send this message explicitly or by using the **TabCtrl_DeleteAllItems** macro.

```
TCM_DELETEALLITEMS
    wParam = 0;
    lParam = 0;
```

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: Requires version 1.0 or later.
Header: Declared in commctrl.h.

TCM_DELETEITEM

Removes an item from a tab control. You can send this message explicitly or by using the **TabCtrl_DeleteItem** macro.

```
TCM_DELETEITEM
    wParam = (WPARAM) (int) iItem;
    lParam = 0;
```

Parameters

iItem
Index of the item to delete.

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: Requires version 1.0 or later.
Header: Declared in commctrl.h.

TCM_DESELECTALL

Resets items in a tab control, clearing any that were set to the **TCIS_BUTTONPRESSED** state. You can send this message explicitly or by using the **TabCtrl_DeselectAll** macro.

```
TCM_DESELECTALL
    wParam = (WPARAM) (DWORD) fExcludeFocus;
    lParam = 0;
```

Parameters

fExcludeFocus

Flag that specifies the scope of the item deselection. If this parameter is set to FALSE, all tab items will be reset. If it is set to TRUE, then all tab items except for the one currently selected will be reset.

Return Values

The return value for this message is not used.

Remarks

This message is only meaningful if the **TCS_BUTTONS** style flag has been set.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

TCM_GETCURFOCUS

Returns the index of the item that has the focus in a tab control. You can send this message explicitly or by using the **TabCtrl_GetCurFocus** macro.

```
TCM_GETCURFOCUS
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the index of the tab item that has the focus.

Remarks

The item that has the focus may be different than the selected item.

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 commctrl.h.

TCM_GETCURSEL

Determines the currently selected tab in a tab control. You can send this message explicitly or by using the **TabCtrl_GetCurSel** macro.

```
TCM_GETCURSEL
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the index of the selected tab if successful, or –1 if no tab is selected.

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 commctrl.h.

TCM_GETEXTENDEDSTYLE

Retrieves the extended styles that are currently in use for the tab control. You can send this message explicitly or by using the **TabCtrl_GetExtendedStyle** macro.

```
TCM_GETEXTENDEDSTYLE
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a DWORD value that represents the extended styles currently in use for the tab control. This value is a combination of tab control extended styles.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

TCM_GETIMAGELIST

Retrieves the image list associated with a tab control. You can send this message explicitly or by using the **TabCtrl_GetImageList** macro.

```
TCM_GETIMAGELIST
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the handle to the image list if successful, or NULL otherwise.

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 commctrl.h.

TCM_GETITEM

Retrieves information about a tab in a tab control. You can send this message explicitly or by using the **TabCtrl_GetItem** macro.

```
TCM_GETITEM
    wParam = (WPARAM) (int) iItem;
    lParam = (LPARAM) (LPTCITEM) pitem;
```

Parameters

iItem
Index of the tab.

pitem
Address of a **TCITEM** structure that specifies the information to retrieve and receives information about the tab. When the message is sent, the **mask** member specifies which attributes to return.

If the **mask** member specifies the TCIF_TEXT value, the **pszText** member must contain the address of the buffer that receives the item text, and the **cchTextMax** member must specify the size of the buffer.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

If the TCIF_TEXT flag is set in the **mask** member of the **TCITEM** structure, the control may change the **pszText** member of the structure to point to the new text instead of filling the buffer with the requested text. The control may set the **pszText** member to NULL to indicate that no text is associated with the item.

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 commctrl.h.

TCM_GETITEMCOUNT

Retrieves the number of tabs in the tab control. You can send this message explicitly or by using the **TabCtrl_GetItemCount** macro.

```
TCM_GETITEMCOUNT
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the number of items if successful, or zero otherwise.

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 commctrl.h.

TCM_GETITEMRECT

Retrieves the bounding rectangle for a tab in a tab control. You can send this message explicitly or by using the **TabCtrl_GetItemRect** macro.

```
TCM_GETITEMRECT
    wParam = (WPARAM) (int) iItem;
    lParam = (LPARAM) (RECT FAR *) prc;
```

Parameters

iItem
Index of the tab.

prc
Address of a **RECT** structure that receives the bounding rectangle of the tab, in viewport coordinates.

Return Values

Returns TRUE if successful, or FALSE otherwise.

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 commctrl.h.

TCM_GETROWCOUNT

Retrieves the current number of rows of tabs in a tab control. You can send this message explicitly or by using the **TabCtrl_GetRowCount** macro.

```
TCM_GETROWCOUNT
   wParam = 0;
   lParam = 0;
```

Return Values

Returns the number of rows of tabs.

Remarks

Only tab controls that have the **TCS_MULTILINE** style can have multiple rows of tabs.

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 commctrl.h.

TCM_GETTOOLTIPS

Retrieves the handle to the tooltip control associated with a tab control. You can send this message explicitly or by using the **TabCtrl_GetToolTips** macro.

```
TCM_GETTOOLTIPS
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the handle to the tooltip control if successful, or NULL otherwise.

Remarks

A tab control creates a tooltip control if it has the **TCS_TOOLTIPS** style. You can also assign a tooltip control to a tab control by using the **TCM_SETTOOLTIPS** 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 commctrl.h.

TCM_GETUNICODEFORMAT

Retrieves the UNICODE character format flag for the control. You can send this message explicitly or use the **TabCtrl_GetUnicodeFormat** macro.

```
TCM_GETUNICODEFORMAT
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the UNICODE format flag for the control. If this value is nonzero, the control is using UNICODE characters. If this value is zero, the control is using ANSI characters.

Remarks

See the remarks for **CCM_GETUNICODEFORMAT** for a discussion of this message.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

See Also

TCM_SETUNICODEFORMAT

TCM_HIGHLIGHTITEM

Sets the highlight state of a tab item. You can send this message explicitly or by using the **TabCtrl_HighlightItem** macro.

```
TCM_HIGHLIGHTITEM
    wParam = (WPARAM) idItem;
    lParam = (LPARAM) MAKELONG(fHighlight,0);
```

Parameters

idItem

Zero-based index of a tab control item.

fHighlight

Value specifying the highlight state to be set. If this value is TRUE, the tab is highlighted; if FALSE, the tab is set to its default state.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

TCM_HITTEST

Determines which tab, if any, is at a specified screen position. You can send this message explicitly or by using the **TabCtrl_HitTest** macro.

```
TCM_HITTEST
    wParam = 0;
    lParam = (LPARAM) (LPTCHITTESTINFO) pinfo;
```

Parameters

pinfo
Address of a **TCHITTESTINFO** structure that specifies the screen position to test.

Return Values

Returns the index of the tab, or –1 if no tab is at the specified position.

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 commctrl.h.

TCM_INSERTITEM

Inserts a new tab in a tab control. You can send this message explicitly or by using the **TabCtrl_InsertItem** macro.

```
TCM_INSERTITEM
   wParam = (WPARAM) (int) iItem;
   lParam = (LPARAM) (const LPTCITEM) pitem;
```

Parameters

iItem
Index of the new tab.

pitem
Address of a **TCITEM** structure that specifies the attributes of the tab. The **dwState** and **dwStateMask** members of this structure are ignored by this message.

Return Values

Returns the index of the new tab if successful, or –1 otherwise.

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 commctrl.h.

TCM_REMOVEIMAGE

Removes an image from a tab control's image list. You can send this message explicitly or by using the **TabCtrl_RemoveImage** macro.

```
TCM_REMOVEIMAGE
    wParam = (WPARAM) (int) iImage;
    lParam = 0;
```

Parameters

iImage
Index of the image to remove.

Return Values

No return value.

Remarks

The tab control updates each tab's image index, so each tab remains associated with the same image as before. If a tab is using the image being removed, the tab will be set to have no image.

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 commctrl.h.

TCM_SETCURFOCUS

Sets the focus to a specified tab in a tab control. You can send this message explicitly or by using the **TabCtrl_SetCurFocus** macro.

```
TCM_SETCURFOCUS
    wParam = (WPARAM) (int) iItem;
    lParam = 0;
```

Parameters

iItem
Index of the tab that gets the focus.

Return Values

No return value.

Remarks

If the tab control has the **TCS_BUTTONS** style (button mode), the tab with the focus may be different from the selected tab. For example, when a tab is selected, the user can press the arrow keys to set the focus to a different tab without changing the selected tab. In button mode, TCM_SETCURFOCUS sets the input focus to the button associated with the specified tab, but it does not change the selected tab.

If the tab control does not have the **TCS_BUTTONS** style, changing the focus also changes the selected tab. In this case, the tab control sends the **TCN_SELCHANGING** and **TCN_SELCHANGE** notification messages to its parent 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 commctrl.h.

See Also

TCM_GETCURFOCUS

TCM_SETCURSEL

Selects a tab in a tab control. You can send this message explicitly or by using the **TabCtrl_SetCurSel** macro.

```
TCM_SETCURSEL
    wParam = (WPARAM) (int) iItem;
    lParam = 0;
```

Parameters

iItem
Index of the tab to select.

Return Values

Returns the index of the previously selected tab if successful, or –1 otherwise.

Remarks

A tab control does not send a **TCN_SELCHANGING** or **TCN_SELCHANGE** notification message when a tab is selected using this message.

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 commctrl.h.

TCM_SETEXTENDEDSTYLE

Sets the extended styles that the tab control will use. You can send this message explicitly or by using the **TabCtrl_SetExtendedStyle** macro.

```
TCM_SETEXTENDEDSTYLE
    wParam = (WPARAM)dwExMask;
    lParam = (LPARAM)dwExStyle;
```

Parameters

dwExMask
A DWORD value that indicates which styles in *dwExStyle* are to be affected. Only the extended styles in *dwExMask* will be changed. All other styles will be maintained as they are. If this parameter is zero, then all of the styles in *dwExStyle* will be affected.

dwExStyle
Value specifying the extended tab control styles. This value is a combination of tab control extended styles.

Return Values

Returns a DWORD value that contains the previous tab control extended styles.

Remarks

The *dwExMask* parameter allows you to modify one or more extended styles without having to retrieve the existing styles first. For example, if you pass **TCS_EX_FLATSEPARATORS** for *dwExMask* and 0 for *dwExStyle*, the **TCS_EX_FLATSEPARATORS** style will be cleared, but all other styles will remain the same.

For backward compatibility reasons, the **TabCtrl_SetExtendedStyle** macro has not been updated to use *dwExMask*.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

TCM_SETIMAGELIST

Assigns an image list to a tab control. You can send this message explicitly or by using the **TabCtrl_SetImageList** macro.

```
TCM_SETIMAGELIST
    wParam = 0;
    lParam = (LPARAM) (HIMAGELIST) himl;
```

Parameters

himl
Handle to the image list to assign to the tab control.

Return Values

Returns the handle to the previous image list, or NULL if there is no previous image list.

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 commctrl.h.

TCM_SETITEM

Sets some or all of a tab's attributes. You can send this message explicitly or by using the **TabCtrl_SetItem** macro.

```
TCM_SETITEM
    wParam = (WPARAM) (int) iItem;
    lParam = (LPARAM) (LPTCITEM) pitem;
```

Parameters

iItem
Index of the item.

pitem
Address of a **TCITEM** structure that contains the new item attributes. The **mask** member specifies which attributes to set.

If the **mask** member specifies the LVIF_TEXT value, the **pszText** member is the address of a null-terminated string and the **cchTextMax** member is 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: Requires version 1.0 or later.
Header: Declared in commctrl.h.

TCM_SETITEMEXTRA

Sets the number of bytes per tab reserved for application-defined data in a tab control. You can send this message explicitly or by using the **TabCtrl_SetItemExtra** macro.

```
TCM_SETITEMEXTRA
    wParam = (WPARAM) (int) cb;
    lParam = 0;
```

Parameters

cb
Number of extra bytes.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

By default, the number of extra bytes is four. An application that changes the number of extra bytes cannot use the **TCITEM** structure to retrieve and set the application-defined data for a tab. Instead, you must define a new structure that consists of the **TCITEMHEADER** structure followed by application-defined members.

An application should only change the number of extra bytes when a tab control does not contain any tabs.

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 commctrl.h.

TCM_SETITEMSIZE

Sets the width and height of tabs in a fixed-width or owner-drawn tab control. You can send this message explicitly or by using the **TabCtrl_SetItemSize** macro.

```
TCM_SETITEMSIZE
    wParam = 0;
    lParam = MAKELPARAM(cx, cy);
```

Parameters

cx and *cy*
New width and height, in pixels.

Return Values

Returns the old width and height. The width is in the low-order word of the return value, and the height is in the high-order word.

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 commctrl.h.

TCM_SETMINTABWIDTH

Sets the minimum width of items in a tab control. You can send this message explicitly or by using the **TabCtrl_SetMinTabWidth** macro.

```
TCM_SETMINTABWIDTH
    wParam = 0;
    lParam = (LPARAM) (INT) cx;
```

Parameters

cx
Minimum width to be set for a tab control item. If this parameter is set to –1, the control will use the default tab width.

Return Values

Returns an INT value that represents the previous minimum tab width.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

TCM_SETPADDING

Sets the amount of space (padding) around each tab's icon and label in a tab control. You can send this message explicitly or by using the **TabCtrl_SetPadding** macro.

```
TCM_SETPADDING
    wParam = 0;
    lParam = MAKELPARAM(cx, cy);
```

Parameters

cx and *cy*
Amount of horizontal and vertical padding, in pixels.

Return Values

No return value.

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 commctrl.h.

TCM_SETTOOLTIPS

Assigns a tooltip control to a tab control. You can send this message explicitly or by using the **TabCtrl_SetToolTips** macro.

```
TCM_SETTOOLTIPS
    wParam = (WPARAM) (HWND) hwndTT;
    lParam = 0;
```

Parameters

hwndTT
Handle to the tooltip control.

Return Values

No return value.

Remarks

You can get the tooltip control associated with a tab control by using the **TCM_GETTOOLTIPS** 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 commctrl.h.

TCM_SETUNICODEFORMAT

Sets the UNICODE character format flag for the control. This message allows you to change the character set used by the control at run time rather than having to re-create the control. You can send this message explicitly or use the **TabCtrl_SetUnicodeFormat** macro.

```
TCM_SETUNICODEFORMAT
    wParam = (WPARAM)(BOOL)fUnicode;
    lParam = 0;
```

Parameters

fUnicode

Determines the character set that is used by the control. If this value is nonzero, the control will use UNICODE characters. If this value is zero, the control will use ANSI characters.

Return Values

Returns the previous UNICODE format flag for the control.

Remarks

See the remarks for **CCM_SETUNICODEFORMAT** for a discussion of this message.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

See Also

TCM_GETUNICODEFORMAT

Tab Control Macros

TabCtrl_AdjustRect

Calculates a tab control's display area given a window rectangle, or calculates the window rectangle that would correspond to a specified display area. You can use this macro or send the **TCM_ADJUSTRECT** message explicitly.

```
VOID TabCtrl_AdjustRect(
    HWND hwnd,
    BOOL fLarger,
    RECT FAR *prc
);
```

Parameters

hwnd
Handle to the tab control.

fLarger
Operation to perform. If this parameter is TRUE, *prc* specifies a display rectangle and receives the corresponding window rectangle. If this parameter is FALSE, *prc* specifies a window rectangle and receives the corresponding display area.

prc
Address of a **RECT** structure that specifies the given rectangle and receives the calculated rectangle.

Return Values

No return value.

Remarks

This message only applies to tab controls that are at the top. It does not apply to tab controls that are on the sides or bottom.

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 commctrl.h.

TabCtrl_DeleteAllItems

Removes all items from a tab control. You can use this macro or send the **TCM_DELETEALLITEMS** message explicitly.

```
BOOL TabCtrl_DeleteAllItems(
    HWND hwnd
);
```

Parameters

hwnd
Handle to the tab control.

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: Requires version 1.0 or later.
Header: Declared in commctrl.h.

TabCtrl_DeleteItem

Removes an item from a tab control. You can use this macro or send the **TCM_DELETEITEM** message explicitly.

```
BOOL TabCtrl_DeleteItem(
    HWND hwnd,
    int iItem
);
```

Parameters

hwnd
Handle to the tab control.

iItem
Index of the item to delete.

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: Requires version 1.0 or later.
Header: Declared in commctrl.h.

TabCtrl_DeselectAll

Resets items in a tab control, clearing any that were set to the **TCIS_BUTTONPRESSED** state. You can use this macro or send the **TCM_DESELECTALL** message explicitly.

```
void TabCtrl_DeselectAll(
    HWND hwndTab,
    UINT fExcludeFocus
);
```

Parameters

hwndTab
Handle to the tab control.

fExcludeFocus
Flag value that specifies the scope of the item deselection. If this parameter is set to FALSE, all tab items will be reset. If it is set to TRUE, all but the currently selected tab item will be reset.

Return Values

The return value is not used.

Remarks

This message is only meaningful if the **TCS_BUTTONS** style flag has been set.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).

Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

TabCtrl_GetCurFocus

Returns the index of the item that has the focus in a tab control. You can use this macro or send the **TCM_GETCURFOCUS** message explicitly.

```
int TabCtrl_GetCurFocus(
    HWND hwnd
);
```

Parameters

hwnd
Handle to the tab control.

Return Values

Returns the index of the tab item that has the focus.

Remarks

The item that has the focus may be different than the selected item.

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 commctrl.h.

TabCtrl_GetCurSel

Determines the currently selected tab in a tab control. You can use this macro or send the **TCM_GETCURSEL** message explicitly.

```
int TabCtrl_GetCurSel(
    HWND hwnd
);
```

Parameters

hwnd
Handle to the tab control.

Return Values

Returns the index of the selected tab if successful, or –1 if no tab is selected.

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 commctrl.h.

TabCtrl_GetExtendedStyle

Retrieves the extended styles that are currently in use for the tab control. You can use this macro or send the **TCM_GETEXTENDEDSTYLE** message explicitly.

```
DWORD TabCtrl_GetExtendedStyle(
    HWND hwndTab
);
```

Parameters

hwndTab
Handle to the tab control.

Return Values

Returns a DWORD value that represents the extended styles currently in use for the tab control. This value is a combination of tab control extended styles.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

TabCtrl_GetImageList

Retrieves the image list associated with a tab control. You can use this macro or send the **TCM_GETIMAGELIST** message explicitly.

```
HIMAGELIST TabCtrl_GetImageList(
    HWND hwnd
);
```

Return Values

Returns the handle to the image list if successful, or NULL otherwise.

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 commctrl.h.

TabCtrl_GetItem

Retrieves information about a tab in a tab control. You can use this macro or send the **TCM_GETITEM** message explicitly.

```
BOOL TabCtrl_GetItem(
    HWND hwnd,
    int iItem ,
    LPTCITEM pitem
);
```

Parameters

hwnd
Handle to the tab control.

iItem
Index of the tab.

pitem
Address of a **TCITEM** structure that specifies the information to retrieve and receives information about the tab. When the message is sent, the **mask** member specifies which attributes to return.

If the **mask** member specifies the TCIF_TEXT value, the **pszText** member must contain the address of the buffer that receives the item text, and the **cchTextMax** member must specify the size of the buffer.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

If the TCIF_TEXT flag is set in the **mask** member of the **TCITEM** structure, the control may change the **pszText** member of the structure to point to the new text instead of

filling the buffer with the requested text. The control may set the **pszText** member to NULL to indicate that no text is associated with the item.

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 commctrl.h.

TabCtrl_GetItemCount

Retrieves the number of tabs in the tab control. You can use this macro or send the **TCM_GETITEMCOUNT** message explicitly.

```
int TabCtrl_GetItemCount(
    HWND hwnd
);
```

Parameters

hwnd
Handle to the tab control.

Return Values

Returns the number of items if successful, or zero otherwise.

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 commctrl.h.

TabCtrl_GetItemRect

Retrieves the bounding rectangle for a tab in a tab control. You can use this macro or send the **TCM_GETITEMRECT** message explicitly.

```
BOOL TabCtrl_GetItemRect(
    HWND hwnd,
    int iItem,
    RECT FAR *prc
);
```

Parameters

hwnd
Handle to the tab control.

iItem
Index of the tab.

prc
Address of a **RECT** structure that receives the bounding rectangle of the tab, in viewport coordinates.

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: Requires version 1.0 or later.
Header: Declared in commctrl.h.

TabCtrl_GetRowCount

Retrieves the current number of rows of tabs in a tab control. You can use this macro or send the **TCM_GETROWCOUNT** message explicitly.

```
int TabCtrl_GetRowCount(
    HWND hwnd
);
```

Parameters

hwnd
Handle to the tab control.

Return Values

Returns the number of rows of tabs.

Remarks

Only tab controls that have the TCS_MULTILINE style can have multiple rows of tabs.

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 commctrl.h.

TabCtrl_GetToolTips

Retrieves the handle to the tooltip control associated with a tab control. You can use this macro or send the **TCM_GETTOOLTIPS** message explicitly.

```
int TabCtrl_GetToolTips(
    HWND hwnd
);
```

Parameters

hwnd
Handle to the tab control.

Return Values

Returns the handle to the tooltip control if successful, or NULL otherwise.

Remarks

A tab control creates a tooltip control if it has the **TCS_TOOLTIPS** style. You can also assign a tooltip control to a tab control by using the **TCM_SETTOOLTIPS** 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 commctrl.h.

TabCtrl_GetUnicodeFormat

Retrieves the UNICODE character format flag for the control. You can use this macro or send the **TCM_GETUNICODEFORMAT** message explicitly.

```
BOOL TabCtrl_GetUnicodeFormat(
    HWND hwnd
);
```

Parameters

hwnd
Handle to the control.

Return Values

Returns the UNICODE format flag for the control. If this value is nonzero, the control is using UNICODE characters. If this value is zero, the control is using ANSI characters.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

See Also

TabCtrl_SetUnicodeFormat

TabCtrl_HighlightItem

Sets the highlight state of a tab item. You can use this macro or send the **TCM_HIGHLIGHTITEM** message explicitly.

```
BOOL TabCtrl_HighlightItem(
    HWND hwndTab,
    INT idItem,
    WORD fHighlight
);
```

Parameters

hwndTab
Handle to the tab control.

idItem
Zero-based index of a tab control item.

fHighlight
Value specifying the highlight state to be set. If this value is nonzero, the tab is highlighted. If this value is zero, the tab is set to its default state.

Return Values

Returns nonzero if successful, or zero otherwise.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

TabCtrl_HitTest

Determines which tab, if any, is at a specified screen position. You can use this macro or send the **TCM_HITTEST** message explicitly.

```
int TabCtrl_HitTest(
    HWND hwnd,
    LPTCHITTESTINFO pinfo
);
```

Parameters

hwnd
Handle to the tab control.

pinfo
Address of a TCHITTESTINFO structure that specifies the screen position to test.

Return Values

Returns the index of the tab, or –1 if no tab is at the specified position.

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 commctrl.h.

TabCtrl_InsertItem

Inserts a new tab in a tab control. You can use this macro or send the **TCM_INSERTITEM** message explicitly.

```
int TabCtrl_InsertItem(
    HWND hwnd,
    int iItem,
    const LPTCITEM pitem
);
```

Parameters

hwnd
Handle to the tab control.

iItem
Index of the new tab.

pitem
Address of a **TCITEM** structure that specifies the attributes of the tab. The **dwState** and **dwStateMask** members of this structure are ignored by this message.

Return Values

Returns the index of the new tab if successful, or –1 otherwise.

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 commctrl.h.

TabCtrl_RemoveImage

Removes an image from a tab control's image list. You can use this macro or send the **TCM_REMOVEIMAGE** message explicitly.

```
void TabCtrl_RemoveImage(
    HWND hwnd,
    int iImage
);
```

Parameters

hwnd
Handle to the tab control.

iImage
Index of the image to remove.

Return Values

No return value.

Remarks

The tab control updates each tab's image index, so each tab remains associated with the same image as before. If a tab is using the image being removed, the tab will be set to have no image.

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 commctrl.h.

TabCtrl_SetCurFocus

Sets the focus to a specified tab in a tab control. You can use this macro or send the **TCM_SETCURFOCUS** message explicitly.

```
VOID TabCtrl_SetCurFocus(
    HWND hwnd,
    int iItem
);
```

Parameters

hwnd
Handle to the tab control.

iItem
Zero-based index of the tab that gets the focus.

Return Values

No return value.

Remarks

If the tab control has the **TCS_BUTTONS** style (button mode), the tab with the focus may be different from the selected tab. For example, when a tab is selected, the user can press the arrow keys to set the focus to a different tab without changing the selected tab. In button mode, the **TabCtrl_SetCurFocus** macro sets the input focus to the button associated with the specified tab, but it does not change the selected tab.

If the tab control does not have the **TCS_BUTTONS** style, changing the focus also changes the selected tab. In this case, the tab control sends the **TCN_SELCHANGING** and **TCN_SELCHANGE** notification messages to its parent 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 commctrl.h.

See Also

TabCtrl_GetCurFocus, TCM_GETCURFOCUS

TabCtrl_SetCurSel

Selects a tab in a tab control. You can use this macro or send the **TCM_SETCURSEL** message explicitly.

```
int TabCtrl_SetCurSel(
    HWND hwnd,
    int iItem
);
```

Parameters

hwnd
Handle to the tab control.

iItem
Index of the tab to select.

Return Values

Returns the index of the previously selected tab if successful, or –1 otherwise.

Remarks

A tab control does not send a **TCN_SELCHANGING** or **TCN_SELCHANGE** notification message when a tab is selected using the **TCM_SETCURSEL** message.

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 commctrl.h.

TabCtrl_SetExtendedStyle

Sets the extended styles that the tab control will use. You can use this macro or send the **TCM_SETEXTENDEDSTYLE** message explicitly.

```
DWORD TabCtrl_SetExtendedStyle(
    HWND hwndTab,
    DWORD dwExStyle
);
```

Parameters

hwndTab
Handle to the tab control.

dwExStyle
Value that contains the new tab control extended styles. This value is a combination of tab control **extended styles**.

Return Values

Returns a DWORD value that contains the previous tab control extended styles.

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

TabCtrl_SetImageList

Assigns an image list to a tab control. You can use this macro or send the **TCM_SETIMAGELIST** message explicitly.

```
BOOL TabCtrl_SetImageList(
    HWND hwnd,
    HIMAGELIST himl
);
```

Parameters

hwnd
Handle to the tab control.

himl
Handle to the image list to assign to the tab control.

Return Values

Returns the handle to the previous image list, or NULL if there is no previous image list.

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 commctrl.h.

TabCtrl_SetItem

Sets some or all of a tab's attributes. You can use this macro or send the **TCM_SETITEM** message explicitly.

```
BOOL TabCtrl_SetItem(
    HWND hwnd,
    int iItem,
    LPTCITEM pitem
);
```

Parameters

hwnd
Handle to the tab control.

iItem
Index of the item.

pitem
Address of a **TCITEM** structure that contains the new item attributes. The **mask** member specifies which attributes to set.

If the **mask** member specifies the LVIF_TEXT value, the **pszText** member is the address of a null-terminated string and the **cchTextMax** member is 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: Requires version 1.0 or later.
Header: Declared in commctrl.h.

TabCtrl_SetItemExtra

Sets the number of bytes per tab reserved for application-defined data in a tab control. You can use this macro or send the **TCM_SETITEMEXTRA** message explicitly.

```
BOOL TabCtrl_SetItemExtra(
    HWND hwnd,
    int cb
);
```

Parameters

hwnd
Handle to the tab control.

cb
Number of extra bytes.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Remarks

By default, the number of extra bytes is four. An application that changes the number of extra bytes cannot use the **TCITEM** structure to retrieve and set the application-defined data for a tab. Instead, you must define a new structure that consists of the **TCITEMHEADER** structure followed by application-defined members.

An application should only change the number of extra bytes when a tab control does not contain any tabs.

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 commctrl.h.

TabCtrl_SetItemSize

Sets the width and height of tabs in a fixed-width or owner-drawn tab control. You can use this macro or send the **TCM_SETITEMSIZE** message explicitly.

```
DWORD TabCtrl_SetItemSize(
    HWND hwnd,
    int cx,
    int cy
);
```

Parameters

hwnd
Handle to the tab control.

cx and *cy*
New width and height, in pixels.

Return Values

Returns the old width and height. The width is in the low-order word of the return value, and the height is in the high-order word.

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 commctrl.h.

TabCtrl_SetMinTabWidth

Sets the minimum width of items in a tab control. You can use this macro or send the **TCM_SETMINTABWIDTH** message explicitly.

```
int TabCtrl_SetMinTabWidth(
    HWND hwndTab,
    INT cx
);
```

Parameters

hwndTab
Handle to the tab control.

cx
Minimum width to be set for a tab control item. If this parameter is set to –1, the control will use the default tab width.

Return Values

Returns an INT value that represents the previous minimum tab width.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

TabCtrl_SetPadding

Sets the amount of space (padding) around each tab's icon and label in a tab control. You can use this macro or send the **TCM_SETPADDING** message explicitly.

```
void TabCtrl_SetPadding(
    HWND hwnd,
    int cx,
    int cy
);
```

Parameters

hwnd
Handle to the tab control.

cx and *cy*
Amount of horizontal and vertical padding, in pixels.

Return Values

No return value.

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 commctrl.h.

TabCtrl_SetToolTips

Assigns a tooltip control to a tab control. You can use this macro or send the **TCM_SETTOOLTIPS** message explicitly.

```
void TabCtrl_SetToolTips(
    HWND hwndTab,
    HWND hwndTT
);
```

Parameters

hwndTab
Handle to the tab control.

hwndTT
Handle to the tooltip control.

Return Values

No return value.

Remarks

You can get the tooltip control associated with a tab control by using the **TCM_GETTOOLTIPS** 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 commctrl.h.

TabCtrl_SetUnicodeFormat

Sets the UNICODE character format flag for the control. This message allows you to change the character set used by the control at run time rather than having to re-create the control. You can use this macro or send the **TCM_SETUNICODEFORMAT** message explicitly.

```
BOOL TabCtrl_SetUnicodeFormat(
    HWND hwnd,
    BOOL fUnicode
);
```

Parameters

hwnd
Handle to the control.

fUnicode
Determines the character set that is used by the control. If this value is nonzero, the control will use UNICODE characters. If this value is zero, the control will use ANSI characters.

Return Values

Returns the previous UNICODE format flag for the control.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

See Also

TabCtrl_GetUnicodeFormat

Tab Control Notification Messages

NM_CLICK (tab)

Notifies a tab control's parent window that the user has clicked the left mouse button within the control. NM_CLICK is sent in the form of a **WM_NOTIFY** message.

```
NM_CLICK
    lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh
Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The return value is ignored by the tab control.

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 commctrl.h.

NM_RCLICK (tab)

Notifies a tab control's parent window that the user has clicked the right mouse button within the control. NM_RCLICK is sent in the form of a **WM_NOTIFY** message.

```
NM_RCLICK
    lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh
Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The return value is ignored by the tab control.

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in commctrl.h.

NM_RELEASEDCAPTURE (tab)

Notifies a tab control's parent window that the control is releasing mouse capture. This notification is sent in the form of a **WM_NOTIFY** message.

```
NM_RELEASEDCAPTURE
    lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh

Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The control ignores the return value from this notification.

Version 4.71 and later of Comctl32.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 commctrl.h.

TCN_FOCUSCHANGE

Notifies a tab control's parent window that the button focus has changed.

```
TCN_FOCUSCHANGE
```

Parameters
None

Return Values
No return value.

Requirements

Version 5.80 and later of Comctl32.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 commctrl.h.
Import Library: comctl32.lib.

TCN_GETOBJECT

Sent by a tab control when it has the **TCS_EX_REGISTERDROP** extended style and an object is dragged over a tab item in the control. This notification message is sent in the form of a **WM_NOTIFY** message.

```
TCN_GETOBJECT
    lpnmon = (LPNMOBJECTNOTIFY) lParam;
```

Parameters
lpnmon
Address of an **NMOBJECTNOTIFY** structure that contains information about the tab item the object is dragged over and receives data the application returns in response to this message.

Return Values
The application processing this notification must return zero.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

TCN_KEYDOWN

Notifies a tab control's parent window that a key has been pressed. This message is sent in the form of a **WM_NOTIFY** message.

```
TCN_KEYDOWN
   pnm = (NMTCKEYDOWN FAR *) lParam;
```

Parameters

pnm
Address of an **NMTCKEYDOWN** structure.

Return Values

No return value.

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 commctrl.h.

TCN_SELCHANGE

Notifies a tab control's parent window that the currently selected tab has changed. This message is sent in the form of a **WM_NOTIFY** message.

```
TCN_SELCHANGE
   lpnmhdr = (LPNMHDR) lParam;
```

Parameters

lpnmhdr
Address of an **NMHDR** structure. The **hwndFrom** member is the handle to the tab control. The **idFrom** member is the child window identifier of the tab control. The **code** member is TCN_SELCHANGE.

Return Values

No return value.

Remarks

To determine the currently selected tab, use the **TabCtrl_GetCurSel** macro.

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 commctrl.h.

See Also

TCN_SELCHANGING

TCN_SELCHANGING

Notifies a tab control's parent window that the currently selected tab is about to change. This message is sent in the form of a **WM_NOTIFY** message.

```
TCN_SELCHANGING
    lpnmhdr = (LPNMHDR) lParam;
```

Parameters

lpnmhdr
Address of an **NMHDR** structure. The **hwndFrom** member is the handle to the tab control. The **idFrom** member is the child window identifier of the tab control. The **code** member is TCN_SELCHANGING.

Return Values

Returns TRUE to prevent the selection from changing, or FALSE to allow the selection to change.

Remarks

To determine the currently selected tab, use the **TabCtrl_GetCurSel** macro.

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 commctrl.h.

See Also

TCN_SELCHANGE

Tab Control Structures

NMTCKEYDOWN

Contains information about a key press in a tab control. It is used with the **TCN_KEYDOWN** notification message. This structure supersedes the **TC_KEYDOWN** structure.

```
typedef struct tagNMTCKEYDOWN {
    NMHDR hdr;
    WORD wVKey;
    UINT flags;
} NMTCKEYDOWN;
```

Members

hdr
NMHDR structure that contains information about the notification message.

wVKey
Virtual key code.

flags
Value that is identical to the *lParam* parameter of the **WM_KEYDOWN** message.

Requirements

Version 4.71 and later of Comctl32.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: Requires version 2.0 or later.
Header: Declared in commctrl.h.

TCHITTESTINFO

Contains information about a hit test. This structure supersedes the **TC_HITTESTINFO** structure.

```
typedef struct tagTCHITTESTINFO {
    POINT pt;
    UINT flags;
} TCHITTESTINFO, FAR * LPTCHITTESTINFO;
```

Members

pt

Position to hit test, in client coordinates.

flags

Variable that receives the results of a hit test. The tab control sets this member to one of the following values:

TCHT_NOWHERE	The position is not over a tab.
TCHT_ONITEM	The position is over a tab but not over its icon or its text. For owner-drawn tab controls, this value is specified if the position is anywhere over a tab.
TCHT_ONITEMICON	The position is over a tab's icon.
TCHT_ONITEMLABEL	The position is over a tab's text.
	TCHT_ONITEM is a bitwise-OR operation on TCHT_ONITEMICON and TCHT_ONITEMLABEL.

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 commctrl.h.

See Also

TCM_HITTEST

TCITEM

Specifies or receives the attributes of a tab item. It is used with the **TCM_INSERTITEM**, **TCM_GETITEM**, and **TCM_SETITEM** messages. This structure supersedes the **TC_ITEM** structure.

```
typedef struct tagTCITEM {
    UINT mask;
#if (_WIN32_IE >= 0x0300)
    DWORD dwState;
    DWORD dwStateMask;
#else
    UINT lpReserved1;
    UINT lpReserved2;
#endif
    LPTSTR pszText;
```

(continued)

(continued)

```
    int cchTextMax;
    int iImage;
    LPARAM lParam;
} TCITEM, FAR *LPTCITEM;
```

Members

mask

Value that specifies which members to retrieve or set. This member can be a combination of the following values:

TCIF_IMAGE	The iImage member is valid.
TCIF_PARAM	The lParam member is valid.
TCIF_RTLREADING	The string pointed to by pszText will be displayed in the opposite direction to the text in the parent window.
TCIF_STATE	Version 4.70. The dwState member is valid.
TCIF_TEXT	The pszText member is valid.

dwState

Version 4.70. Specifies the item's current state if information is being retrieved. If item information is being set, this member contains the state value to be set for the item. For a list of valid tab control item states, see *Tab Control Item States*. This member is ignored in the **TCM_INSERTITEM** message.

dwStateMask

Version 4.70. Specifies which bits of the **dwState** member contain valid information. This member is ignored in the **TCM_INSERTITEM** message.

lpReserved1

Version 4.00. Not used.

lpReserved2

Version 4.00. Not used.

pszText

Address of a null-terminated string that contains the tab text when item information is being set. If item information is being retrieved, this member specifies the address of the buffer that receives the tab text.

cchTextMax

Size of the buffer pointed to by the **pszText** member. If the structure is not receiving information, this member is ignored.

iImage

Index in the tab control's image list, or –1 if there is no image for the tab.

lParam

Application-defined data associated with the tab control item. If more or less than 4 bytes of application-defined data exist per tab, an application must define a structure

and use it instead of the **TCITEM** structure. The first member of the application-defined structure must be a **TCITEMHEADER** structure.

Remarks

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, **pszText** will be displayed in same direction as the text in its parent window. If TCIF_RTLREADING is set, **pszText** will read in the opposite direction from the text in the parent window.

Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

TCITEMHEADER

Specifies or receives the attributes of a tab. It is used with the **TCM_INSERTITEM**, **TCM_GETITEM**, and **TCM_SETITEM** messages. This structure supersedes the **TC_ITEMHEADER** structure.

```
typedef struct tagTCITEMHEADER{
    UINT   mask;
    UINT   lpReserved1;
    UINT   lpReserved2;
    LPTSTR pszText;
    int    cchTextMax;
    int    iImage;
} TCITEMHEADER, FAR *LPTCITEMHEADER;
```

Members

mask
Value that specifies which members to retrieve or set. This member can be a combination of the following values:

TCIF_IMAGE	The **iImage** member is valid.
TCIF_RTLREADING	The string pointed to by **pszText** will be displayed in the opposite direction to the text in the parent window.
TCIF_TEXT	The **pszText** member is valid.

lpReserved1
Reserved member. Do not use.

lpReserved2
Reserved member. Do not use.

pszText
Address of a null-terminated string that contains the tab text if item information is being set. If item information is being retrieved, this member specifies the address of the buffer that receives the tab text.

cchTextMax
Size of the buffer pointed to by the pszText member. If the structure is not receiving information, this member is ignored.

iImage
Index into the tab control's image list, or –1 if there is no image for the tab.

Remarks

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, **pszText** will be displayed in same direction as the text in its parent window. If TCIF_RTLREADING is set, **pszText** will read in the opposite direction from the text in the parent window.

Requirements

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

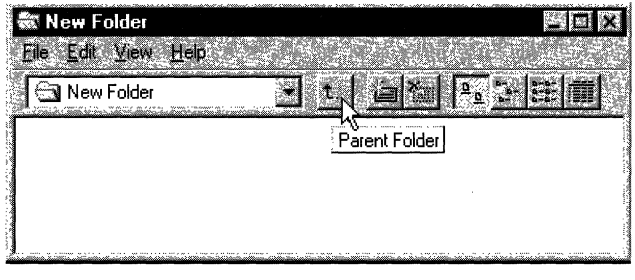
CHAPTER 25

Tooltip Controls

A *tooltip control* is a small pop-up window that displays a single line of text that describes the purpose of a tool in an application. A *tool* is either a window, such as a child window or control, or an application-defined rectangular area within a window's client area.

About Tooltip Controls

A tooltip control is hidden most of the time, appearing only when the user puts the cursor on a tool and leaves it there for approximately one-half second. The tooltip control appears near the cursor and disappears when the user clicks a mouse button or moves the cursor off of the tool. A single tooltip control can support any number of tools. The following illustration shows a standard tooltip control associated with a button in a toolbar control. Tooltips also can have a multiline style, with multiple lines of text, or balloon style, with rounded corners and a stem, similar to a cartoon balloon.



Tooltip Creation

To create a tooltip control, call **CreateWindowEx** and specify the TOOLTIPS_CLASS window class. This class is registered when the common control dynamic-link library (DLL) is loaded. To ensure that this DLL is loaded, include the **InitCommonControls** function in your application. You must define explicitly a tooltip control as topmost. Otherwise, it might be covered by the parent window. The following code fragment shows how to create a tooltip control:

```
HWND hwndTip = CreateWindowEx(NULL, TOOLTIPS_CLASS, NULL,
    WS_POPUP | TTS_NOPREFIX | TTS_ALWAYSTIP,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    hwndParent, NULL, hinstMyDll,
    NULL);
```

(continued)

(continued)

```
SetWindowPos(hwndTip, HWND_TOPMOST,0, 0, 0, 0,
    SWP_NOMOVE | SWP_NOSIZE | SWP_NOACTIVATE);
```

The window procedure for a tooltip control automatically sets the size, position, and visibility of the control. The height of the tooltip window is based on the height of the font currently selected into the device context for the tooltip control. The width varies based on the length of the string currently in the tooltip window.

Activation

A tooltip control itself can be either active or inactive. When it is active, the tooltip control appears when the cursor is on a tool. When it is inactive, the tooltip control does not appear, even if the cursor is on a tool. The **TTM_ACTIVATE** message activates and deactivates a tooltip control.

Types of Tools

A tooltip control can support any number of tools. To support a particular tool, you must register the tool with the tooltip control by sending the control the **TTM_ADDTOOL** message. The message includes the address of a **TOOLINFO** structure, which provides information the tooltip control needs to display text for the tool. The **cbSize** member is required and must specify the size of the structure.

A tooltip control supports tools implemented as windows (such as child windows or control windows) and as rectangular areas within a window's client area. When you add a tool implemented as a rectangular area, the **hwnd** member of **TOOLINFO** must specify the handle to the window that contains the area, and the **rect** member must specify the client coordinates of the area's bounding rectangle. In addition, the **uId** member must specify the application-defined identifier for the tool.

When you add a tool implemented as a window, the **uId** member of **TOOLINFO** must contain the window handle to the tool. Also, the **uFlags** member must specify the TTF_IDISHWND value, which tells the tooltip control to interpret the **uId** member as a window handle.

Tooltip Text

When you add a tool to a tooltip control, the **lpszText** member of the **TOOLINFO** structure must specify the address of the string to display for the tool. You can change the text any time after adding the tool by using the **TTM_UPDATETIPTEXT** message.

If the high-order word of **lpszText** is zero, the low-order word must be the identifier of a string resource. When the tooltip control needs the text, the system loads the specified string resource from the application instance identified by the **hinst** member of **TOOLINFO**.

If you specify the LPSTR_TEXTCALLBACK value in the **lpszText** member, the tooltip control notifies the window specified in the **hwnd** member of **TOOLINFO** whenever the tooltip control needs to display text for the tool. The tooltip control sends the **TTN_NEEDTEXT** notification message to the window. The message includes the address of a **TOOLTIPTEXT** structure, which contains the window handle as well as the application-defined identifier for the tool. The window examines the structure to determine the tool for which text is needed, and it fills the appropriate structure members with information that the tooltip control needs to display the string.

Note The maximum length for tooltip text is 80 characters. For more information, see the **NMTTDISPINFO** structure.

Many applications create toolbars containing tools that correspond to menu commands. For such tools, it is convenient for the tooltip control to display the same text as the corresponding menu item. The system automatically strips the ampersand (&) accelerator characters from all strings passed to a tooltip control, unless the control has the **TTS_NOPREFIX** style.

To retrieve the text for a tool, use the **TTM_GETTEXT** message.

Relaying Mouse Messages to the Tooltip Control

Tooltips are normally displayed when the cursor hovers over an area, typically the rectangle defined by a tool such as a button control. However, Windows sends mouse-related messages only to the window that contains the cursor, not the tooltip control itself. Mouse-related information must be relayed to the tooltip control in order for it to display the tooltip at the appropriate time and place.

You can have messages relayed automatically if:

- The tool is a control or is defined as a rectangle in the tool's **TOOLINFO** structure.
- The window associated with the tool is in the same thread as the tooltip control.

If these two conditions are met, set the TTF_SUBCLASS flag in the **uFlags** member of the tool's **TOOLINFO** structure when you add the tool to the tooltip control with **TTM_ADDTOOL**. The necessary mouse messages will then be automatically relayed to the tooltip control.

Setting TTF_SUBCLASS to have mouse messages relayed to the control is sufficient for most purposes. However, it will not work in cases where there is no direct connection between the tooltip control and the tool's window. For example, if a tool is implemented as a rectangular area in an application-defined window, the window procedure receives the mouse messages. Setting TTF_SUBCLASS is sufficient to ensure that they are passed to the control. However, if a tool is implemented as a system-defined window, mouse messages are sent to that window and are not directly available to the application. In this case, you must either subclass the window or use a message hook to access the mouse messages. You must then explicitly relay mouse messages to the

tooltip control with **TTM_RELAYEVENT**. See **Using a Tooltip Control with a Dialog Box** for an example of how to use TTM_RELAYEVENT.

When a tooltip control receives a **WM_MOUSEMOVE** message, it determines whether the cursor is in the bounding rectangle of a tool. If it is, the tooltip control sets a timer. At the end of the time-out interval, the tooltip control checks the position of the cursor to see if it has moved. If it has not, the tooltip control retrieves the text for the tool, and displays the tooltip. The tooltip control continues to show the window until it receives a relayed button-up or button-down message or until a WM_MOUSEMOVE message indicates that the cursor has moved outside the bounding rectangle of the tool.

A tooltip control actually has three time-out durations associated with it. The *initial duration* is the length of time that the cursor must remain stationary within the bounding rectangle of a tool before the tooltip window is displayed. The *reshow duration* is the length of the delay before subsequent tooltip windows are displayed when the cursor moves from one tool to another. The *pop-up duration* is the length of time that the tooltip window remains displayed before it is hidden. That is, if the cursor remains stationary within the bounding rectangle after the tooltip window is displayed, the tooltip window is automatically hidden at the end of the pop-up duration. You can adjust all of the time-out durations by using the **TTM_SETDELAYTIME** message.

If an application includes a tool implemented as a rectangular area and the size or position of the control changes, the application can use the **TTM_NEWTOOLRECT** message to report the change to the tooltip control. An application does not need to report size and position changes for a tool implemented as a window, because the tooltip control uses the tool's window handle to determine if the cursor is on the tool, not the tool's bounding rectangle.

When a tooltip is about to be displayed, the tooltip control sends the owner window a **TTN_SHOW** notification message. The owner window receives a **TTN_POP** notification when a tooltip is about to be hidden. Each notification is sent in the context of a **WM_NOTIFY** message.

Tooltip Hit-Testing

The **TTM_HITTEST** message allows you to retrieve information that a tooltip control maintains about the tool occupying a particular point. The message includes a **TTHITTESTINFO** structure that contains a window handle, the coordinates of a point, and the address of a **TOOLINFO** structure. The tooltip control determines whether a tool occupies the point and, if it does, fills **TOOLINFO** with information about the tool.

Miscellaneous Messages

The **TTM_GETCURRENTTOOL** and **TTM_GETTOOLINFO** messages fill a **TOOLINFO** structure with information about a tool that has been registered with a tooltip control. The **TTM_SETTOOLINFO** message allows you to change the information that a tooltip control maintains for a particular tool. The **TTM_DELTOOL** message deletes a tool from a tooltip control.

Default Tooltip Control Message Processing

This section describes the messages handled by the window procedure for the TOOLTIPS_CLASS window class.

Message	Description
WM_CREATE	Ensures that the tooltip control has the WS_EX_TOOLWINDOW and WS_POPUP window styles. It also allocates memory and initializes internal variables.
WM_DESTROY	Frees resources allocated for the tooltip control.
WM_GETFONT	Returns the handle of the font that the tooltip control will use to draw text.
WM_MOUSEMOVE	Hides the tooltip window.
WM_PAINT	Draws the tooltip window.
WM_SETFONT	Sets the handle of the font that the tooltip control will use to draw text.
WM_TIMER	Hides the tooltip window if the tool has changed position or if the cursor has moved outside the tool. Otherwise, it shows the tooltip window.
WM_WININICHANGE	Resets internal variables that are based on system metrics.

Using Tooltip Controls

This section provides examples that demonstrate how to create a tooltip control and use a tooltip control with a dialog box.

Creating a Tooltip Control

The following example demonstrates how to create a tooltip control and add several tools to it. The example creates a grid of rectangles in the client area of a window and then uses the **TTM_ADDTOOL** message to add each rectangle to the tooltip control. The TTF_SUBCLASS flag is set in the **uFlags** member of the **TOOLINFO** structure to have mouse messages automatically passed to the tooltip control:

```
// DoCreateTooltip - creates a tooltip control and
// adds some tools to it.
//
// Returns the handle of the tooltip control if successful,
// or NULL otherwise.
//
// hwndOwner - handle of the owner window
```

(continued)

(continued)

```
//
// Global variable
//   g_hinst - handle of the application instance
extern HINSTANCE g_hinst;

HWND DoCreateTooltip(HWND hwndOwner)
{
    HWND hwndTT;    // handle of the tooltip
    int row, col;   // rows and columns
    TOOLINFO ti;    // tool information
    int id = 0;     // offset to string identifiers
    static char *szTips[NUM_TIPS] =   // tooltip text
    {
    "Cut", "Copy", "Paste", "Undo", "Open", "Save"
    };

    // Ensure that the common control DLL is loaded,
    // and create a tooltip control.
    InitCommonControls();

    hwndTT = CreateWindow(TOOLTIPS_CLASS, (LPSTR) NULL, TTS_ALWAYSTIP,
    CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, (HMENU) NULL, g_hinst, NULL);

    if (hwndTT == (HWND) NULL)
        return (HWND) NULL;

    // Divide the client area into a grid of rectangles,
    // and add each rectangle to the tooltip.
    for (row = 0; row < MAX_ROWS ; row++ )
        for (col = 0; col < MAX_COLS; col++) {
            ti.cbSize = sizeof(TOOLINFO);
            ti.uFlags = TTF_SUBCLASS;
            ti.hwnd = hwndOwner;
            ti.hinst = g_hinst;
            ti.uId = (UINT) id;
            ti.lpszText = (LPTSTR) szTips[id++];
            ti.rect.left = col * CX_COLUMN;
            ti.rect.top = row * CY_ROW;
            ti.rect.right = ti.rect.left + CX_COLUMN;
            ti.rect.bottom = ti.rect.top + CY_ROW;

            if (!SendMessage(hwndTT, TTM_ADDTOOL, 0,
                    (LPARAM) (LPTOOLINFO) &ti))
```

```
            return NULL;
        }

    return hwndTT;
}
```

Using a Tooltip Control with a Dialog Box

The following example includes a set of application-defined functions that implement a tooltip control for a dialog box. The DoCreateDialogTooltip function creates a tooltip control and uses the EnumChildWindows function to enumerate the controls in the dialog box. The enumeration procedure, EnumChildProc, registers each control with the tooltip control. The procedure specifies the dialog box as the parent window of each tooltip control and includes the LPSTR_TEXTCALLBACK value for each tooltip control. As a result, the dialog box receives a **WM_NOTIFY** message that contains the **TTN_NEEDTEXT** notification message whenever the tooltip control needs the text for a control. The dialog box procedure calls the OnWMNotify function to process the TTN_NEEDTEXT notifications. OnWMNotify provides the appropriate string based on the identifier of the tooltip control.

This example shows how to use **TTM_RELAYEVENT** to pass mouse messages explicitly to the tooltip control. To access the messages, the DoCreateDialogTooltip function installs a hook procedure of the WH_GETMESSAGE type. The hook procedure, GetMsgProc, monitors the message stream for mouse messages intended for one of the control windows and relays the messages to the tooltip control.

```
// DoCreateDialogTooltip - creates a tooltip control for
//     a dialog box, enumerates the child control windows,
//     and installs a hook procedure to monitor the message
//     stream for mouse messages posted to the control
//     windows.
// Returns TRUE if successful, or FALSE otherwise.
//
// Global variables
// g_hinst - handle to the application instance
// g_hwndTT - handle to the tooltip control
// g_hwndDlg - handle to the dialog box
// g_hhk - handle to the hook procedure

BOOL DoCreateDialogTooltip(void)
{

    // Ensure that the common control DLL is loaded,
    // and create a tooltip control.
    InitCommonControls();
    g_hwndTT = CreateWindowEx(0, TOOLTIPS_CLASS, (LPSTR) NULL,
```

(continued)

(continued)

```
        TTS_ALWAYSTIP, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, g_hwndDlg, (HMENU) NULL, g_hinst, NULL);

    if (g_hwndTT == NULL)
        return FALSE;

    // Enumerate the child windows to register them with
    // the tooltip control.
    if (!EnumChildWindows(g_hwndDlg, (WNDENUMPROC) EnumChildProc, 0))
        return FALSE;

    // Install a hook procedure to monitor the message
    // stream for mouse messages intended for the controls
    // in the dialog box.
    g_hhk = SetWindowsHookEx(WH_GETMESSAGE, GetMsgProc,
        (HINSTANCE) NULL, GetCurrentThreadId());

    if (g_hhk == (HHOOK) NULL)
        return FALSE;

    return TRUE;
}

// EnumChildProc - registers control windows with a tooltip
//     control by using the TTM_ADDTOOL message to pass the
//     address of a TOOLINFO structure.
// Returns TRUE if successful, or FALSE otherwise.
//
// hwndCtrl - handle of a control window
// lParam - application-defined value (not used)
BOOL EnumChildProc(HWND hwndCtrl, LPARAM lParam)
{
    TOOLINFO ti;
    char szClass[64];

    // Skip static controls.
    GetClassName(hwndCtrl, szClass, sizeof(szClass));
    if (lstrcmpi(szClass, "STATIC")) {
        ti.cbSize = sizeof(TOOLINFO);
        ti.uFlags = TTF_IDISHWND;
        ti.hwnd = g_hwndDlg;
        ti.uId = (UINT) hwndCtrl;
        ti.hinst = 0;
        ti.lpszText = LPSTR_TEXTCALLBACK;
```

```
            SendMessage(g_hwndTT, TTM_ADDTOOL, 0,
                (LPARAM) (LPTOOLINFO) &ti);
        }
        return TRUE;
        }

// GetMsgProc - monitors the message stream for mouse
//      messages intended for a control window in the
//      dialog box.
// Returns a message-dependent value.
//
// nCode - hook code
// wParam - message flag (not used)
// lParam - address of an MSG structure
LRESULT CALLBACK GetMsgProc(int nCode, WPARAM wParam,
LPARAM lParam)
{
    MSG *lpmsg;

    lpmsg = (MSG *) lParam;
    if (nCode < 0 || !(IsChild(g_hwndDlg, lpmsg->hwnd)))
        return (CallNextHookEx(g_hhk, nCode, wParam, lParam));

    switch (lpmsg->message) {
        case WM_MOUSEMOVE:
        case WM_LBUTTONDOWN:
        case WM_LBUTTONUP:
        case WM_RBUTTONDOWN:
        case WM_RBUTTONUP:
            if (g_hwndTT != NULL) {
                MSG msg;

                msg.lParam = lpmsg->lParam;
                msg.wParam = lpmsg->wParam;
                msg.message = lpmsg->message;
                msg.hwnd = lpmsg->hwnd;
                SendMessage(g_hwndTT, TTM_RELAYEVENT, 0,
                    (LPARAM) (LPMSG) &msg);
            }
            break;
        default:
            break;
    }
    return (CallNextHookEx(g_hhk, nCode, wParam, lParam));
}
```

(continued)

(continued)

```
// OnWMNotify - provides the tooltip control with the
//     appropriate text to display for a control window.
//     This function is called by the dialog box procedure
//     in response to a WM_NOTIFY message.
//
// lParam - second message parameter of the WM_NOTIFY
//     message.
VOID OnWMNotify(LPARAM lParam)
{
    LPTOOLTIPTEXT lpttt;
    int idCtrl;

    if ((((LPNMHDR) lParam)->code) == TTN_NEEDTEXT) {
        idCtrl = GetDlgCtrlID((HWND) ((LPNMHDR) lParam)->idFrom);
        lpttt = (LPTOOLTIPTEXT) lParam;

        switch (idCtrl) {
            case ID_HORZSCROLL:
                lpttt->lpszText = "A horizontal scroll bar.";
                return;

            case ID_CHECK:
                lpttt->lpszText = "A check box.";
                return;

            case ID_EDIT:
                lpttt->lpszText = "An edit control.";
                return;
        }
    }
return;
}
```

Tooltip Control Updates in Internet Explorer

Tooltip controls in Microsoft Internet Explorer support two new features: *tracking tooltips* and *multiline tooltips*.

Tracking Tooltips

Tooltip controls support tracking tooltips, which are tooltip windows that you can position dynamically on the screen. By rapidly updating the position, the tooltip window appears

to move smoothly, or "track." This functionality can be useful if you need tooltip text to follow the position of the pointer as it moves.

To create a tracking tooltip, use the **TTM_ADDTOOL** message, including the TTF_TRACK flag in the **uFlags** member of the accompanying **TOOLINFO** structure.

Your application must manually activate and deactivate a tracking tooltip using the **TTM_TRACKACTIVATE** message. While the tooltip is active, your application must supply the location at which the tooltip window will appear by using the **TTM_TRACKPOSITION** message. Tracking tooltip controls do not support the TTF_SUBCLASS style, so all mouse events must be forwarded from the parent to the child using **TTM_RELAYEVENT** messages.

The **TTM_TRACKPOSITION** message causes the tooltip control to display the window using one of two placement styles:

- By default, the tooltip is displayed next to the corresponding tool in a position the control chooses. The location chosen is relative to the coordinates you provide using this message. In this case, the tooltip window appears to move beside the corresponding tool.
- If you include the TTF_ABSOLUTE value in the **uFlags** member of the **TOOLINFO** structure the tooltip will be displayed at the pixel location specified in the message. In this case, the control does not attempt to change the tooltip window's location from the coordinates you provide.

For more information and implementation details, see **Creating Tracking Tooltips** and **Supporting Tracking Tooltips**.

Creating Tracking Tooltips

The following example demonstrates how to create a tooltip control and assign a tool to it. The example specifies the main window's entire client area as the tool, but you could specify distinct portions of the client area or specify a different window altogether.

The example uses the **TTM_ADDTOOL** message to add the tool to the tooltip control. Tracking tooltips do not support the TTF_SUBCLASS flag, so the control's owner must manually forward pertinent messages (like **WM_MOUSEMOVE**) by using **TTM_RELAYEVENT**.

Additionally, the **uFlags** member of the **TOOLINFO** structure used in the example includes the TTF_ABSOLUTE flag. This flag causes the tooltip control to display tooltip text at the exact coordinates the application provides when it sends the **TTM_TRACKPOSITION** message. Without the TTF_ABSOLUTE flag, the tooltip control chooses a location to display the tooltip text based on the coordinates you provide. This causes tooltip text to appear next to the corresponding tool, but not necessarily at the exact coordinates the application provided.

For additional information about using the **TTM_TRACKPOSITION** message, see *Supporting Tracking Tooltips.*

```
HWND WINAPI CreateTT(HWND hwndOwner)
{
    INITCOMMONCONTROLSEX icex;
    HWND         hwndTT;
    TOOLINFO     ti;

    // Load the tooltips class from the DLL.
    icex.dwSize = sizeof(icex);
    icex.dwICC  = ICC_BAR_CLASSES;

    if(!InitCommonControlsEx(&icex))
        return NULL;

    // Create the tooltip control.
    hwndTT = CreateWindow(TOOLTIPS_CLASS, TEXT(""),
                          WS_POPUP,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, (HMENU)NULL, g_hinst,
                          NULL);

    // Prepare the TOOLINFO structure to be used for a
    //     tracking tooltip.
    ti.cbSize = sizeof(TOOLINFO);
    ti.uFlags = TTF_IDISHWND | TTF_TRACK | TTF_ABSOLUTE;
    ti.hwnd   = hwndOwner;
    ti.uId    = (UINT)g_hwndMain;
    ti.hinst  = g_hinst;
    ti.lpszText  = LPSTR_TEXTCALLBACK;
    ti.rect.left = ti.rect.top = ti.rect.bottom = ti.rect.right = 0;

    // Add the tool to the control, displaying an error
    //    if needed.
    if(!SendMessage(hwndTT,TTM_ADDTOOL,0,(LPARAM)&ti)){
        MessageBox(hwndOwner,"Couldn't create the tooltip
control.","Error",MB_OK);
        return NULL;
    }

    // Activate (display) the tracking tooltip. Then, set
    // a global flag value to indicate that the tooltip is
    // active, so other functions can check to see if it's
    // visible.
    SendMessage(hwndTT,TTM_TRACKACTIVATE,(WPARAM)TRUE,(LPARAM)&ti);
    g_bIsVisible = TRUE;
```

```
    return(hwndTT);
}
```

Supporting Tracking Tooltips

The following example is a simple window process function that supports tracking tooltips. It requests the current position of the cursor using the **GetCursorPos** function, and then adds 15 pixels to the x- and y-coordinates, so the tooltip appears slightly below and to the right of the pointer.

Note that the example relies on the value of a global variable, g_bIsVisible, to determine whether the application should send the **TTM_TRACKPOSITION** message. For the purpose of this example, g_bIsVisible is a Boolean variable that another function sets to TRUE upon sending the **TTM_TRACKACTIVATE** message to activate the tooltip. This way, if the tooltip is inactive, the additional overhead to calculate and send a message is not incurred:

```
LRESULT CALLBACK WndProc (HWND hwnd, UINT msg, WPARAM
wParam, LPARAM lParam)
{
    HDC   hdc;
    PAINTSTRUCT ps;
    MSG   mssg;
    POINT pt;

    switch(msg){
        case WM_MOUSEMOVE:
            if(g_bIsVisible){
                mssg.hwnd    = g_hwndMain;
                mssg.message = msg;
                mssg.wParam  = wParam;
                mssg.lParam  = lParam;

                GetCursorPos(&pt);
                mssg.pt.x    = pt.x;
                mssg.pt.y    = pt.y;

#define X_OFFSET 15
#define Y_OFFSET X_OFFSET
                SendMessage(g_hwndTT,
                            TTM_TRACKPOSITION,0,
                            (LPARAM)MAKELPARAM(pt.x+X_OFFSET,
                            pt.y+Y_OFFSET));
            }
            break;
```

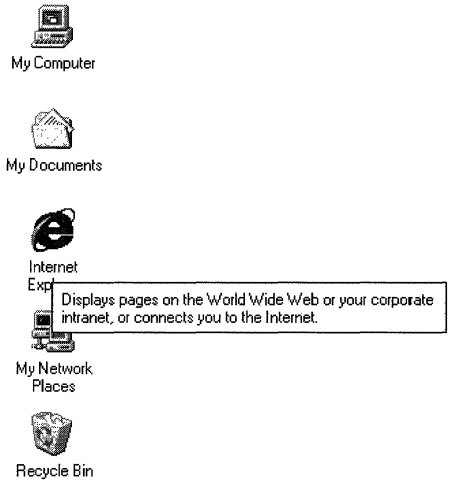
(continued)

(continued)

```
/*
*
*  Other standard window messages can be handled here.
*
*/
    }
    return 0;
}
```

Multiline Tooltips

Multiline tooltip support allows text to be displayed on more than one line. This feature is useful if your message is too lengthy to be read easily as a single line of text. The following example shows a multiline tooltip that is displayed when the cursor hovers over the Internet Explorer icon on the desktop.



Creating Multiline Tooltips

Your application creates a multiline tooltip by responding to a **TTN_GETDISPINFO** notification message. To force the tooltip control to use multiple lines, send a **TTM_SETMAXTIPWIDTH** message, specifying the width of the display rectangle. Text that exceeds this width will wrap to the next line, instead of widening the display region. The rectangle height will be increased as needed to accommodate the additional lines. The tooltip control will wrap the lines automatically, or you can use a carriage return/line-feed combination, "\r\n", to force line breaks at particular locations.

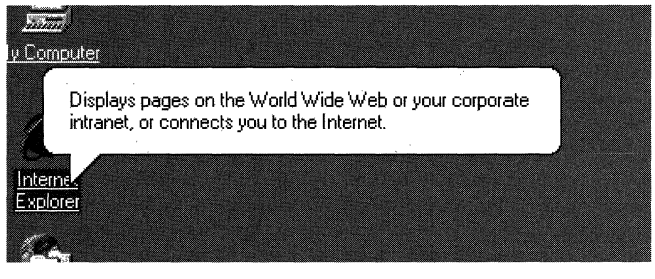
Note that the text buffer specified by the **szText** member of the **NMTTDISPINFO** structure can accommodate only 80 characters. If you need to use a longer string, point the **lpszText** member of **NMTTDISPINFO** to a buffer containing the desired text.

The following code fragment is an example of a simple TTN_GETDISPINFO notification handler. It creates a multiline tooltip by setting the display rectangle to 300 pixels and setting the **lpszText** member of **NMTTDISPINFO** to point to a buffer with the desired text:

```
char szLongMessage[] =
"This is a long message for the tooltip, which will "
"automatically be wrapped when it exceeds the maximum "
"tip width.  Alternatively, you can use a \r\n carriage "
"return/line-feed combination\r\n to force line breaks at "
"specific\r\n locations.";

switch (lpnmhdr->code) {
  case TTN_GETDISPINFO:
    lpttd = (LPNMTTDISPINFO)lpnmhdr;
    SendMessage(lpnmhdr->hwndFrom, TTM_SETMAXTIPWIDTH, 0, 300);
    lpttd->lpszText = szLongMessage;
    return 0;

    ...
 //Other notification handlers go here, as needed.
}
```

Balloon Tooltips

Balloon tooltips are similar to standard tooltips, but are displayed in a cartoon style "balloon" with a stem pointing to the tool.



Balloon tooltips can be either single-line or multiline, and they are created and handled in much the same way as standard tooltips. The default position of the stem and rectangle is shown in the illustration. If the tool is too close to the top of the screen, the tooltip appears below and to the right of the tool's rectangle. If the tool is too close to the right side of the screen, similar principles apply, but the tooltip appears to the left of the tool's rectangle.

You can change the default positioning by setting the TTF_CENTERTIP flag in the **uFlags** member of the tooltip's **TOOLINFO** structure. In that case, the stem normally will point to the center of the lower edge of the tool's rectangle and the text rectangle will be displayed directly below the tool. The stem will attach to the text rectangle at the center

of the upper edge. If the tool is too close to the bottom of the screen, the text rectangle will be centered above the tool, and the stem will attach to the center of the lower edge.

If you want to specify where the stem points, set the TTF_TRACK flag in the **uFlags** member of the tooltip's **TOOLINFO** structure. You then specify the coordinate by sending a **TTM_TRACKPOSITION** message, with the x- and y-coordinates in the *lParam* value. If TTF_CENTERTIP is also set, the stem still points to the position specified by the **TTM_TRACKPOSITION** message.

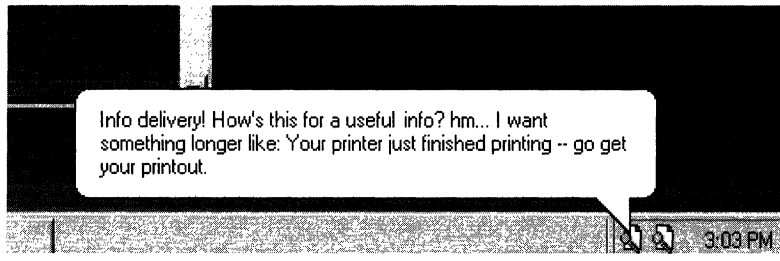
The following code fragment illustrates how to implement a centered balloon tooltip:

```
hwndToolTips = CreateWindow(TOOLTIPS_CLASS,
                            NULL,
                            WS_POPUP | TTS_NOPREFIX | TTS_BALLOON,
                            0, 0,
                            0, 0,
                            NULL, NULL,
                            g_hinst,
                            NULL);
if (hwndTooltip)
{
// Do the standard tooltip coding.
    TOOLINFO ti;

    ti.cbSize = sizeof(ti);
    ti.uFlags = TTF_TRANSPARENT | TTF_CENTERTIP;
    ti.hwnd = hwnd;
    ti.uId = 0;
    ti.hinst = NULL;
    ti.lpszText = LPSTR_TEXTCALLBACK;

    GetClientRect(hwnd, &ti.rect);
    SendMessage(hwndToolTips, TTM_ADDTOOL, 0, (LPARAM) &ti );
}
```

Balloon Tooltips for Status-Bar Icons

An un-intrusive way to display an explanatory message for a status-bar icon is to implement a balloon tooltip with its stem pointing to the icon. The tooltip will disappear when clicked, but you can specify also a time-out value. The tooltip will look similar to the following illustration.

To pop up a balloon tooltip, set the NIF_INFO flag in the **NOTIFYICONDATA** structure, and use the **szInfo** and **uTimeout** members to specify the tooltip text and time-out duration. The following code fragment illustrates how to add a ballon tooltip to a status-bar icon:




```
NOTIFYICONDATA IconData = {0};

IconData.cbSize = SIZEOF(IconData);
IconData.hWnd = hwndNI;
IconData.uFlags = NIF_INFO;

lstrcpyn(IconData.szInfo, TEXT("Your message text goes here."),
ARRAYSIZE(IconData.szInfo));
IconData.uTimeout = 15000; // in milliseconds

Shell_NotifyIcon(NIM_MODIFY, &IconData);
```

For a detailed discussion of the status bar, see the **Taskbar** documentation.

Tooltip Styles

Tooltip controls support a variety of control styles in addition to standard window styles. A tooltip control always has the WS_POPUP and WS_EX_TOOLWINDOW window styles, regardless of whether you specify them when creating the control.

The following control styles are used with tooltip controls:

TTS_ALWAYSTIP
: Indicates that the tooltip control appears when the cursor is on a tool, even if the tooltip control's owner window is inactive. Without this style, the tooltip appears only when the tool's owner window is active.

TTS_BALLOON
: **Version 5.80**. Indicates that the tooltip control has the appearance of a cartoon "balloon," with rounded corners and a stem pointing to the item.

TTS_NOANIMATE
: **Version 5.80**. Disables sliding tooltip animation on Microsoft Windows 98 and Microsoft Windows 2000 systems. This style is ignored on earlier systems.

TTS_NOFADE
: **Version 5.80**. Disables fading tooltip animation on Windows 2000 systems. This style is ignored on earlier Windows NT systems, and on Windows 95 and Windows 98.

TTS_NOPREFIX
Prevents the system from stripping the ampersand (&) character from a string. Without this style, the system automatically strips ampersand characters. This allows an application to use the same string as both a menu item and as text in a tooltip control.

Tooltip Control Reference

Tooltip Control Messages

TTM_ACTIVATE

Activates or deactivates a tooltip control.

```
TTM_ACTIVATE
    wParam = (WPARAM) (BOOL) fActivate;
    lParam = 0;
```

Parameters

fActivate
Activation flag. If this parameter is TRUE, the tooltip control is activated. If it is FALSE, the tooltip control is deactivated.

Return Values

No return value.

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 commctrl.h.

TTM_ADDTOOL

Registers a tool with a tooltip control.

```
TTM_ADDTOOL
    wParam = 0;
    lParam = (LPARAM) (LPTOOLINFO) lpti;
```

Parameters

lpti

Address of a **TOOLINFO** structure containing information that the tooltip control needs to display text for the tool. The **cbSize** member of this structure must be filled in before sending this message.

Return Values

Returns TRUE if successful, or FALSE otherwise.

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in commctrl.h.

TTM_ADJUSTRECT

Calculates a tooltip control's text-display rectangle from its window rectangle, or the tooltip window rectangle needed to display a specified text-display rectangle.

```
TTM_ADJUSTRECT
   wParam = (WPARAM) (BOOL) fLarger;
   lParam = (LPARAM) (LPRECT) prc;
```

Parameters

fLarger

Value that specifies which operation to perform. If TRUE, *prc* is used to specify a text-display rectangle, and it receives the corresponding window rectangle. If FALSE, *prc* is used to specify a window rectangle, and it receives the corresponding text-display rectangle.

prc

RECT structure to hold either a tooltip window rectangle or a text-display rectangle.

Return Values

Returns a nonzero value if the rectangle is successfully adjusted, and returns zero if an error occurs.

Remarks

This message is particularly useful when you want to use a tooltip control to display the full text of a string that normally gets truncated. It is commonly used with listview and treeview controls. You normally send this message in response to a **TTN_SHOW** notification message, so that you can position the tooltip control properly.

The tooltip's window rectangle is somewhat larger than the text-display rectangle that bounds the tooltip string. The window origin also is offset up and to the left from the origin of the text-display rectangle. To position the text-display rectangle, you must calculate the corresponding window rectangle, and use that rectangle to position the tooltip. TTM_ADJUSTRECT handles this calculation for you.

If you set *fLarger* to TRUE, TTM_ADJUSTRECT takes the size and position of the desired tooltip text-display rectangle, and passes back the size and position of the tooltip window needed to display the text in the specified position. If you set *fLarger* to FALSE, you can specify a tooltip window rectangle, and TTM_ADJUSTRECT will return the size and position of its text rectangle.

Requirements

Version 5.80 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5 or later installed).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).
Windows CE: Unsupported.
Header: Declared in comctl32.h.

TTM_DELTOOL

Removes a tool from a tooltip control.

```
TTM_DELTOOL
    wParam = 0;
    lParam = (LPARAM) (LPTOOLINFO) lpti;
```

Parameters

lpti

Address of a **TOOLINFO** structure. The **hwnd** and **uId** members identify the tool to remove, and the **cbSize** member must specify the size of the structure. All other members are ignored.

Return Values

No return value.

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 commctrl.h.

TTM_ENUMTOOLS

Retrieves the information that a tooltip control maintains about the current tool—that is, the tool for which the tooltip is currently displaying text.

```
TTM_ENUMTOOLS
    wParam = (WPARAM) (UINT) iTool;
    lParam = (LPARAM) (LPTOOLINFO) lpti;
```

Parameters

iTool
Zero-based index of the tool for which to retrieve information.

lpti
Address of a **TOOLINFO** structure that receives information about the tool. Before sending this message, the **cbSize** member must specify the size of the structure.

Return Values

Returns TRUE if any tools are enumerated, 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 commctrl.h.

TTM_GETBUBBLESIZE

Returns the width and height of a tooltip control.

```
TTM_GETBUBBLESIZE
    wParam = 0;
    lParam = (LPARAM) (LPTOOLINFO) pTtm;
```

Parameters

pTtm
Pointer to the tooltip's **TOOLINFO** structure.

Return Values

Returns the width of the tooltip in the low word and the height in the high word if successful. Otherwise, it returns FALSE.

Remarks

If the TTF_TRACK and TTF_ABSOLUTE flags are set in the **uFlags** member of the tooltip's **TOOLINFO** structure, this message can be used to help position the tooltip accurately.

Requirements

Version 5.80 and later of Comctl32.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 comctl32.h.

TTM_GETCURRENTTOOL

Retrieves the information for the current tool in a tooltip control.

```
TTM_GETCURRENTTOOL
    wParam = 0;
    lParam = (LPARAM)(LPTOOLINFO) lpti;
```

Parameters

lpti

Address of a **TOOLINFO** structure that receives information about the current tool. If this value is NULL, the return value indicates the existence of the current tool without actually retrieving the tool information. If this value is not NULL, the **cbSize** member of the **TOOLINFO** structure must be filled in before sending this message.

Return Values

Returns nonzero if successful, or zero otherwise. If *lpti* is NULL, returns nonzero if a current tool exists, 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 commctrl.h.

TTM_GETDELAYTIME

Retrieves the initial, pop-up, and reshow durations currently set for a tooltip control.

```
TTM_GETDELAYTIME
    wParam = (DWORD) dwDuration;
    lParam = 0;
```

Parameters

dwDuration

Flag that specifies which duration value will be retrieved. This parameter can have one of the following values:

TTDT_AUTOPOP	Retrieve the length of time the tooltip window remains visible if the pointer is stationary within a tool's bounding rectangle.
TTDT_INITIAL	Retrieve the length of time the pointer must remain stationary within a tool's bounding rectangle before the tooltip window appears.
TTDT_RESHOW	Retrieve the length of time it takes for subsequent tooltip windows to appear as the pointer moves from one tool to another.

Return Values

Returns an INT value with the specified duration in milliseconds.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

See Also

TTM_SETDELAYTIME

TTM_GETMARGIN

Retrieves the top, left, bottom, and right margins set for a tooltip window. A margin is the distance, in pixels, between the tooltip window border and the text contained within the tooltip window.

```
TTM_GETMARGIN
    wParam = 0;
    lParam = (LPARAM)(LPRECT) lprc;
```

Parameters

lprc

Address of a **RECT** structure that will receive the margin information.

The members of the **RECT** structure do not define a bounding rectangle. For the purpose of this message, the structure members are interpreted as follows:

top	Distance between top border and top of tooltip text, in pixels.
left	Distance between left border and left end of tooltip text, in pixels.
bottom	Distance between bottom border and bottom of tooltip text, in pixels.
right	Distance between right border and right end of tooltip text, in pixels.

Return Values

The return value for this message is not used.

Remarks

All four margins default to zero when you create the tooltip control.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

See Also

TTM_SETMARGIN

TTM_GETMAXTIPWIDTH

Retrieves the maximum width for a tooltip window.

```
TTM_GETMAXTIPWIDTH
    wParam = 0;
    lParam = 0;
```

Return Values

Returns an INT value that represents the maximum tooltip width, in pixels. If no maximum width was set previously, the message returns –1.

Remarks

The maximum tooltip width value does not indicate a tooltip window's actual width. Instead, if a tooltip string exceeds the maximum width, the control breaks the text into multiple lines, using spaces to determine line breaks. If the text cannot be segmented into multiple lines, it will be displayed on a single line. The length of this line can exceed the maximum tooltip width.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

TTM_SETMAXTIPWIDTH

TTM_GETTEXT

Retrieves the information a tooltip control maintains about a tool.

```
TTM_GETTEXT
    wParam = 0;
    lParam = (LPARAM) (LPTOOLINFO) lpti;
```

Parameters

lpti

Address of a **TOOLINFO** structure.

The **cbSize** member of this structure must be filled in before sending this message. Set the **hwnd** and **uId** members to identify the tool for which to retrieve information. Set the **lpszText** member to point to a buffer that receives the text. There currently is no way to specify the size of the buffer or to determine the required buffer size.

Return Values

No return value.

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 commctrl.h.

TTM_GETTIPBKCOLOR

Retrieves the background color in a tooltip window.

```
TTM_GETTIPBKCOLOR
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a **COLORREF** value that represents the background color.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

See Also

TTM_SETTIPBKCOLOR

TTM_GETTOOLCOUNT

Retrieves a count of the tools maintained by a tooltip control.

```
TTM_GETTOOLCOUNT
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a count of tools.

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 commctrl.h.

TTM_GETTOOLINFO

Retrieves the information that a tooltip control maintains about a tool.

```
TTM_GETTOOLINFO
    wParam = 0;
    lParam = (LPARAM) (LPTOOLINFO) lpti;
```

Parameters

lpti

Address of a **TOOLINFO** structure. When sending the message, the **hwnd** and **uId** members identify a tool, and the **cbSize** member must specify the size of the structure. If the tooltip control includes the tool, the structure receives information about the tool.

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 commctrl.h.

TTM_HITTEST

Tests a point to determine whether it is within the bounding rectangle of the specified tool and, if it is, retrieves information about the tool.

```
TTM_HITTEST
    wParam = 0;
    lParam = (LPARAM) (LPHITTESTINFO) lphti;
```

Parameters

lphti

Address of a **TTHITTESTINFO** structure. When sending the message, the **hwnd** member must specify the handle to a tool and the **pt** member must specify the coordinates of a point. If the return value is TRUE, the **ti** member (a **TOOLINFO** structure) receives information about the tool that occupies the point. The **cbSize** member of the **ti** structure must be filled in before sending this message.

Return Values

Returns TRUE if the tool occupies the specified point, 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 commctrl.h.

TTM_NEWTOOLRECT

Sets a new bounding rectangle for a tool.

```
TTM_NEWTOOLRECT
    wParam = 0;
    lParam = (LPARAM) (LPTOOLINFO) lpti;
```

Parameters

lpti

Address of a **TOOLINFO** structure. The **hwnd** and **uId** members identify a tool, and the **rect** member specifies the new bounding rectangle. The **cbSize** member of this structure must be filled in before sending this message.

Return Values

No return value.

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 commctrl.h.

TTM_POP

Removes a displayed tooltip window from view.

```
TTM_POP
    wParam = 0;
    lParam = 0;
```

Return Values

The return value for this message is not used.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

TTM_RELAYEVENT

Passes a mouse message to a tooltip control for processing.

```
TTM_RELAYEVENT
    wParam = 0;
    lParam = (LPARAM) (LPMSG) lpmsg;
```

Parameters

lpmsg
Address of an **MSG** structure that contains the message to relay.

Return Values

No return value.

Remarks

A tooltip control processes only the following messages passed to it by the TTM_RELAYEVENT message:

- WM_LBUTTONDOWN
- WM_LBUTTONUP
- WM_MBUTTONDOWN
- WM_MBUTTONUP

- WM_MOUSEMOVE
- WM_RBUTTONDOWN
- WM_RBUTTONUP

All other messages are ignored.

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 commctrl.h.

TTM_SETDELAYTIME

Sets the initial, pop-up, and reshow durations for a tooltip control.

```
TTM_SETDELAYTIME
    wParam = (WPARAM)(DWORD) dwDuration;
    lParam = (LPARAM)(INT) MAKELONG(iTime,0);
```

Parameters

dwDuration

Flag that specifies the duration value to set. This parameter can be one of the following values:

TTDT_AUTOMATIC	Set all three delay times to default proportions. The autopop time will be ten times the initial time and the reshow time will be one fifth the initial time. If this flag is set, use a positive value of *iTime* to specify the initial time, in milliseconds. Set *iTime* to a negative value to specify the default values of 500 ms, 5000 ms, and 100 ms for the initial, autopop, and reshow times, respectively.
TTDT_AUTOPOP	Set the length of time a tooltip window remains visible if the pointer is stationary within a tool's bounding rectangle.
TTDT_INITIAL	Set the length of time a pointer must remain stationary within a tool's bounding rectangle before the tooltip window appears.
TTDT_RESHOW	Set the length of time it takes for subsequent tooltip windows to appear as the pointer moves from one tool to another.

iTime

Delay time, in milliseconds.

Return Values

The return value for this message is not used.

Windows NT/2000: Requires Windows NT 3.51 or later
Windows 95/98: Requires Windows 95 or later
Windows CE: Unsupported.
Header: Declared in commctrl.h.

TTM_GETDELAYTIME

TTM_SETMARGIN

Sets the top, left, bottom, and right margins for a tooltip window. A margin is the distance, in pixels, between the tooltip window border and the text contained within the tooltip window.

```
TTM_SETMARGIN
    wParam = 0;
    lParam = (LPARAM)(LPRECT) lprc;
```

Parameters

lprc

Address of a **RECT** structure that contains the margin information to be set.

The members of the **RECT** structure do not define a bounding rectangle. For the purpose of this message, the structure members are interpreted as follows:

top	Distance between top border and top of tooltip text, in pixels.
left	Distance between left border and left end of tooltip text, in pixels.
bottom	Distance between bottom border and bottom of tooltip text, in pixels.
right	Distance between right border and right end of tooltip text, in pixels.

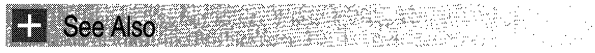
Return Values

The return value for this message is not used.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).

Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.



TTM_GETMARGIN

TTM_SETMAXTIPWIDTH

Sets the maximum width for a tooltip window.

```
TTM_SETMAXTIPWIDTH
    wParam = 0;
    lParam = (LPARAM)(INT) iWidth;
```

Parameters

iWidth
Maximum tooltip window width to be set.

Return Values

Returns an INT value that represents the previous maximum tooltip width.

Remarks

The maximum tooltip width value does not indicate a tooltip window's actual width. Instead, if a tooltip string exceeds the maximum width, the control breaks the text into multiple lines, using spaces to determine line breaks. If the text cannot be segmented into multiple lines, it will be displayed on a single line. The length of this line can exceed the maximum tooltip width.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

TTM_GETMAXTIPWIDTH

TTM_SETTIPBKCOLOR

Sets the background color in a tooltip window.

```
TTM_SETTIPBKCOLOR
    wParam = (WPARAM)(COLORREF) clr;
    lParam = 0;
```

Parameters

clr
New background color.

Return Values

The return value for this message is not used.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

See Also

TTM_GETTIPBKCOLOR

TTM_SETTIPTEXTCOLOR

Sets the text color in a tooltip window.

```
TTM_SETTIPTEXTCOLOR
    wParam = (WPARAM)(COLORREF) clr;
    lParam = 0;
```

Parameters

clr
New text color.

Return Values

The return value for this message is not used.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

See Also

TTM_GETTIPTEXTCOLOR

TTM_SETTITLE

Adds a standard icon and title string to a tooltip.

```
TTM_SETTITLE
wParam = icon;
lParam = (LPCTSTR) pszTitle;
```

Parameters

icon

Set *wParam* to one of the following values to specify the icon to be displayed:

0	No icon
1	Info icon
2	Warning icon
3	Error Icon

pszTitle

A pointer to the title string. You must assign a value to *pszTitle*.

Return Values

Returns TRUE if successful, FALSE if not.

Requirements

Version 5.80 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 5.0 or later installed).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 5.0 or later).

Windows CE: Unsupported.
Header: Declared in comctl32.h.

TTM_SETTOOLINFO

Sets the information that a tooltip control maintains for a tool.

```
TTM_SETTOOLINFO
    wParam = 0;
    lParam = (LPARAM) (LPTOOLINFO) lpti;
```

Parameters

lpti

Address of a **TOOLINFO** structure that specifies the information to set. The **cbSize** member of this structure must be filled in before sending this message.

Return Values

No return value.

Remarks

Some internal properties of a tool are established when the tool is created, and are not recomputed when a TTM_SETTOOLINFO message is sent. If you assign values to a **TOOLINFO** structure and pass it to the tooltip control with a TTM_SETTOOLINFO message, these properties can be lost. Instead, your application should first request the tool's current TOOLINFO structure by sending the tooltip control a **TTM_GETTOOLINFO** message. Then, modify the members of this structure as needed and pass it back to the tooltip control with TTM_SETTOOLINFO.

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 commctrl.h.

TTM_TRACKACTIVATE

Activates or deactivates a tracking tooltip.

```
TTM_TRACKACTIVATE
    wParam = (WPARAM)(BOOL) bActivate;
    lParam = (LPARAM)(LPTOOLINFO) lpti;
```

Parameters

bActivate

Value specifying whether tracking is being activated or deactivated. This value can be one of the following:

FALSE	Deactivate tracking.
TRUE	Activate tracking.

lpti

Address of a **TOOLINFO** structure that identifies the tool to which this message applies. The **hwnd** and **uId** members identify the tool, and the **cbSize** member specifies the size of the structure. All other members are ignored.

Return Values

The return value for this message is not used.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

See Also

Tracking Tooltips, **TTM_TRACKPOSITION**

TTM_TRACKPOSITION

Sets the position of a tracking tooltip.

```
TTM_TRACKPOSITION
    wParam = 0;
    lParam = (LPARAM)(DWORD) MAKELONG(xPos, yPos);
```

Parameters

xPos and yPos

The x- and y-coordinates of the point at which the tracking tooltip will be displayed, in screen coordinates.

Return Values

The return value for this message is not used.

Remarks

The tooltip control chooses where to display the tooltip window based on the coordinates you provide with this message. This causes the tooltip window to appear beside the tool to which it corresponds. To have tooltip windows displayed at specific coordinates, include the TTF_ABSOLUTE flag in the **uFlags** member of the **TOOLINFO** structure when adding the tool.

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

Tracking Tooltips, **TTM_TRACKACTIVATE**

TTM_UPDATE

Forces the current tool to be redrawn.

```
TTM_UPDATE
    wParam = 0;
    lParam = 0;
```

Return Values

The return value for this message is not used.

Version 4.71 and later of Comctl32.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 commctrl.h.

TTM_UPDATETIPTEXT

Sets the tooltip text for a tool.

```
TTM_UPDATETIPTEXT
    wParam = 0;
    lParam = (LPARAM) (LPTOOLINFO) lpti;
```

Parameters

lpti

Address of a **TOOLINFO** structure. The **hinst** and **lpszText** members must specify the instance handle and the address of the text. The **hwnd** and **uId** members identify the tool to update. The **cbSize** member of this structure must be filled in before sending this message.

Return Values

No return value.

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 commctrl.h.

TTM_WINDOWFROMPOINT

Allows a subclass procedure to cause a tooltip to display text for a window other than the one beneath the mouse cursor.

```
TTM_WINDOWFROMPOINT
    wParam = 0;
    lParam = (POINT FAR *) lppt;
```

Parameters

lppt

Address of a **POINT** structure that defines the point to be checked.

Return Values

The return value is the handle to the window that contains the point, or NULL if no window exists at the specified point.

Remarks

This message is intended to be processed by an application that subclasses a tooltip. It is not intended to be sent by an application. A tooltip sends this message to itself before displaying the text for a window. By changing the coordinates of the point specified by *lppt*, the subclass procedure can cause the tooltip to display text for a window other than the one beneath the mouse cursor.

Windows NT/2000: Requires Windows NT 3.51 or later.
Windows 95/98: Requires Windows 95 or later.
Windows CE: Unsupported.
Header: Declared in commctrl.h.

Tooltip Control Notification Messages

NM_CUSTOMDRAW (Tooltip)

Sent by a tooltip control to notify its parent windows about drawing operations. This notification message is sent in the form of a **WM_NOTIFY** message.

```
NM_CUSTOMDRAW
    lpNMCustomDraw = (LPNMTTCUSTOMDRAW) lParam;
```

Parameters

lpNMCustomDraw
Pointer to an **NMTTCUSTOMDRAW** structure that contains information about the drawing operation.

Return Values

The value that your application can return depends on the current drawing stage. The **dwDrawStage** member of the associated **NMCUSTOMDRAW** structure holds a value that specifies the drawing stage. You must return one of the following values.

When **dwDrawStage** equals CDDS_PREPAINT:

CDRF_DODEFAULT
The control will draw itself. It will not send any additional **NM_CUSTOMDRAW** notification messages for this paint cycle.

CDRF_NOTIFYITEMDRAW
The control will notify the parent of any item-related drawing operations. It will send **NM_CUSTOMDRAW** notification messages before and after drawing items.

CDRF_NOTIFYITEMERASE
The control will notify the parent when an item will be erased. It will send **NM_CUSTOMDRAW** notification messages before and after erasing items.

CDRF_NOTIFYPOSTERASE
The control will notify the parent after erasing an item.

CDRF_NOTIFYPOSTPAINT
The control will notify the parent after painting an item.

CDRF_NOTIFYSUBITEMDRAW
Version 4.71. The control will notify the parent when a list view subitem is being drawn.

When **dwDrawStage** equals CDDS_ITEMPREPAINT:

CDRF_NEWFONT
Your application specified a new font for the item. The control will use the new font. For more information about changing fonts, see **Changing Fonts and Colors**.

CDRF_SKIPDEFAULT
Your application drew the item manually. The control will not draw the item.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

See Also

Using Custom Draw

TTN_GETDISPINFO

Sent by a tooltip control to retrieve information needed to display a tooltip window. This notification supersedes the TTN_NEEDTEXT notification. This notification is sent in the form of a **WM_NOTIFY** message.

```
TTN_GETDISPINFO
    lpnmtdi = (LPNMTTDISPINFO)lParam;

#define TTN_NEEDTEXT TTN_GETDISPINFO
```

Parameters

lpnmtdi
Address of an **NMTTDISPINFO** structure that identifies the tool that needs text and receives the requested information.

Return Values

The return value for this notification is not used.

Remarks

Fill the structure's appropriate fields to return the requested information to the tooltip control. If your message handler sets the **uFlags** field of the **NMTTDISPINFO** structure to TTF_DI_SETITEM, the tooltip control stores the information and will not request it again.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

TTN_POP

Notifies the owner window that a tooltip is about to be hidden. This notification message is sent in the form of a **WM_NOTIFY** message.

```
TTN_POP
    idTT = (int) wParam;
    pnmh = (LPNMHDR) lParam;
```

Parameters

idTT
Identifier of the tooltip control.

pnmh
Address of an **NMHDR** structure.

Return Values

No return value.

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 commctrl.h.

TTN_SHOW

Notifies the owner window that a tooltip control is about to be displayed. This notification message is sent in the form of a **WM_NOTIFY** message.

```
TTN_SHOW
    idTT = (int) wParam;
    pnmh = (LPNMHDR) lParam;
```

Parameters

idTT
Identifier of the tooltip control.

pnmh
Pointer to an **NMHDR** structure.

Return Values

Version 4.70. To display the tooltip in its default location, return zero. To customize the tooltip position, reposition the tooltip window with the **SetWindowPos** function and return TRUE.

Note For versions earlier than 4.70, there is no return value.

Remarks

A tooltip's window rectangle is somewhat larger than its text-display rectangle, and its origin is offset up and to the left. If you need to accurately position the text-display rectangle of a tool tip, the **TTM_ADJUSTRECT** message converts a text-display rectangle into the corresponding tooltip window rectangle, and vice versa.

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 commctrl.h.

Tooltip Control Structures

NMTTCUSTOMDRAW

Contains information specific to an **NM_CUSTOMDRAW** notification message sent by a tooltip control.

```
typedef struct tagNMTTCUSTOMDRAW {
    NMCUSTOMDRAW nmcd;
    UINT         uDrawFlags;
} NMTTCUSTOMDRAW, FAR * LPNMTTCUSTOMDRAW;
```

Members

nmcd
NMCUSTOMDRAW structure that contains general custom draw information.

uDrawFlags
UINT value specifying how tooltip text will be formatted when it is displayed. An application may change this field to alter the way text is drawn. This value is passed to the **DrawText** function internally. All values for the *uFormat* parameter of **DrawText** are valid.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 or later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

NMTTDISPINFO

Contains information used in handling the **TTN_GETDISPINFO** notification message. This structure supersedes the **TOOLTIPTEXT** structure.

```
typedef struct tagNMTTDISPINFO {
    NMHDR     hdr;
    LPTSTR    lpszText;
    char      szText[80];
    HINSTANCE hinst;
    UINT      uFlags;
```

(continued)

(continued)

```
#if (_WIN32_IE >= 0x0300)
    LPARAM    lParam;
#endif
} NMTTDISPINFO, FAR *LPNMTTDISPINFO;

#define TOOLTIPTEXT NMTTDISPINFO
```

Members

hdr

NMHDR structure that contains additional information about the notification message.

lpszText

Address of a null-terminated string that will be displayed as the tooltip text. If **hinst** specifies an instance handle, this member must be the identifier of a string resource.

szText

Buffer that receives the tooltip text. An application can copy the text to this buffer instead of specifying a string address or string resource. For tooltip text that exceeds 80 characters, see the comments in the remarks section of this document.

hinst

Handle to the instance that contains a string resource to be used as the tooltip text. If **lpszText** is the address of the tooltip text string, this member must be NULL.

uFlags

Flags that indicates how to interpret the **idFrom** member of the included **NMHDR** structure.

TTF_IDISHWND

If this flag is set, **idFrom** is the tool's handle. Otherwise, it is the tool's identifier.

TTF_RTLREADING

Windows can be *mirrored* to display languages such as Hebrew or Arabic that read from right to left (RTL). Normally, tooltip text is read in the same direction as the text in its parent window. To have a tooltip read in the opposite direction from its parent window, add the TTF_RTLREADING flag to the **uFlags** member when processing the notification.

TTF_DI_SETITEM

Version 4.70. If you add this flag to **uFlags** while processing the notification, the tooltip control will retain the supplied information and not request it again.

lParam

Version 4.70. Application-defined data associated with the tool.

Remarks

You need to point the **lpszText** array to your own private buffer when the text used in the tooltip exceeds 80 characters in length. The system automatically strips the

ampersand (&) accelerator characters from all strings passed to a tooltip control, unless the control has the **TTS_NOPREFIX** style.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

TOOLINFO

The **TOOLINFO** structure contains information about a tool in a tooltip control.

```
typedef struct tagTOOLINFO{
    UINT      cbSize;
    UINT      uFlags;
    HWND      hwnd;
    WPARAM    uId;
    RECT      rect;
    HINSTANCE hinst;
    LPTSTR    lpszText;
#if (_WIN32_IE >= 0x0300)
    LPARAM lParam;
#endif
} TOOLINFO, NEAR *PTOOLINFO, FAR *LPTOOLINFO;
```

Members

cbSize
Size of this structure, in bytes. This member must be specified.

uFlags
Flags that control the tooltip display. This member can be a combination of the following values:

TTF_ABSOLUTE
Version 4.70. Positions the tooltip window at the same coordinates provided by **TTM_TRACKPOSITION**. This flag must be used with the TTF_TRACK flag.

TTF_CENTERTIP
Centers the tooltip window below the tool specified by the **uId** member.

TTF_IDISHWND
Indicates that the **uId** member is the window handle to the tool. If this flag is not set, **uId** is the tool's identifier.

TTF_RTLREADING
Indicates that the tooltip text will be displayed in the opposite direction to the text in the parent window.

TTF_SUBCLASS
Indicates that the tooltip control should subclass the tool's window to intercept messages, such as WM_MOUSEMOVE. If this flag is not set, you must use the **TTM_RELAYEVENT** message to forward messages to the tooltip control. For a list of messages that a tooltip control processes, see TTM_RELAYEVENT.

TTF_TRACK
Version 4.70. Positions the tooltip window next to the tool to which it corresponds and moves the window according to coordinates supplied by the **TTM_TRACKPOSITION** messages. You must activate this type of tool using the **TTM_TRACKACTIVATE** message.

TTF_TRANSPARENT
Version 4.70. Causes the tooltip control to forward mouse event messages to the parent window. This is limited to mouse events that occur within the bounds of the tooltip window.

hwnd
Handle to the window that contains the tool. If **lpszText** includes the LPSTR_TEXTCALLBACK value, this member identifies the window that receives the **TTN_GETDISPINFO** notification messages.

uId
Application-defined identifier of the tool. If **uFlags** includes the TTF_IDISHWND flag, **uId** must specify the window handle to the tool.

rect
Tool's bounding rectangle coordinates. The coordinates are relative to the upper-left corner of the client area of the window identified by **hwnd**. If **uFlags** includes the TTF_IDISHWND flag, this member is ignored.

hinst
Handle to the instance that contains the string resource for the tool. If **lpszText** specifies the identifier of a string resource, this member is used.

lpszText
Pointer to the buffer that contains the text for the tool, or identifier of the string resource that contains the text. This member is used sometimes to return values. If you need to examine the returned value, **lpszText** must point to a valid buffer of sufficient size. Otherwise, it can be set to NULL. If **lpszText** is set to LPSTR_TEXTCALLBACK, the control sends the **TTN_NEEDTEXT** notification message to the owner window to retrieve the text.

lParam
Version 4.70. A 32-bit, application-defined value that is associated with the tool.

Remarks

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, tooltip text is displayed in the same direction as the text in its parent window. If TTF_RTLREADING is set, tooltip text will read in the opposite direction from the text in the parent window.

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 commctrl.h.

TTHITTESTINFO

Contains information that a tooltip control uses to determine whether a point is in the bounding rectangle of the specified tool. If the point is in the rectangle, the structure receives information about the tool.

```
typedef struct _TT_HITTESTINFO {
    HWND hwnd;
    POINT pt;
    TOOLINFO ti;
} TTHITTESTINFO, FAR * LPHITTESTINFO;
```

Members

hwnd
Handle to the tool or window with the specified tool.

pt
Client coordinates of the point to test.

ti
TOOLINFO structure. If the point specified by **pt** is in the tool specified by **hwnd**, this structure receives information about the tool. The **cbSize** member of this structure must be filled in before sending this message.

Remarks

This structure is used with the **TTM_HITTEST** 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 commctrl.h.

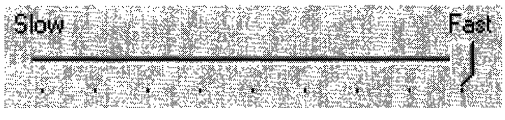
CHAPTER 26

Trackbar Controls

A *trackbar* is a window that contains a slider and optional tick marks. When the user moves the slider, using either the mouse or the direction keys, the trackbar sends notification messages to indicate the change.

About Trackbar Controls

Trackbars are useful when you want the user to select a discrete value or a set of consecutive values in a range. For example, you might use a trackbar to allow the user to set the repeat rate of the keyboard by moving the slider to a given tick mark. The following illustration shows a typical trackbar.



The slider in a trackbar moves in increments that you specify when you create it. This range of values is referred to as "logical units." For example, if you specify that the trackbar should have logical units that range from zero to five, the slider can occupy only six positions: a position at the left side of the trackbar and one position for each increment in the range. Typically, each of these positions is identified by a tick mark.

You create a trackbar by using the **CreateWindowEx** function, specifying the TRACKBAR_CLASS window class. After you have created a trackbar, you can use trackbar messages to set and retrieve many of its properties. Changes that you can make include setting the minimum and maximum positions for the slider, drawing tick marks, setting a selection range, and repositioning the slider.

Trackbar Messages

The logical units of a trackbar are the set of contiguous values that the trackbar can represent. They are usually defined by specifying the range of possible values with a **TBM_SETRANGE** message when the trackbar is first created. Applications can dynamically alter the range by using the **TBM_SETRANGEMAX** and **TBM_SETRANGEMIN** messages. An application that allows the range to be changed dynamically typically retrieves the final range settings when the user has finished working with the trackbar. To retrieve these settings, use the **TBM_GETRANGEMAX** and **TBM_GETRANGEMIN** messages.

An application can send messages to the trackbar to retrieve information about the window and to change its characteristics. To retrieve the position of the slider (that is, the

value the user has chosen), use the **TBM_GETPOS** message. To set the position of the slider, use the **TBM_SETPOS** message.

A trackbar automatically displays tick marks at its beginning and end, unless you specify the **TBS_NOTICKS** style. You can use the **TBS_AUTOTICKS** style to automatically display additional tick marks at regular intervals along the trackbar. By default, a **TBS_AUTOTICKS** trackbar displays a tick mark at each increment of the trackbar's range. To specify a different interval for the automatic tick marks, send the **TBM_SETTICFREQ** message to the trackbar. For example, you could use this message to display only 10 tick marks in a range of 1 through 100.

To set the position of a single tick mark, send the **TBM_SETTIC** message. A trackbar maintains an array of DWORD values that stores the position of each tick mark. The array does not include the first and last tick marks that the trackbar creates automatically. You can specify an index in this array when you send the **TBM_GETTIC** message to get the position of the corresponding tick mark. Alternatively, you can send the **TBM_GETPTICS** message to get a pointer to the array. The number of elements in the array is equal to two less than the tick count returned by the **TBM_GETNUMTICS** message. This is because the count returned by **TBM_GETNUMTICS** includes the first and last tick marks that are not included in the array. To retrieve the physical position of a tick mark, in client coordinates of the trackbar's window, send the **TBM_GETTICPOS** message. The **TBM_CLEARTICS** message removes all but the first and last of a trackbar's tick marks.

A trackbar's line size determines how far the slider moves in response to keyboard input from the arrow keys, such as the RIGHT ARROW or DOWN ARROW key. To retrieve or set the line size, send the **TBM_GETLINESIZE** and **TBM_SETLINESIZE** messages. The trackbar also sends the **TB_LINEUP** and **TB_LINEDOWN** notification messages to its parent window when the user presses the arrow keys.

A trackbar's page size determines how far the slider moves in response to keyboard input, such as the PAGE UP or PAGE DOWN keys, or mouse input, such as clicks in the trackbar channel. To retrieve or set the page size, send the **TBM_GETPAGESIZE** and **TBM_SETPAGESIZE** messages. The trackbar also sends the **TB_PAGEUP** and **TB_PAGEDOWN** notification messages to its parent window when it receives keyboard or mouse input that scrolls the page. For more information, see *Trackbar Notification Messages*.

An application can send messages to retrieve the dimensions of a trackbar. The **TBM_GETTHUMBRECT** message retrieves the bounding rectangle for the slider. The **TBM_GETTHUMBLENGTH** message retrieves the length of the slider. The **TBM_GETCHANNELRECT** message retrieves the bounding rectangle for the trackbar's channel, which is the area over which the slider moves. It contains the highlight when a range is selected. If a trackbar has the **TBS_FIXEDLENGTH** style, you can send the **TBM_SETTHUMBLENGTH** message to change the length of the slider.

If you create a trackbar with the **TBS_ENABLESELRANGE** style, you can specify a "selection range", which restricts the user to a specified portion of the total range. The logical units do not change, but only a subset of them will be available for use. The

trackbar will highlight the available range and display triangular tick marks at the start and end. Typically, an application handles the trackbar's notification messages and sets the trackbar's selection range according to the user's input.

You retrieve or set the selection range by sending messages to the trackbar. Use the **TBM_SETSEL** message to set the starting and ending positions of a selection. To set just the starting position or just the ending position of a selection, send a **TBM_SETSELSTART** or **TBM_SETSELEND** message. To retrieve the starting or ending positions of a selection range, send a **TBM_GETSELSTART** or **TBM_GETSELEND** messages. To clear a selection range and restore the trackbar to its original range, send the **TBM_CLEARSEL** message.

Trackbar Notification Messages

A trackbar notifies its parent window of user actions by sending the parent **WM_HSCROLL** or **WM_VSCROLL** messages. A trackbar with the **TBS_HORZ** style sends **WM_HSCROLL** messages. A trackbar with the **TBS_VERT** style sends **WM_VSCROLL** messages. The low-order word of the *wParam* parameter of **WM_HSCROLL** or **WM_VSCROLL** contains the notification code. For the **TB_THUMBPOSITION** and **TB_THUMBTRACK** notifications, the high-order word of the *wParam* parameter specifies the position of the slider. For all other notifications, the high-order word is zero; send the **TBM_GETPOS** message to determine the slider position. The *lParam* parameter is the handle of the trackbar.

The system sends the **TB_BOTTOM**, **TB_LINEDOWN**, **TB_LINEUP**, and **TB_TOP** notification messages only when the user interacts with a trackbar by using the keyboard. The **TB_THUMBPOSITION** and **TB_THUMBTRACK** notification messages are only sent when the user is using the mouse. The **TB_ENDTRACK**, **TB_PAGEDOWN**, and **TB_PAGEUP** notification messages are sent in both cases. The following table lists the trackbar notification messages and the events (virtual key codes or mouse events) that cause the notifications to be sent:

Notification message	Reason sent
TB_BOTTOM	VK_END
TB_ENDTRACK	WM_KEYUP (the user released a key that sent a relevant virtual key code)
TB_LINEDOWN	VK_RIGHT or VK_DOWN
TB_LINEUP	VK_LEFT or VK_UP
TB_PAGEDOWN	VK_NEXT (the user clicked the channel below or to the right of the slider)
TB_PAGEUP	VK_PRIOR (the user clicked the channel above or to the left of the slider)
TB_THUMBPOSITION	WM_LBUTTONUP following a TB_THUMBTRACK notification message
TB_THUMBTRACK	Slider movement (the user dragged the slider)
TB_TOP	VK_HOME

Default Trackbar Message Processing

This section describes the window message processing performed by a trackbar.

Message	Processing performed
WM_CAPTURECHANGED	Kills the timer if one was set during **WM_LBUTTONDOWN** processing and sends the **TB_THUMBPOSITION** notification message, if necessary. It always sends the **TB_ENDTRACK** notification message.
WM_CREATE	Performs additional initialization, such as setting the line size, page size, and tick mark frequency to default values.
WM_DESTROY	Frees resources.
WM_ENABLE	Repaints the trackbar window.
WM_ERASEBKGND	Erases the window background, using the current background color for the trackbar.
WM_GETDLGCODE	Returns the DLGC_WANTARROWS value.
WM_KEYDOWN	Processes the direction keys and sends the **TB_TOP**, **TB_BOTTOM**, **TB_PAGEUP**, **TB_PAGEDOWN**, **TB_LINEUP**, and **TB_LINEDOWN** notification messages, as appropriate.
WM_KEYUP	Sends the **TB_ENDTRACK** notification message if the key was one of the direction keys.
WM_KILLFOCUS	Repaints the trackbar window.
WM_LBUTTONDOWN	Sets the focus and the mouse capture to the trackbar. When necessary, it sets a timer that determines how quickly the slider moves toward the mouse cursor when the user holds down the mouse button in the window.
WM_LBUTTONUP	Releases the mouse capture and kills the timer if one was set during **WM_LBUTTONDOWN** processing. It sends the **TB_THUMBPOSITION** notification message, if necessary. It always sends the **TB_ENDTRACK** notification message.
WM_MOUSEMOVE	Moves the slider and sends the **TB_THUMBTRACK** notification message when tracking the mouse (see ***WM_TIMER***).
WM_PAINT	Paints the trackbar. If the *wParam* parameter is non-NULL, the control assumes that the value is an HDC and paints using that device context.
WM_SETFOCUS	Repaints the trackbar window.

Message	Processing performed
WM_SIZE	Sets the dimensions of the trackbar, removing the slider if there is not enough room to display it.
WM_TIMER	Retrieves the mouse position and updates the position of the slider. (It is received only when the user is dragging the slider.)
WM_WININICHANGE	Initializes slider dimensions.

Using Trackbar Controls

This section provides examples that demonstrate how to create a trackbar and process trackbar notification messages.

Creating a Trackbar

The following example shows how to create a trackbar with the **TBS_AUTOTICKS** and **TBS_ENABLESELRANGE** styles. When the trackbar is created, both its range and its selection range are initialized. The page size is also set at this time.

```
        // CreateTrackbar - creates and initializes a
        //     trackbar.
        //
        // Global variable
        //     g_hinst - instance handle
        HWND WINAPI CreateTrackbar(
    HWND hwndDlg,  // handle of dialog box (parent window)
    UINT iMin,     // minimum value in trackbar range
    UINT iMax,     // maximum value in trackbar range
    UINT iSelMin,  // minimum value in trackbar selection
    UINT iSelMax)  // maximum value in trackbar selection
  {

    InitCommonControls(); // loads common control's DLL

    hwndTrack = CreateWindowEx(
        0,                              // no extended
styles
        TRACKBAR_CLASS,                 // class name
        "Trackbar Control",             // title (caption)
        WS_CHILD | WS_VISIBLE |
        TBS_AUTOTICKS | TBS_ENABLESELRANGE,  // style
        10, 10,                         // position
        200, 30,                        // size
```

(continued)

(continued)

```
        hwndDlg,                    // parent window
        ID_TRACKBAR,              // control identifier
        g_hinst,                    // instance
        NULL                        // no WM_CREATE
parameter
        );

    SendMessage(hwndTrack, TBM_SETRANGE,
        (WPARAM) TRUE,              // redraw flag
        (LPARAM) MAKELONG(iMin, iMax));  // min. & max.
positions
    SendMessage(hwndTrack, TBM_SETPAGESIZE,
        0, (LPARAM) 4);             // new page size

    SendMessage(hwndTrack, TBM_SETSEL,
        (WPARAM) FALSE,             // redraw flag
        (LPARAM) MAKELONG(iSelMin, iSelMax));
    SendMessage(hwndTrack, TBM_SETPOS,
        (WPARAM) TRUE,              // redraw flag
        (LPARAM) iSelMin);

    SetFocus(hwndTrack);

    return hwndTrack;
  }
```

Processing Trackbar Notification Messages

The following example is a function that is called whenever a **WM_HSCROLL** message is received by the dialog box containing the trackbar. The trackbar has the **TBS_ENABLESELRANGE** style. The position of the slider is compared against the selection range, and the slider is moved to the starting or ending position of the selection range, when necessary.

A dialog containing a trackbar with the **TBS_VERT** style could use this function when it receives a **WM_VSCROLL** message.

```
    // TBNotifications - handles trackbar
    //     notifications received in the wParam
    //     parameter of WM_HSCROLL. This function
    //     simply ensures that the slider remains
    //     within the selection range.

  VOID WINAPI TBNotifications(
  WPARAM wParam,  // wParam of WM_HSCROLL message
  HWND hwndTrack, // handle of trackbar window
```

```
    UINT iSelMin,   // minimum value of trackbar selection
    UINT iSelMax)   // maximum value of trackbar selection
    {
    DWORD dwPos;    // current position of slider

    switch (LOWORD(wParam)) {
        case TB_ENDTRACK:
            dwPos = SendMessage(hwndTrack, TBM_GETPOS, 0,
0);
            if (dwPos > iSelMax)
                SendMessage(hwndTrack, TBM_SETPOS,
                    (WPARAM) TRUE,       // redraw flag
                    (LPARAM) iSelMax);
            else if (dwPos < iSelMin)
                SendMessage(hwndTrack, TBM_SETPOS,
                    (WPARAM) TRUE,       // redraw flag
                    (LPARAM) iSelMin);
            break;

        default:
            break;
```

CDRF_NEWFONT
: Your application specified a new font for the item. The control will use the new font. For more information about changing fonts, see *Changing Fonts and Colors*.

CDRF_SKIPDEFAULT
: Your application drew the item manually. The control will not draw the item.

Trackbar Control Updates in Internet Explorer

Trackbar controls in Microsoft Internet Explorer support the following new features.

Buddy Windows
: Trackbar controls now provide support for up to two buddy windows. Trackbar buddy windows are automatically positioned by the control to appear centered at the ends of the trackbar control. To assign an existing window to a trackbar, use the **TBM_SETBUDDY** message. To retrieve the handle to a given buddy window, send the **TBM_GETBUDDY** message.

Tooltips
: Trackbar controls now support tooltips. A trackbar creates a default tooltip control when created with the **TBS_TOOLTIPS** style. However, you can assign a new tooltip control to a trackbar by sending the **TBM_SETTOOLTIPS** message. To retrieve the handle to an assigned tooltip control, use the **TBM_GETTOOLTIPS** message.

Trackbar Control Styles

This section contains information about the styles used with trackbar controls.

TBS_AUTOTICKS
The trackbar control will have a tick mark for each increment in its range of values.

TBS_BOTH
The trackbar control will display tick marks on both sides of the control. This will be both top and bottom when used with TBS_HORZ or both left and right if used with TBS_VERT.

TBS_BOTTOM
The trackbar control will display tick marks below the control. This style is only valid with TBS_HORZ.

TBS_ENABLESELRANGE
The trackbar control can display a selection range only. The tick marks at the starting and ending positions of a selection range are displayed as triangles (instead of vertical dashes), and the selection range is highlighted.

TBS_FIXEDLENGTH
The trackbar control allows the size of the slider to be changed with the **TBM_SETTHUMBLENGTH** message.

TBS_HORZ
The trackbar control will be oriented vertically. This is the default orientation.

TBS_LEFT
The trackbar control will display tick marks to the left of the control. This style is only valid with TBS_VERT.

TBS_NOTHUMB
The trackbar control does not display a slider.

TBS_NOTICKS
The trackbar control will not display any tick marks.

TBS_REVERSED
Version 5.80. This style bit is used for "reversed" trackbars, where a smaller number indicates "higher" and a larger number indicates "lower". It has no effect on the control, but is simply a label that can be checked to determine whether a trackbar is normal or reversed.

TBS_RIGHT
The trackbar control will display tick marks to the right of the control. This style is only valid with TBS_VERT.

TBS_TOOLTIPS
Version 4.70. The trackbar control will support tooltips. When a trackbar control is created using this style, it automatically creates a default tooltip control that displays the slider's current position. You can change where the tooltips are displayed by using the **TBM_SETTIPSIDE** message.

TBS_TOP
The trackbar control will display tick marks above the control. This style is only valid with TBS_HORZ.

TBS_VERT
The trackbar control will be oriented vertically. This is the default orientation.

Custom Draw Values

Trackbar controls use the following values to identify a control's parts. One of these values is specified in the **dwItemSpec** member of the **NMCUSTOMDRAW** structure.

TBCD_CHANNEL	Identifies the channel that the trackbar control's thumb marker slides along. This is the part of the control that the user moves.
TBCD_THUMB	Identifies the trackbar control's thumb marker.
TBCD_TICS	Identifies the tick marks that are displayed along the trackbar control's edge.

Trackbar Control Reference

Trackbar Control Messages

TBM_CLEARSEL

Clears the current selection range in a trackbar.

```
TBM_CLEARSEL
    wParam = (WPARAM) (BOOL) fRedraw;
    lParam = 0;
```

Parameters

fRedraw
Redraw flag. If this parameter is TRUE, the trackbar is redrawn after the selection is cleared.

Return Values

No return value.

Remarks

A trackbar can have a selection range only if you specified the **TBS_ENABLESELRANGE** style when you created it.

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 commctrl.h.

See Also

TBM_SETSEL, **TBM_SETSELEND**, **TBM_SETSELSTART**

TBM_CLEARTICS

Removes the current tick marks from a trackbar. This message does not remove the first and last tick marks, which are created automatically by the trackbar.

```
TBM_CLEARTICS
    wParam = (WPARAM) (BOOL) fRedraw;
    lParam = 0;
```

Parameters

fRedraw
Redraw flag. If this parameter is TRUE, the trackbar is redrawn after the tick marks are cleared. If this parameter is FALSE, the message clears the tick marks but does not redraw the trackbar.

Return Values

No return value.

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 commctrl.h.

TBM_GETBUDDY

Retrieves the handle to a trackbar control buddy window at a given location. The specified location is relative to the control's orientation (horizontal or vertical).

```
TBM_GETBUDDY
    wParam = (WPARAM)(BOOL) fLocation;
    lParam = 0;
```

Parameters

fLocation

Value indicating which buddy window handle will be retrieved, by relative location. This value can be one of the following:

TRUE	Retrieves the handle to the buddy to the left of the trackbar. If the trackbar control uses the TBS_VERT style, the message will retrieve the buddy above the trackbar.
FALSE	Retrieves the handle to the buddy to the right of the trackbar. If the trackbar control uses the TBS_VERT style, the message will retrieve the buddy below the trackbar.

Return Values

Returns the handle to the buddy window at the location specified by *fLocation*, or NULL if no buddy window exists at that location.

Requirements

Version 4.70 and later of Comctl32.dll

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

TBM_GETCHANNELRECT

Retrieves the size and position of the bounding rectangle for a trackbar's channel. (The channel is the area over which the slider moves. It contains the highlight when a range is selected.)

```
TBM_GETCHANNELRECT
    wParam = 0;
    lParam = (LPARAM) (LPRECT) lprc;
```

Parameters

lprc

Address of a **RECT** structure. The message fills this structure with the channel's bounding rectangle, in client coordinates of the trackbar's window.

Return Values

No return value.

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 commctrl.h.

TBM_GETLINESIZE

Retrieves the number of logical positions the trackbar's slider moves in response to keyboard input from the arrow keys, such as the RIGHT ARROW or DOWN ARROW keys. The logical positions are the integer increments in the trackbar's range of minimum to maximum slider positions.

```
TBM_GETLINESIZE
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a 32-bit value that specifies the line size for the trackbar.

Remarks

The default setting for the line size is 1.

The trackbar also sends the **TB_LINEUP** and **TB_LINEDOWN** notification messages to its parent window when the user presses the arrow keys.

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 commctrl.h.

See Also

TBM_SETLINESIZE

TBM_GETNUMTICS

Retrieves the number of tick marks in a trackbar.

```
TBM_GETNUMTICS
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the number of tick marks.

Remarks

The TBM_GETNUMTICS message counts all of the tick marks, including the first and last tick marks created by the trackbar.

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 commctrl.h.

TBM_GETPAGESIZE

Retrieves the number of logical positions the trackbar's slider moves in response to keyboard input, such as the PAGE UP or PAGE DOWN keys, or mouse input, such as clicks in the trackbar's channel. The logical positions are the integer increments in the trackbar's range of minimum to maximum slider positions.

```
TBM_GETPAGESIZE
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a 32-bit value that specifies the page size for the trackbar.

Remarks

The trackbar also sends the TB_PAGEUP and TB_PAGEDOWN notification messages to its parent window when it receives keyboard or mouse input that scrolls the page.

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 commctrl.h.

See Also

TBM_SETPAGESIZE

TBM_GETPOS

Retrieves the current logical position of the slider in a trackbar. The logical positions are the integer values in the trackbar's range of minimum to maximum slider positions.

```
TBM_GETPOS
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a 32-bit value that specifies the current logical position of the trackbar's slider.

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 commctrl.h.

See Also

TBM_SETPOS

TBM_GETPTICS

Retrieves the address of an array that contains the positions of the tick marks for a trackbar.

```
TBM_GETPTICS
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the address of an array of DWORD values. The elements of the array specify the logical positions of the trackbar's tick marks, not including the first and last tick marks created by the trackbar. The logical positions can be any of the integer values in the trackbar's range of minimum to maximum slider positions.

Remarks

The number of elements in the array is two less than the tick count returned by the **TBM_GETNUMTICS** message. Note that the values in the array may include duplicate positions and may not be in sequential order. The returned pointer is valid until you change the trackbar's tick marks.

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 commctrl.h.

TBM_GETRANGEMAX

Retrieves the maximum position for the slider in a trackbar.

```
TBM_GETRANGEMAX
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a 32-bit value that specifies the maximum position in the trackbar's range of minimum to maximum slider positions.

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 commctrl.h.

See Also

TBM_GETRANGEMIN, TBM_SETRANGE, TBM_SETRANGEMAX

TBM_GETRANGEMIN

Retrieves the minimum position for the slider in a trackbar.

```
TBM_GETRANGEMIN
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a 32-bit value that specifies the minimum position in the trackbar's range of minimum to maximum slider positions.

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 commctrl.h.

See Also

TBM_GETRANGEMAX, TBM_SETRANGE, TBM_SETRANGEMAX

TBM_GETRANGEMIN

Retrieves the minimum position for the slider in a trackbar.

```
TBM_GETRANGEMIN
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a 32-bit value that specifies the minimum position in the trackbar's range of minimum to maximum slider positions.

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 commctrl.h.

See Also

TBM_GETRANGEMAX, TBM_SETRANGE, TBM_SETRANGEMAX

TBM_GETSELEND

Retrieves the ending position of the current selection range in a trackbar.

```
TBM_GETSELEND
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a 32-bit value that specifies the ending position of the current selection range.

Remarks

A trackbar can have a selection range only if you specified the **TBS_ENABLESELRANGE** style when you created it.

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 commctrl.h.

TBM_GETSELSTART, **TBM_SETSEL**, **TBM_SETSELEND**, **TBM_SETSELSTART**

TBM_GETSELSTART

Retrieves the starting position of the current selection range in a trackbar.

```
TBM_GETSELSTART
    wParam = 0;
    lParam = 0;
```

Return Values

Returns a 32-bit value that specifies the starting position of the current selection range.

Remarks

A trackbar can have a selection range only if you specified the **TBS_ENABLESELRANGE** style when you created it.

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 commctrl.h.

TBM_GETSELEND, **TBM_SETSEL**, **TBM_SETSELEND**, **TBM_SETSELSTART**

TBM_GETTHUMBLENGTH

Retrieves the length of the slider in a trackbar.

```
TBM_GETTHUMBLENGTH
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the length, in pixels, of the slider.

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 commctrl.h.

See Also

TBM_GETTHUMBRECT, **TBM_SETTHUMBLENGTH**

TBM_GETTHUMBRECT

Retrieves the size and position of the bounding rectangle for the slider in a trackbar.

```
TBM_GETTHUMBRECT
    wParam = 0;
    lParam = (LPARAM) (LPRECT) lprc;
```

Parameters

lprc

Address of a **RECT** structure. The message fills this structure with the bounding rectangle of the trackbar's slider, in client coordinates of the trackbar's window.

Return Values

No return value.

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 commctrl.h.

TBM_GETTIC

Retrieves the logical position of a tick mark in a trackbar. The logical position can be any of the integer values in the trackbar's range of minimum to maximum slider positions.

```
TBM_GETTIC
    wParam = (WPARAM) (WORD) iTic;
    lParam = 0;
```

Parameters

iTic

Zero-based index identifying a tick mark. Valid indexes are in the range from zero to two less than the tick count returned by the **TBM_GETNUMTICS** message.

Return Values

Returns the logical position of the specified tick mark, or -1 if *iTic* does not specify a valid index.

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 commctrl.h.

TBM_GETTICPOS

Retrieves the current physical position of a tick mark in a trackbar.

```
TBM_GETTICPOS
    wParam = (WPARAM) (WORD) iTic;
    lParam = 0;
```

Parameters

iTic

Zero-based index identifying a tick mark. Valid indexes are in the range from zero to two less than the tick count returned by the **TBM_GETNUMTICS** message.

Return Values

Returns the distance, in client coordinates, from the left or top of the trackbar's client area to the specified tick mark. The return value is the x-coordinate of the tick mark for a horizontal trackbar or the y-coordinate for a vertical trackbar. If *iTic* is not a valid index, the return value is –1.

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 commctrl.h.

TBM_GETTOOLTIPS

Retrieves the handle to the tooltip control assigned to the trackbar, if any.

```
TBM_GETTOOLTIPS
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the handle to the tooltip control assigned to the trackbar, or NULL if tooltips are not in use. If the trackbar control does not use the **TBS_TOOLTIPS** style, the return value is NULL.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

TBM_GETUNICODEFORMAT

Retrieves the UNICODE character format flag for the control.

```
TBM_GETUNICODEFORMAT
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the UNICODE format flag for the control. If this value is nonzero, the control is using UNICODE characters. If this value is zero, the control is using ANSI characters.

Remarks

See the remarks for **CCM_GETUNICODEFORMAT** for a discussion of this message.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

See Also

TBM_SETUNICODEFORMAT

TBM_SETBUDDY

Assigns a window as the buddy window for a trackbar control. Trackbar buddy windows are automatically displayed in a location relative to the control's orientation (horizontal or vertical).

```
TBM_SETBUDDY
    wParam = (WPARAM)(BOOL) fLocation;
    lParam = (LPARAM)(HWND) hwndBuddy;
```

Parameters

fLocation

Value specifying the location at which to display the buddy window. This value can be one of the following:

TRUE	The buddy will appear to the left of the trackbar if the trackbar control uses the **TBS_HORZ** style. If the trackbar uses the **TBS_VERT** style, the buddy appears above the trackbar control.
FALSE	The buddy will appear to the right of the trackbar if the trackbar control uses the **TBS_HORZ** style. If the trackbar uses the **TBS_VERT** style, the buddy appears below the trackbar control.

hwndBuddy

Handle to the window that will be set as the trackbar control's buddy.

Return Values

Returns the handle to the window that was previously assigned to the control at that location.

Remarks

Note Trackbar controls support up to two buddy windows. This can be useful when you must display text or images at each end of the control.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Requires version 2.0 or later.
Header: Declared in commctrl.h.

TBM_SETLINESIZE

Sets the number of logical positions the trackbar's slider moves in response to keyboard input from the arrow keys, such as the RIGHT ARROW or DOWN ARROW keys. The logical positions are the integer increments in the trackbar's range of minimum to maximum slider positions.

```
TBM_SETLINESIZE
    wParam = 0;
    lParam = (LONG) lLineSize;
```

Parameters

lLineSize
New line size.

Return Values

Returns a 32-bit value that specifies the previous line size.

Remarks

The default setting for the line size is 1.

The trackbar also sends the **TB_LINEUP** and **TB_LINEDOWN** notification messages to its parent window when the user presses the arrow keys.

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 commctrl.h.

See Also

TBM_GETLINESIZE

TBM_SETPAGESIZE

Sets the number of logical positions the trackbar's slider moves in response to keyboard input, such as the PAGE UP or PAGE DOWN keys, or mouse input, such as clicks in the trackbar's channel. The logical positions are the integer increments in the trackbar's range of minimum to maximum slider positions.

```
TBM_SETPAGESIZE
    wParam = 0;
    lParam = (LONG) lPageSize;
```

Parameters

lPageSize
New page size.

Return Values

Returns a 32-bit value that specifies the previous page size.

Remarks

The trackbar also sends the **TB_PAGEUP** and **TB_PAGEDOWN** notification messages to its parent window when it receives keyboard or mouse input that scrolls the page.

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 commctrl.h.

See Also

TBM_GETPAGESIZE

TBM_SETPOS

Sets the current logical position of the slider in a trackbar.

```
TBM_SETPOS
    wParam = (WPARAM) (BOOL) fPosition;
    lParam = (LPARAM) (LONG) lPosition;
```

Parameters

fPosition
Redraw flag. If this parameter is TRUE, the message redraws the control with the slider at the position given by *lPosition*. If this parameter is FALSE, the message does

not redraw the slider at the new position. Note that the message sets the value of the slider position (as returned by the **TBM_GETPOS** message) regardless of the *fPosition* parameter.

lPosition

New logical position of the slider. Valid logical positions are the integer values in the trackbar's range of minimum to maximum slider positions. If this value is outside the control's maximum and minimum range, the position is set to the maximum or minimum value.

Return Values

No return value.

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 commctrl.h.

TBM_SETRANGE

Sets the range of minimum and maximum logical positions for the slider in a trackbar.

```
TBM_SETRANGE
   wParam = (WPARAM) (BOOL) fRedraw;
   lParam = (LPARAM) MAKELONG(lMinimum, lMaximum);
```

Parameters

fRedraw

Redraw flag. If this parameter is TRUE, the trackbar is redrawn after the range is set. If this parameter is FALSE, the message sets the range but does not redraw the trackbar.

lMinimum

Minimum position for the slider.

lMaximum

Maximum position for the slider.

Return Values

No return value.

Remarks

If the current slider position is outside the new range, the TBM_SETRANGE message sets the slider position to the new maximum or minimum value.

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 commctrl.h.

See Also

TBM_SETRANGEMAX, TBM_SETRANGEMIN

TBM_SETRANGEMAX

Sets the maximum logical position for the slider in a trackbar.

```
TBM_SETRANGEMAX
    wParam = (WPARAM) fRedraw;
    lParam = (LPARAM) lMaximum;
```

Parameters

fRedraw
Redraw flag. If this parameter is TRUE, the trackbar is redrawn after the range is set. If this parameter is FALSE, the message sets the range but does not redraw the trackbar.

lMaximum
Maximum position for the slider.

Return Values

No return value.

Remarks

If the current slider position is greater than the new maximum, the TBM_SETRANGEMAX message sets the slider position to the new maximum value.

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 commctrl.h.

See Also

TBM_SETRANGE, TBM_SETRANGEMIN

TBM_SETRANGEMIN

Sets the minimum logical position for the slider in a trackbar.

```
TBM_SETRANGEMIN
    wParam = (WPARAM) fRedraw;
    lParam = (LPARAM) lMinimum;
```

Parameters

fRedraw
Redraw flag. If this parameter is TRUE, the message redraws the trackbar after the range is set. If this parameter is FALSE, the message sets the range but does not redraw the trackbar.

lMinimum
Minimum position for the slider.

Return Values

No return value.

Remarks

If the current slider position is less than the new minimum, the TBM_SETRANGEMIN message sets the slider position to the new minimum value.

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 commctrl.h.

See Also

TBM_SETRANGE, TBM_SETRANGEMAX

TBM_SETSEL

Sets the starting and ending positions for the available selection range in a trackbar.

```
TBM_SETSEL
    wParam = (WPARAM) (BOOL) fRedraw;
    lParam = (LPARAM) MAKELONG(lMinimum, lMaximum);
```

Parameters

fRedraw
Redraw flag. If this parameter is TRUE, the message redraws the trackbar after the selection range is set. If this parameter is FALSE, the message sets the selection range but does not redraw the trackbar.

lMinimum
Starting logical position for the selection range.

lMaximum
Ending logical position for the selection range.

Return Values

No return value.

Remarks

This message is ignored if the trackbar does not have the **TBS_ENABLESELRANGE** style.

TBM_SETSEL allow you to restrict the pointer to only a portion of the range available to the progress bar.

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 commctrl.h.

TBM_GETSELEND, **TBM_GETSELSTART**, **TBM_SETSELEND**, **TBM_SETSELSTART**

TBM_SETSELEND

Sets the ending logical position of the current selection range in a trackbar. This message is ignored if the trackbar does not have the **TBS_ENABLESELRANGE** style.

```
TBM_SETSELEND
    wParam = (WPARAM) (BOOL) fRedraw;
    lParam = (LPARAM) (LONG) lEnd;
```

Parameters

fRedraw

Redraw flag. If this parameter is TRUE, the message redraws the trackbar after the selection range is set. If this parameter is FALSE, the message sets the selection range but does not redraw the trackbar.

lEnd

Ending logical position of the selection range.

Return Values

No return value.

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 commctrl.h.

See Also

TBM_GETSELEND, **TBM_GETSELSTART**, **TBM_SETSEL**, **TBM_SETSELSTART**

TBM_SETSELSTART

Sets the starting logical position of the current selection range in a trackbar. This message is ignored if the trackbar does not have the **TBS_ENABLESELRANGE** style.

```
TBM_SETSELSTART
    wParam = (WPARAM) (BOOL) fRedraw;
    lParam = (LPARAM) (LONG) lStart;
```

Parameters

fRedraw

Redraw flag. If this parameter is TRUE, the message redraws the trackbar after the selection range is set. If this parameter is FALSE, the message sets the selection range but does not redraw the trackbar.

lStart

Starting position of the selection range.

Return Values

No return value.

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 commctrl.h.

See Also

TBM_GETSELEND, **TBM_GETSELSTART**, **TBM_SETSEL**, **TBM_SETSELEND**

TBM_SETTHUMBLENGTH

Sets the length of the slider in a trackbar. This message is ignored if the trackbar does not have the **TBS_FIXEDLENGTH** style.

```
TBM_SETTHUMBLENGTH
    wParam = (WPARAM) (UINT) iLength;
    lParam = 0;
```

Parameters

iLength
Length, in pixels, of the slider.

Return Values

No return value.

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 commctrl.h.

See Also

TBM_GETTHUMBLENGTH

TBM_SETTIC

Sets a tick mark in a trackbar at the specified logical position.

```
TBM_SETTIC
    wParam = 0;
    lParam = (LPARAM) (LONG) lPosition;
```

Parameters

lPosition

Position of the tick mark. This parameter can be any of the integer values in the trackbar's range of minimum to maximum slider positions.

Return Values

Returns TRUE if the tick mark is set, or FALSE otherwise.

Remarks

A trackbar creates its own first and last tick marks. Do not use this message to set the first and last tick marks.

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 commctrl.h.

See Also

TBM_GETPTICS, TBM_GETTIC

TBM_SETTIPSIDE

Positions a tooltip control used by a trackbar control. Trackbar controls that use the **TBS_TOOLTIPS** style display tooltips.

```
TBM_SETTIPSIDE
  wParam = (WPARAM)(int) fLocation;
  lParam = 0;
```

Parameters

fLocation

Value representing the location at which to display the tooltip control. This value can be one of the following:

TBTS_TOP	The tooltip control will be positioned above the trackbar. This flag is for use with horizontal trackbars.
TBTS_LEFT	The tooltip control will be positioned to the left of the trackbar. This flag is for use with vertical trackbars.
TBTS_BOTTOM	The tooltip control will be positioned below the trackbar. This flag is for use with horizontal trackbars.
TBTS_RIGHT	The tooltip control will be positioned to the right of the trackbar. This flag is for use with vertical trackbars.

Return Values

Returns a value that represents the tooltip control's previous location. The return value equals one of the possible values for *fLocation*.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

TBM_SETTOOLTIPS

Assigns a tooltip control to a trackbar control.

```
TBM_SETTOOLTIPS
    wParam = (WPARAM)(HWND) hwndTT;
    lParam = 0;
```

Parameters

hwndTT
Handle to an existing tooltip control.

Return Values

The return value for this message is not used.

Remarks

When a trackbar control is created with the **TBS_TOOLTIPS** style, it creates a default tooltip control that appears next to the slider, displaying the slider's current position.

Requirements

Version 4.70 and later of Comctl32.dll

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

Trackbar Control Notifications

NM_CUSTOMDRAW (trackbar)

Sent by a trackbar control to notify its parent windows about drawing operations. This notification is sent in the form of a **WM_NOTIFY** message.

```
NM_CUSTOMDRAW
    lpNMCustomDraw = (LPNMCUSTOMDRAW) lParam;
```

Parameters

lpNMCustomDraw

Address of an **NMCUSTOMDRAW** structure that contains information about the drawing operation. The **dwItemSpec** member of this structure will contain one of the Custom Draw Values that indicates which part of the control is being drawn. Trackbar controls insert the following values into the **dwItemSpec** member of this structure to identify the portion of the control being drawn:

TBCD_CHANNEL	Identifies the channel that the trackbar control's thumb marker slides along.
TBCD_THUMB	Identifies the trackbar control's thumb marker. This is the portion of the control that the user moves.
TBCD_TICS	Identifies the increment tick marks that appear along the edge of the trackbar control.

Return Values

The value your application can return depends on the current drawing stage. The **dwDrawStage** member of the associated **NMCUSTOMDRAW** structure holds a value that specifies the drawing stage. You must return one of the following values.

When **dwDrawStage** equals CDDS_PREPAINT:

CDRF_DODEFAULT

The control will draw itself. It will not send any additional **NM_CUSTOMDRAW** messages for this paint cycle.

CDRF_NOTIFYITEMDRAW

The control will notify the parent of any item-related drawing operations. It will send **NM_CUSTOMDRAW** notification messages before and after drawing items.

CDRF_NOTIFYITEMERASE

The control will notify the parent when an item will be erased. It will send **NM_CUSTOMDRAW** notification messages before and after erasing items.

CDRF_NOTIFYPOSTERASE

The control will notify the parent after erasing an item.

CDRF_NOTIFYPOSTPAINT

The control will notify the parent after painting an item.

CDRF_NOTIFYSUBITEMDRAW
Version 4.71. The control will notify the parent when a list view subitem is being drawn.

When **dwDrawStage** equals CDDS_ITEMPREPAINT:

CDRF_NEWFONT
Your application specified a new font for the item; the control will use the new font. For more information on changing fonts, see *Changing Fonts and Colors*.

CDRF_SKIPDEFAULT
Your application drew the item manually. The control will not draw the item.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

See Also

Using Custom Draw

NM_RELEASEDCAPTURE (trackbar)

Notifies a trackbar control's parent window that the control is releasing mouse capture. This notification is sent in the form of a **WM_NOTIFY** message.

```
NM_RELEASEDCAPTURE
    lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh
Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The control ignores the return value from this notification.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

CHAPTER 27

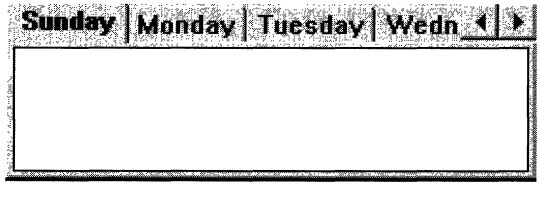
Up-Down Controls

An *up-down control* is a pair of arrow buttons that the user can click to increment or decrement a value, such as a scroll position or a number displayed in a companion control. The value associated with an up-down control is called its *current position*. An up-down control is most often used with a companion control, which is called a *buddy window*.

About Up-Down Controls

To the user, an up-down control and its buddy window often look like a single control. You can specify that an up-down control automatically position itself next to its buddy window and that it automatically set the caption of the buddy window to its current position. For example, you can use an up-down control with an edit control to prompt the user for numeric input. The following illustration shows an up-down control with an edit control as its buddy window, a combination that is sometimes referred to as a *spinner control*.

An up-down control without a buddy window functions as a sort of simplified scroll bar. For example, a tab control sometimes displays an up-down control to enable the user to scroll additional tabs into view. The following illustration shows an up-down control in the upper-right corner of a tab control.



You can create an up-down control and specify its buddy window in several ways. The UPDOWN_CLASS value specifies an up-down control's window class. You can specify this window class in a dialog box template or in a call to the **CreateWindowEx** function. Another way is to use the **CreateUpDownControl** function to create an up-down control and, at the same time, specify its buddy window, current position, and minimum and maximum positions.

The UPDOWN_CLASS window class is registered when the common controls dynamic-link library (DLL) is loaded. If you create an up-down control without using the

CreateUpDownControl function, you must ensure that the DLL is loaded. You can do so by using the **InitCommonControls** function.

CreateUpDownControl enables you to specify a buddy window. If you create an up-down control without using this function, you can assign a buddy window by specifying the **UDS_AUTOBUDDY** window style or by using the **UDM_SETBUDDY** message. If **UDS_AUTOBUDDY** is specified, the up-down control automatically selects the previous window in the z-order as its buddy window. This window might be the previous control in a dialog box template. You can use **UDM_SETBUDDY** to assign a specific buddy window to an up-down control. To determine an up-down control's current buddy window, use the **UDM_GETBUDDY** message. An up-down control and its buddy window must have the same parent window.

An up-down control notifies its parent window when its current position changes by sending it a **UDN_DELTAPOS** notification message and a **WM_VSCROLL** or **WM_HSCROLL** message. A vertical up-down control (which does not have the **UDS_HORZ** style) sends a **WM_VSCROLL** message. A horizontal up-down control (which has the **UDS_HORZ** style) sends a **WM_HSCROLL** message.

About Up-Down Control Styles

Using window styles, you can manipulate characteristics of an up-down control, such as how it positions itself relative to its buddy window, whether it sets the text of its buddy window, and whether it processes the UP ARROW and DOWN ARROW keys.

An up-down control with the **UDS_ALIGNLEFT** or **UDS_ALIGNRIGHT** style aligns with the left or right edge of its buddy window. The width of the buddy window is decreased to accommodate the width of the up-down control.

An up-down control with the **UDS_SETBUDDYINT** style sets the caption of its buddy window whenever the current position changes. The control inserts a thousands separator between every three digits of a decimal string unless the **UDS_NOTHOUSANDS** style is specified. If the buddy window is a list box, an up-down control sets its current selection instead of its caption.

You can specify the **UDS_ARROWKEYS** style to provide a keyboard interface for an up-down control. If this style is specified, the control processes the UP ARROW and DOWN ARROW keys. The control also subclasses the buddy window so that it can process these keys when the buddy window has the focus.

If you use an up-down control for horizontal scrolling, you can specify the **UDS_HORZ** style. This style causes the up-down control's arrows to point left and right instead of up and down.

By default, the current position does not change if the user attempts to increment it or decrement it beyond the maximum or minimum value. You can change this behavior by using the **UDS_WRAP** style, so the position "wraps" to the opposite extreme. For example, incrementing past the upper limit wraps the position back to the lower limit.

Position and Acceleration

After an up-down control is created, you can change the control's current position, minimum position, and maximum position by sending messages. You can also change the radix base used to display the current position in the buddy window and the rate at which the current position changes when the up or down arrow is clicked.

To retrieve the current position of an up-down control, use the **UDM_GETPOS** message. For an up-down control with a buddy window, the current position is the number in the buddy window's caption. Because the caption may have changed (for example, the user may have edited the text of an edit control), the up-down control retrieves the current caption and updates its current position accordingly.

The buddy window's caption may be either a decimal or hexadecimal string, depending on the radix base (that is, either base 10 or 16) of the up-down control. You can set the radix base by using the **UDM_SETBASE** message and retrieve the radix base by using the **UDM_GETBASE** message.

The **UDM_SETPOS** message sets the current position of a buddy window. Note that unlike a scroll bar, an up-down control automatically changes its current position when the up and down arrows are clicked. An application, therefore, does not need to set the current position when processing the **WM_VSCROLL** or **WM_HSCROLL message.**

You can change the minimum and maximum positions of an up-down control by using the **UDM_SETRANGE** message. The maximum position may be less than the minimum, and in that case clicking the up arrow button *decreases* the current position. To put it another way, up means moving towards the maximum position. To retrieve the minimum and maximum positions for an up-down control, use the **UDM_GETRANGE** message.

You can control the rate at which the position changes when the user holds down an arrow button by setting the up-down control's *acceleration.* The acceleration is defined by an array of **UDACCEL** structures. Each structure specifies a time interval and the number of units by which to increment or decrement at the end of that interval. To set the acceleration, use the **UDM_SETACCEL** message. To retrieve acceleration information, use the **UDM_GETACCEL** message.

Default Up-Down Controls Message Processing

This section describes the standard Windows messages processed by an up-down control.

Message	Processing performed
WM_CREATE	Allocates and initializes a private data structure and saves its address as window data.
WM_DESTROY	Frees data allocated during WM_CREATE processing.
WM_ENABLE	Invalidates the window.

(continued)

(continued)

Message	Processing performed
WM_KEYDOWN	Changes the current position in the case of an UP ARROW or DOWN ARROW key.
WM_KEYUP	Completes the position change.
WM_LBUTTONDOWN	Captures the mouse. If the buddy window is an edit control or list box, it sets the focus to the buddy window. If the mouse is over the up or down button, it begins changing the position and sets a timer.
WM_LBUTTONUP	Completes the position change and releases the mouse capture if the up-down control has captured the mouse. If the buddy window is an edit control, it selects all the text in the edit control.
WM_PAINT	Paints the up-down control. If the *wParam* parameter is non-NULL, the control assumes that the value is an HDC and paints using that device context.
WM_TIMER	Changes the current position if the mouse is being held down over a button and a sufficient interval has elapsed.

Up-Down Control Updates in Internet Explorer

Up-down controls in Microsoft Internet Explorer support the following new feature.

Full 32-bit Range
The up-down control now supports a full 32-bit range. The **UDM_SETRANGE32** and **UDM_GETRANGE32** messages have been added to support this feature. The up-down control uses signed integers for its range, so it is necessary to set the range from -0x7FFFFFFF to +0x7FFFFFFF to utilize the full 32-bit range.

Up-Down Control Styles

The following styles are used when creating up-down controls:

UDS_ALIGNLEFT	Positions the up-down control next to the left edge of the buddy window. The buddy window is moved to the right, and its width is decreased to accommodate the width of the up-down control.
UDS_ALIGNRIGHT	Positions the up-down control next to the right edge of the buddy window. The width of the buddy window is decreased to accommodate the width of the up-down control.

UDS_ARROWKEYS	Causes the up-down control to increment and decrement the position when the UP ARROW and DOWN ARROW keys are pressed.
UDS_AUTOBUDDY	Automatically selects the previous window in the z-order as the up-down control's buddy window.
UDS_HORZ	Causes the up-down control's arrows to point left and right instead of up and down.
UDS_HOTTRACK	The control will exhibit "hot tracking" behavior. That is, it will highlight the up and down arrows on the control as the pointer passes over them. This style requires Windows 98 or Windows 2000. If the system is running Windows 95 or Windows NT 4, the flag is ignored. To check whether or not hot tracking is enabled, call **SystemParametersInfo**.
UDS_NOTHOUSANDS	Does not insert a thousands separator between every three decimal digits.
UDS_SETBUDDYINT	Causes the up-down control to set the text of the buddy window (using the WM_SETTEXT message) when the position changes. The text consists of the position formatted as a decimal or hexadecimal string.
UDS_WRAP	Causes the position to "wrap" if it is incremented or decremented beyond the ending or beginning of the range.

Up-Down Control Reference

Up-Down Control Functions

CreateUpDownControl

Creates an up-down control.

```
HWND CreateUpDownControl(
    DWORD dwStyle,
    int x,
    int y,
    int cx,
    int cy,
    HWND hParent,
    int nID,
    HINSTANCE hIns
    HWND hBuddy,
```

(continued)

(continued)

```
    int nUpper,
    int nLower,
    int nPos
);
```

Parameters

dwStyle
Window styles for the control. This parameter should include the WS_CHILD, WS_BORDER, and WS_VISIBLE styles, and it may include any of the window styles specific to the up-down control.

x
Horizontal coordinate, in client coordinates, of the upper-left corner of the control.

y
Vertical coordinate, in client coordinates, of the upper-left corner of the control.

cx
Width, in pixels, of the up-down control.

cy
Height, in pixels, of the up-down control.

hParent
Handle to the parent window of the up-down control.

nID
Identifier for the up-down control.

hInst
Handle to the module instance of the application creating the up-down control.

hBuddy
Handle to the window associated with the up-down control. If this parameter is NULL, the control has no buddy window.

nUpper
Upper limit (range) of the up-down control.

nLower
Lower limit (range) of the up-down control.

nPos
Position of the control.

Return Values

If the function succeeds, the return value is the window handle to the up-down control. If the function fails, the return value is 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 commctrl.h.
Import Library: comctl32.lib.

Up-Down Control Messages

UDM_GETACCEL

Retrieves acceleration information for an up-down control.

```
UDM_GETACCEL
    wParam = (WPARAM) cAccels;
    lParam = (LPARAM) (LPUDACCEL) paAccels;
```

Parameters

cAccels
Number of elements in the array specified by *paAccels*.

paAccels
Address of an array of **UDACCEL** structures that receive acceleration information.

Return Values

The return value is the number of accelerator structures retrieved.

If the *cAccels* parameter is zero and the *paAccels* parameter is NULL, the return value is the number of accelerators currently set for the control.

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 commctrl.h.

See Also

UDM_SETACCEL

UDM_GETBASE

Retrieves the current radix base (that is, either base 10 or 16) for an up-down control.

```
UDM_GETBASE
    wParam = 0;
    lParam = 0;
```

Return Values

The return value is the current base value.

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 commctrl.h.

UDM_GETBUDDY

Retrieves the handle to the current buddy window.

```
UDM_GETBUDDY
    wParam = 0;
    lParam = 0;
```

Return Values

The return value is the handle to the current buddy 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 commctrl.h.

UDM_GETPOS

Retrieves the current position of an up-down control with 16-bit precision.

```
UDM_GETPOS
    wParam = 0;
    lParam = 0;
```

Return Values

If successful, the high order word will be set to zero, and the low-order word will be set to the control's current position. If an error occurs, the high-order word will be set to a non-zero value.

Remarks

When processing this message, the up-down control updates its current position based on the caption of the buddy window. The up-down control returns an error if there is no buddy window or if the caption specifies an invalid or out-of-range value.

If 32-bit values have been enabled for an up-down control with **UDM_SETRANGE32**, this message returns only the lower 16 bits of the position. To retrieve the full 32-bit position, use **UDM_GETPOS32**.

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 commctrl.h.

See Also

UDM_GETRANGE, UDM_GETRANGE32, UDM_SETPOS, UDM_SETRANGE32

UDM_GETRANGE

Retrieves the minimum and maximum positions (range) for an up-down control.

```
UDM_GETRANGE
    wParam = 0;
    lParam = 0;
```

Return Values

The return value is a 32-bit value that contains the minimum and maximum positions. The low-order word is the maximum position for the control, and the high-order word is the minimum position.

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 commctrl.h.

UDM_GETRANGE32

Retrieves the 32-bit range of an up-down control.

```
UDM_GETRANGE32
    wParam = (WPARAM)(LPINT) pLow;
    lParam = (LPARAM)(LPINT) pHigh;
```

Parameters

pLow
Address of a signed integer that receives the lower limit of the up-down control range. This parameter may be NULL.

pHigh
Address of a signed integer that receives the upper limit of the up-down control range. This parameter may be NULL.

Return Values

The return value for this message is not used.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

UDM_GETUNICODEFORMAT

Retrieves the UNICODE character format flag for the control.

```
UDM_GETUNICODEFORMAT
    wParam = 0;
    lParam = 0;
```

Return Values

Returns the UNICODE format flag for the control. If this value is nonzero, the control is using UNICODE characters. If this value is zero, the control is using ANSI characters.

Remarks

See the remarks for **CCM_GETUNICODEFORMAT** for a discussion of this message.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

See Also

UDM_SETUNICODEFORMAT

UDM_SETACCEL

Sets the acceleration for an up-down control.

```
UDM_SETACCEL
    wParam = (WPARAM) nAccels;
    lParam = (LPARAM) (LPUDACCEL) aAccels;
```

Parameters

nAccels
Number of **UDACCEL** structures specified by *aAccels*.

aAccels
Address of an array of **UDACCEL** structures that contain acceleration information. Elements should be sorted in ascending order based on the **nSec** member.

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: Requires version 1.0 or later.
Header: Declared in commctrl.h.

See Also

UDM_GETACCEL

UDM_SETBASE

Sets the radix base for an up-down control. The base value determines whether the buddy window displays numbers in decimal or hexadecimal digits. Hexadecimal numbers are always unsigned, and decimal numbers are signed.

```
UDM_SETBASE
    wParam = (WPARAM) nBase;
    lParam = 0;
```

Parameters

nBase

New base value for the control. This parameter can be 10 for decimal or 16 for hexadecimal.

Return Values

The return value is the previous base value. If an invalid base is given, the return value is zero.

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 commctrl.h.

UDM_SETBUDDY

Sets the buddy window for an up-down control.

```
UDM_SETBUDDY
    wParam = (WPARAM) (HWND) hwndBuddy;
    lParam = 0;
```

Parameters

hwndBuddy

Handle to the new buddy window.

Return Values

The return value is the handle to the previous buddy 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 commctrl.h.

UDM_SETPOS

Sets the current position for an up-down control with 16-bit precision.

```
UDM_SETPOS
    wParam = 0;
    lParam = (LPARAM) MAKELONG((short) nPos, 0);
```

Parameters

nPos

New position for the up-down control. If the parameter is outside the control's specified range, *nPos* will be set to the nearest valid value.

Return Values

The return value is the previous position.

Remarks

This message only supports 16-bit positions. If 32-bit values have been enabled for an up-down control with **UDM_SETRANGE32**, use **UDM_SETPOS32**.

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 commctrl.h.

See Also

UDM_GETRANGE, **UDM_GETPOS**

UDM_SETRANGE

Sets the minimum and maximum positions (range) for an up-down control.

```
UDM_SETRANGE
    wParam = 0;
    lParam = (LPARAM) MAKELONG((short) nUpper, (short)
nLower);
```

Parameters

nUpper and *nLower*
Maximum position and minimum position for the up-down control. Neither position can be greater than the UD_MAXVAL value or less than the UD_MINVAL value. In addition, the difference between the two positions cannot exceed UD_MAXVAL.

Return Values

No return value.

Remarks

The maximum position can be less than the minimum position. Clicking the up arrow button moves the current position closer to the maximum position, and clicking the down arrow button moves towards the minimum position.

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 commctrl.h.

UDM_SETRANGE32

Sets the 32-bit range of an up-down control.

```
UDM_SETRANGE32
    wParam = (WPARAM)(int) iLow;
    lParam = (LPARAM)(int) iHigh;
```

Parameters

iLow
Signed integer value that represents the new lower limit of the up-down control range.

iHigh
Signed integer value that represents the new upper limit of the up-down control range.

Return Values

The return value for this message is not used.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

UDM_SETUNICODEFORMAT

Sets the UNICODE character format flag for the control. This message allows you to change the character set used by the control at run time rather than having to re-create the control.

```
UDM_SETUNICODEFORMAT
    wParam = (WPARAM)(BOOL)fUnicode;
    lParam = 0;
```

Parameters

fUnicode
Determines the character set that is used by the control. If this value is nonzero, the control will use UNICODE characters. If this value is zero, the control will use ANSI characters.

Return Values

Returns the previous UNICODE format flag for the control.

Remarks

See the remarks for **CCM_SETUNICODEFORMAT** for a discussion of this message.

Requirements

Version 4.00 and later of Comctl32.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 commctrl.h.

See Also

UDM_GETUNICODEFORMAT

Up-Down Control Notification Messages

NM_RELEASEDCAPTURE (up-down)

Notifies an up-down control's parent window that the control is releasing mouse capture. This notification is sent in the form of a **WM_NOTIFY** message.

```
NM_RELEASEDCAPTURE
    lpnmh = (LPNMHDR) lParam;
```

Parameters

lpnmh

Address of an **NMHDR** structure that contains additional information about this notification message.

Return Values

The control ignores the return value from this notification.

Requirements

Version 4.71 and later of Comctl32.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 commctrl.h.

UDN_DELTAPOS

Sent by the operating system to the parent window of an up-down control when the position of the control is about to change. This happens when the user requests a change in the value by pressing the control's up or down arrow. The UDN_DELTAPOS message is sent in the form of a **WM_NOTIFY** message.

```
UDN_DELTAPOS
    lpnmud = (LPNMUPDOWN) lParam;
```

Parameters

lpnmud

Address of an **NMUPDOWN** structure that contains information about the position change.

The **iPos** member of this structure contains the current position of the control. The **iDelta** member of the structure is a signed integer that contains the proposed change in position. If the user has clicked the up button, this is a positive value. If the user has clicked the down button, this is a negative value.

Return Values

Return nonzero to prevent the change in the control's position, or zero to allow the change.

Remarks

The UDN_DELTAPOS notification is sent before the **WM_VSCROLL** or **WM_HSCROLL** message, which actually changes the control's position. This lets you examine, allow, modify, or disallow the change.

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 commctrl.h.

See Also

WM_COMMAND

Up-Down Control Structures

NMUPDOWN

Contains information specific to up-down control notification messages. It is identical to and replaces the **NM_UPDOWN** structure.

```
typedef struct _NM_UPDOWN {
    NMHDR hdr;
    int   iPos;
    int   iDelta;
} NMUPDOWN, FAR *LPNMUPDOWN;
```

Members

hdr
NMHDR structure that contains additional information about the notification message.

iPos
Signed integer value that represents the up-down control's current position.

iDelta
Signed integer value that represents the proposed change in the up-down control's position.

Requirements

Version 4.70 and later of Comctl32.dll.

Windows NT/2000: Requires Windows 2000 (or Windows NT 4.0 with Internet Explorer 3.0 and later).
Windows 95/98: Requires Windows 98 (or Windows 95 with Internet Explorer 3.0 or later).
Windows CE: Unsupported.
Header: Declared in commctrl.h.

See Also

UDN_DELTAPOS, **WM_NOTIFY**

UDACCEL

Contains acceleration information for an up-down control.

```
typedef struct {
    UINT nSec;
    UINT nInc;
}UDACCEL, FAR *LPUDACCEL;
```

Members

nSec
Amount of elapsed time, in seconds, before the position change increment specified by **nInc** is used.

nInc
Position change increment to use after the time specified by **nSec** elapses.

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 commctrl.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.

APPENDIX B

Index B: Volume 1, Elements Listed Alphabetically

F

G

H

I

J

T

U

V

W

Z

APPENDIX B

Index B: Volume 2, Elements Listed Alphabetically

A

B

C

D

E

F

G

H

I

K

L

M

O

P

Q

R

S

T

U

V

W

APPENDIX B

Index B: Volume 3, Elements Listed Alphabetically

A

B

C

D

E

F

G

H

I

L

M

O

P

R

S

T

U

V

W

X

APPENDIX B

Index B: Volume 4, Elements Listed Alphabetically

A

C

D

E

F

G

H

I

L

M

N

P

R

S

T

U

W

APPENDIX B

Index B: Volume 5, Elements Listed Alphabetically

H

I

M

P

R

S

T

U

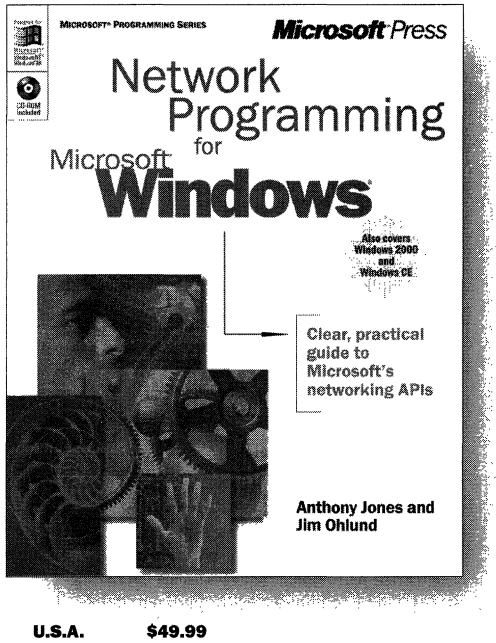
W

Here they are in one place—

practical, detailed explanations

of the Microsoft

networking APIs!



U.S.A. **$49.99**
U.K. £46.99 [V.A.T. included]
Canada $74.99
ISBN 0-7356-0560-2

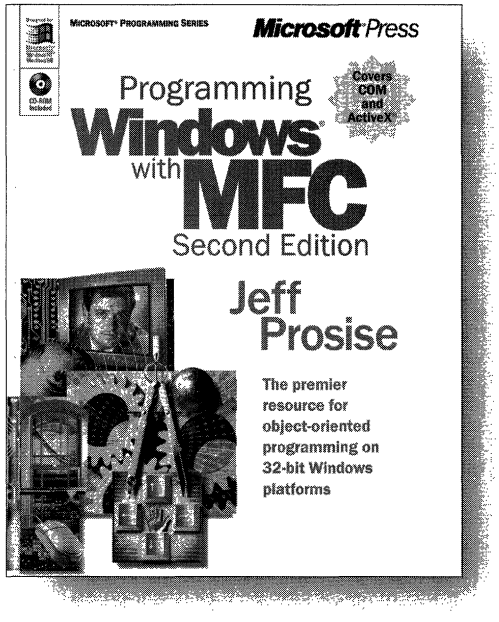
Microsoft has developed many exciting networking technologies, but until now no single source has described how to use them with older, and even some newer, application programming interfaces (APIs). NETWORK PROGRAMMING FOR MICROSOFT® WINDOWS® is the only book that provides definitive, hands-on coverage of how to use legacy networking APIs, such as NetBIOS, on 32-bit platforms, plus recent networking APIs such as Winsock 2 and Remote Access Service (RAS).

Microsoft Press® products are available worldwide wherever quality computer books are sold. For more information, contact your book or computer retailer, software reseller, or local Microsoft Sales Office, or visit our Web site at mspress.microsoft.com. To locate your nearest source for Microsoft Press products, or to order directly, call 1-800-MSPRESS in the U.S. (in Canada, call 1-800-268-2222).

Prices and availability dates are subject to change.

Microsoft®
mspress.microsoft.com

Petzold for the **MFC programmer!**



U.S.A. **$59.99**
U.K. £56.99 [V.A.T. included]
Canada $89.99
ISBN 1-57231-695-0

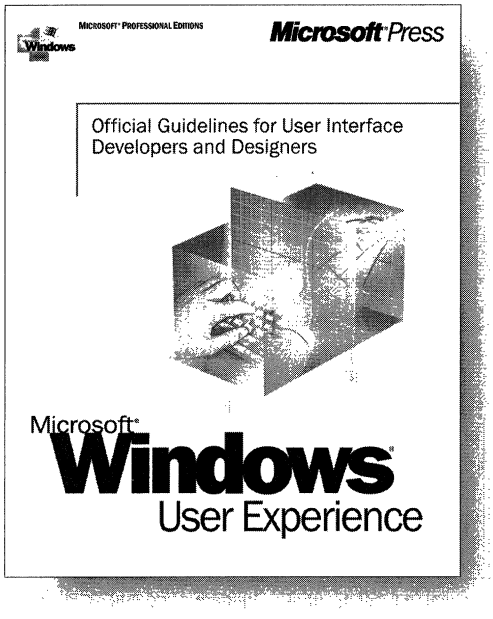
Expanding what's widely considered the definitive exposition of Microsoft's powerful C++ class library for the Windows API, PROGRAMMING WINDOWS® WITH MFC, Second Edition, fully updates the classic original with all-new coverage of COM, OLE, and ActiveX.® Author Jeff Prosise deftly builds your comprehension of underlying concepts and essential techniques for MFC programming with unparalleled expertise—once again delivering the consummate resource for rapid, object-oriented development on 32-bit Windows platforms.

Microsoft Press® products are available worldwide wherever quality computer books are sold. For more information, contact your book or computer retailer, software reseller, or local Microsoft® Sales Office, or visit our Web site at mspress.microsoft.com. To locate your nearest source for Microsoft Press products, or to order directly, call 1-800-MSPRESS in the U.S. (in Canada, call 1-800-268-2222).

Prices and availability dates are subject to change.

Microsoft®
mspress.microsoft.com

Official Guidelines for User Interface Developers and Designers



U.S.A. **$49.99**
U.K. £46.99 [V.A.T. included]
Canada $74.99
ISBN 0-7356-0566-1

Here are the revised, updated, official Microsoft guidelines for creating well-designed, visually and functionally consistent user interfaces for applications that run on the Microsoft Windows family of operating systems, including Windows 98 and Windows 2000. A revision of *The Windows Interface Guidelines for Software Design*, the standard resource for designing Windows interfaces, MICROSOFT WINDOWS USER EXPERIENCE is an essential handbook for all programmers and designers who work with the latest releases of Windows and Microsoft Internet Explorer, regardless of experience level or development tools used. It covers the basic principles of user-interface design and methodologies, and it specifies how you can apply data-centered concepts such as objects and properties to interface design. The book includes detailed information on mouse, keyboard, and other input-device interaction and on how to use the common interface elements supplied by the system. It also includes information about supporting international and disabled users.

Microsoft® Windows® Common Controls

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 Common Controls

This volume provides complete reference materials about using Windows common controls, including the common controls API, creating wizards, customizing a control's appearance, drag list boxes, flat scroll bars, image lists, and list views. It also provides information about controls for animations, date and time picker, headers and hotkeys, toolbars, tabs, tooltips, trackbars, up-down controls, and more.

Part No. 097-0002309

Microsoft