WORDWARE DELPHI DEVELOPER'S LIBRARY

Platinum Edition

"...the definitive reference for the Win32 API..."

Steve Teixeira
Director Core Technology
Zone Labs, Inc. and co-author of Delphi 6 Developer's Guide

"...gives every Delphi developer the knowledge to use the Win32 API powerfully, creatively, and effectively."

Michael Swindell
Director of Product Management
RAD Tools Group, Borland Software Corporation

"...my number one resource when looking for information about how to use the Win32 core API in Delphi."

Bob Swart (a.k.a. "Dr. Bob")
Author, trainer, consultant

The Tomes of Delphi™ Win32 Shell API Windows® 2000 Edition

John Ayres

The Tomes of Delphi™ Win32 Shell API Windows 2000 Edition

John Ayres

Wordware Publishing, Inc.

Library of Congress Cataloging-in-Publication Data

Ayres, John.
Tomes of Delphi : Win32 Shell API—Windows 2000 edition / by John Ayres.
p. cm.
Includes index.
ISBN 1-55622-749-3 (paperback)
1. Microsoft Win32. 2. Delphi (Computer file). I. Title.

QA76.76.063 A98 2002
005.265--dc21 2002004006
CIP

ISBN 1-55622-749-3
10 9 8 7 6 5 4 3 2 1
0204

Dedication

To my second daughter, Victoria Ann Ayres. The entirety of this book was written while she and her mother were still a part of each other. As that miraculous day approaches when I can finally meet her face to face and welcome her to our world, my panic is slowly being replaced with incredible joy and wonder. Children are indeed a miracle, and I am very thankful for this gift from God that we shall soon receive. The responsibility of parenthood can be demanding at times, but it is very rewarding, and I am doubly blessed for the opportunity to once again shape the future of one who is sure to grow into an incredible person. Hero, mentor, role model, playmate, dance partner, confidant, keeper of secrets, and healer of wounds — all these things and much, much more add up to fatherhood. I hope to live up to my daughter's expectations of a father, and may she grow to be more than the sum of her parents.

Contents

Foreword

The Windows API is the foundation upon which most contemporary programs are built. It is the heart and soul of database applications, multimedia applications, even many network based applications. Every Windows application relies on the Windows API to perform everything from the most mundane to the most esoteric task.

All of the good programmers I know have a solid foundation in the Windows API. It is the language in which the architecture of the Windows operating system is most eloquently expressed, and it holds the secrets programmers need to know if they want to develop powerful, well tuned applications.

There are at least three reasons why most serious programmers need to know the Windows API:

1. It is occasionally possible to write strong, robust applications without having a good understanding of the Windows API. However, there comes a time in the course of most application development projects when you simply have to turn to the Windows API in order to solve a particular problem. Usually this happens because a tool you are using does not have a feature you need, or because the feature is not implemented properly. In such cases, you have to turn to the Windows API in order to implement the feature yourself.
2. Another reason to use the Windows API surfaces when you want to create a component or utility that others can use. If you want to build a component, ActiveX control, or simple utility that will perform a useful function needed by other developers or power users, then you probably will need to turn to the Windows API. Without recourse to the Windows API, such projects are usually not feasible.
3. The final and best reason for learning the Windows API is that it helps you see how you should architect your application. We have many high-level tools these days that let us build projects at a very remote, and powerful, level of abstraction. However, each of these tools is built on top of the Windows API, and it is difficult, if not impossible, to understand how to use them without understanding the architecture on which they are founded. If you understand the Windows API then you know what the operating system can do for you, and how it goes about providing that service. With this knowledge under your belt, you can use high-level tools in an intelligent and thoughtful manner.

I am particularly pleased to see the publication of Wordware's books on the Windows API because they are built around the world's greatest development tool: Delphi. Delphi gives you full access to the entire Windows API. It is a tool designed to let you plumb the depths of the operating system, to best utilize the features that have made Windows the preeminent operating system in the world today.

Armed with these books on the Windows API, and a copy of Delphi, you can build any type of application you desire, and can be sure that it is being constructed in the optimal possible manner. No other compiler can bring you closer to the operating system, nor can any other compiler let you take better advantage of the operating system's features. These books are the Rosetta stone which forms the link between Delphi and the Windows API. Readers will be able to use them to create the most powerful applications supported by the operating system. My hat is off to the authors for providing these books as a service to the programming community.

Charles Calvert
former Borland Developer Relations Manager

Acknowledgments

This project would not have been completed without the help of many generous, caring people. In an effort to give credit to those who deserve so much more, the author would like to thank the following people, in no particular order, for their contributions to the book:

Jim Hill, Wes Beckwith, and all the good people down at Wordware Publishing for giving me a platform from which I can do my part to further the overall Delphi community.

Marci Ayres, who performed a lot of code testing, grayscale image conversion, document formatting, and other support functions, along with putting together the help file. This book was written through her entire second pregnancy, and even through all the discomforts that accompany carrying a child, she took over all of my domestic duties so I could concentrate on finishing this book.

Rusty Cornett, who was responsible for introducing me to Delphi and opening up a very rewarding career path.

The authors of the original Tomes of Delphi series, who collectively put in many hours for those first two books.

Of course, no acknowledgment would be complete without thanking the Delphi development staff at Borland for giving all of us such an awesome development tool.

Introduction

The Windows programming environment. No other operating system in history has caused so much controversy or confusion among the programming industry. Of course, no other operating system in history has made so many millionaires either. Like it or not, Windows is here to stay. It's hard to ignore such a large user base, and there are few job opportunities anymore that do not require the programmer to have knowledge of the Windows environment.

In the beginning, a programmer's only choice of tools for creating Windows applications was C/C++. The age of this language has resulted in a wealth of Windows API documentation, filled with abstract and incomplete information, and examples that are as esoteric and arcane as the C language itself. Then along came Delphi. A new era in Windows programming was born, with the ability to easily create complex and advanced Windows applications with a turnaround time unheard of previously. Although Delphi tries its best to insulate the programmer from the underlying Windows architecture, Delphi programmers have found that some programming obstacles simply cannot be overcome without using low-level Windows API functions. Although there have been a few books that touched on the subject of using Windows API functions in Delphi, none have ever discussed the issue in depth. There are numerous magazine articles that describe very specific subsets of the API, but unless the Delphi programmer has a background in C and the time to convert a C example into Delphi, there was simply no recourse of action. Thus, this book was born.

This book is a reference manual for using Windows 32-bit API functions in the Delphi environment. As such, it is not a Windows or Delphi programming tutorial, nor is it a collection of Delphi tricks that solve specific problems. To date, this book is the most complete and accurate reference to the Windows API for the Delphi programmer. It is not a complete reference, as the Windows API includes thousands upon thousands of functions that would fill many volumes much larger than the one you are holding. However, this book covers the most common and important cross section of the Windows API. Additionally, almost every function in this book is available under Windows 95\98\Me and Windows NT\2000\XP (exceptions are noted).

The Chapters

- Chapter 1: Delphi and the Windows API

  This chapter introduces the reader to *The Tomes of Delphi: Win32 Shell API — Windows 2000 Edition*. It covers general Windows programming concerns and techniques, and explains various nuances of programming with the Win32 API in the Delphi environment.

- Chapter 2: Window Movement Functions

  Controlling a window's position can be an important part of a complex user interface. This chapter covers those functions used to control a window's position and size. Examples include moving multiple windows simultaneously and retrieving window positioning and size information.

- Chapter 3: Window Information Functions

  The developer may need to programmatically query a window for some piece of information. This chapter covers functions used to retrieve information on specific windows, such as a window's size, position, and attributes. Examples include subclassing a window and changing window attributes at run time.

- Chapter 4: File Input/Output Functions

  Most applications need the ability to read and write information to an external storage device, and this chapter covers the functions used to manipulate disk-based files. Examples include creating files, manipulating file attributes, reading and writing to a file at the binary level, and performing a file search.

- Chapter 5: Input Functions

  Without the functionality to interpret user input, most applications would be relatively useless. This chapter covers functions used to receive input from the user, such as keyboard and mouse input. Examples include receiving input from the joystick, retrieving information about the keyboard, and manipulating the cursor.

- Chapter 6: String and Atom Functions

  All applications need to display information to the user, which usually takes place in the form of a string. This chapter covers string manipulation functions and functions used to allocate and remove global atoms. Examples include formatting messages, manipulating atoms and strings.

- Chapter 7: Clipboard Functions

  The ability to share information between applications through copy and paste is an expected requirement from Windows users. This chapter covers the functions used to manipulate and view the contents of the clipboard. Examples including enumerating clipboard formats, registering a new clipboard format, and viewing the clipboard contents.

- Chapter 8: System Information Functions

  It may sometimes be useful to retrieve specific information about the system or hardware that is running an application. This chapter covers functions used to query the system for information. Examples include retrieving system hardware information and environment variables, and modifying system parameters.

- Chapter 9: Icon, Cursor, and Caret Functions

  Icons, carets, and cursors are also fundamental graphics objects. This chapter describes the functions used to create and manipulate icons, cursors, and carets. Examples include extracting icons from external files, manipulating the caret, and creating new cursors.

- Chapter 10: Help Functions

  An application without a help system is not a finished application. This chapter covers the two API functions that interface with the help system: WinHelp and HTMLHelp. Several examples are included for both.

- Chapter 11: Shell File Functions

  The Windows environment provides many system-level functions that can be very useful when manipulating files. This chapter covers those functions used to copy and move files and retrieve file information. Examples include creating an appbar application, copying and moving files, and querying file information.

- Chapter 12: Shell Folder Functions

  The Windows namespace includes many special folders that have specific meaning within the shell. This chapter examines the many functions that interact with folders at the shell level. Examples include using the Browse for Folder dialog box and manipulating the recycle bin.

- Chapter 13: Shell Functions

  Windows functionality can be extended through the use of shell hooks and extensions. This chapter discusses methods to implement many common and useful shell extension interfaces. Examples include a dynamic icon extractor and context menu handler.

- Chapter 14: Specialized Shell Functions

  Many shell functions do not fit into a specific category. This chapter examines several unique shell functions that could not be classified into other chapters. Examples include a control panel applet and an appbar.

Conventions

Certain writing conventions have been used throughout this book to convey specific meanings. All example code throughout each chapter appears in a monotype font, such as:

```
function HelloThere(Info: string): Integer;
begin
  ShowMessage(Info);
end;
```

In order to be consistent with other works on Delphi programming, the example code uses Borland's coding conventions, which includes using mixed case for variable names and identifiers, lowercase for reserved words, and nested code indented two spaces per level. Any constants used in the code will appear in all capitals, such as TRUE and FALSE. Also, notice that the name of the unit that contains an individual function is located on the same line as the function name. This unit must be included in the Uses clause of any unit in which this function is used. However, most of the functions covered in this series are located in the Windows.pas file, which is automatically added to the Uses clause by Delphi. In addition, when the text refers to a window, as in a visual object on the screen, the word "window" will begin with a lowercase letter. When the text refers to Windows, as in the operating system, the word "Windows" will be capitalized.

Function Descriptions

The Windows API function descriptions have been laid out in a format that provides an increasing amount of detail to the reader. This should allow the reader to quickly glance at a function description for a simple reminder of required parameters, or to read further for a detailed explanation of the function, an example of its use, and any acceptable constant values used in a parameter.

Each function description includes the exact syntax found in the Delphi source code, a description of what the function does, a detailed list and description of the function's parameters, the value returned from the function, a list of related functions, and an example of its use. Any defined constants used in a function parameter are found in tables that follow the example, so that the descriptive text of the function is not broken by a distraction, and all of the constants are available in one place for easy perusal. Some tables may be repeated under various functions that use the same parameters. This was done to eliminate the need to flip back and forth between several pages while perusing the function descriptions. An asterisk (*) indicates the function is covered in *The Tomes of Delphi: Win32 Core API — Windows 2000 Edition*.

Sample Programs

Although every book reaches a point where the authors are frantically hacking away at the text trying to meet deadlines, I did not want the example code to suffer due to time restraints. Unlike some other books, I wanted to make sure that the example code worked in every case. Therefore, I have taken every effort to ensure that the source code on the companion CD works as expected and that the code found in the book is the exact code found on the CD. This should guarantee that code entered straight from the text will work as described. However, most of the code examples rely on buttons, edit boxes, or other components residing on the form, which may not be apparent from the code listing. When in doubt, always look at the source code included on the CD. Also, bear in mind that some examples may only work under certain conditions; for example, many of the examples demonstrating graphical API calls will only work correctly under a 256-color video mode.

Who This Book is For

Due to the nature of reference manuals and the lack of any involved explanations into general Windows or Delphi programming, this book is intended for use by experienced Delphi programmers with a working knowledge of Windows programming. This is not to say that intermediate or even beginning Delphi programmers will not benefit from this book; in fact, there are quite a few example programs included that solve a number of everyday programming conundrums. The heavily documented examples should provide enough explanation for even the most neophyte Delphi programmer to gain some understanding of the API function being demonstrated. As a reference manual, the book is not intended to be read sequentially from cover to cover. However, the chapters have been laid out in a logical order of progression, starting with the most fundamental Windows API functions and working towards the more specialized functions.

If you are looking for an introduction to Delphi programming, or a step-by-step Windows programming tutorial, there are plenty of other fine books out there to get you started. However, if you've got a nasty problem whose only hope of salvation is using the Windows API, if you want to extend the functionality of Delphi components and objects, or if you want a down-and-dirty, no-holds-barred collection of Delphi Win32 API programming examples, then this book is for you. You will not find a more complete and accurate guide to the Win32 API for the Delphi programmer.

Chapter 1

Delphi and the Windows API

When Delphi was introduced, it brought a new era to Windows programming. Never before was it so easy to create robust, full-featured applications for the Windows environment with such short development times. Now in its sixth incarnation, Delphi has been the development tool for innumerable shareware and freeware applications, internal business and proprietary system applications, several well-known commercial applications, and even a commercial game or two. Delphi's power and ease of use make it a wonderful choice for a development platform that can stand up to C++ and Visual Basic in almost every situation.

One of Delphi's strengths is the Visual Component Library, Borland's object model. This object model has allowed the Delphi development team to encapsulate the vast majority of Windows programming tedium into easy-to-use components. Earlier Windows programming languages required the developer to write large amounts of code just to squeeze a minimal amount of functionality out of Windows. The mere act of creating a window and accepting menu selections could take pages of code to create. Delphi's excellent encapsulation of this dreary requirement of Windows programming has turned what once was a chore into a fun, exciting experience.

Windows Data Types

Windows API functions use a number of data types that may be unfamiliar to the casual Delphi programmer. These data types are all taken from the original C header files that define the Windows API function syntax. For the most part, these new data types are simply Pascal data types that have been renamed to make them similar to the original data types used in legacy Windows programming languages. This was done so that experienced Windows programmers would understand the parameter types and function return values, and the function prototypes would match the syntax shown in existing Windows API documentation to avoid confusion. The following table outlines the most common Windows data types and their correlating Object Pascal data type.

Table 1-1: Windows data types

Windows Data Type	Object Pascal Data Type	Description
LPSTR	PAnsiChar	String pointer
LPCSTR	PAnsiChar	String pointer
DWORD	LongWord	Whole numbers
BOOL	LongBool	Boolean values
PBOOL	^BOOL	Pointer to a Boolean value
PByte	^Byte	Pointer to a byte value
PINT	^Integer	Pointer to an integer value
PSingle	^Single	Pointer to a single (floating-point) value
PWORD	^Word	Pointer to a 16-bit value
PDWORD	^DWORD	Pointer to a 32-bit value
LPDWORD	PDWORD	Pointer to a 32-bit value
UCHAR	Byte	8-bit values (can represent characters)
PUCHAR	^Byte	Pointer to 8-bit values
SHORT	Smallint	Signed 16-bit whole numbers
UINT	LongWord	Unsigned 32-bit whole numbers
PUINT	^UINT	Pointer to unsigned 32-bit whole numbers
ULONG	Cardinal	Unsigned 32-bit whole numbers
PULONG	^ULONG	Pointer to unsigned 32-bit whole numbers
PLongint	^Longint	Pointer to 32-bit values
PInteger	^Integer	Pointer to 32-bit values
PSmallInt	^Smallint	Pointer to 16-bit values
PDouble	^Double	Pointer to double (floating-point) values
LCID	DWORD	A local identifier
LANGID	Word	A language identifier
THandle	LongWord	An object handle. Many Windows API functions return a value of type THandle, which identifies that object within Window's internal object tracking tables.
PHandle	^THandle	A pointer to a handle
WPARAM	Longint	A 32-bit message parameter. Under earlier versions of Windows, this was a 16-bit data type.
LPARAM	Longint	A 32-bit message parameter
LRESULT	Longint	A 32-bit function return value
HWND	LongWord	A handle to a window. All windowed controls, child windows, main windows, etc., have a corresponding window handle that identifies them within Windows' internal tracking tables.
HHOOK	LongWord	A handle to an installed Windows system hook

Windows Data Type	Object Pascal Data Type	Description
ATOM	Word	An index into the local or global atom table for a string
HGLOBAL	THandle	A handle identifying a globally allocated dynamic memory object. Under 32-bit Windows, there is no distinction between globally and locally allocated memory.
HLOCAL	THandle	A handle identifying a locally allocated dynamic memory object. Under 32-bit Windows, there is no distinction between globally and locally allocated memory.
FARPROC	Pointer	A pointer to a procedure, usually used as a parameter type in functions that require a callback function
HGDIOBJ	LongWord	A handle to a GDI object. Pens, device contexts, brushes, etc., all have a handle of this type that identifies them within Window's internal tracking tables.
HBITMAP	LongWord	A handle to a Windows bitmap object
HBRUSH	LongWord	A handle to a Windows brush object
HDC	LongWord	A handle to a device context
HENHMETAFILE	LongWord	A handle to a Windows enhanced metafile object
HFONT	LongWord	A handle to a Windows logical font object
HICON	LongWord	A handle to a Windows icon object
HMENU	LongWord	A handle to a Windows menu object
HMETAFILE	LongWord	A handle to a Windows metafile object
HINST	THandle	A handle to an instance object
HMODULE	HINST	A handle to a module
HPALETTE	LongWord	A handle to a Windows color palette
HPEN	LongWord	A handle to a Windows pen object
HRGN	LongWord	A handle to a Windows region object
HRSRC	THandle	A handle to a Windows resource object
HKL	LongWord	A handle to a keyboard layout
HFILE	LongWord	A handle to an open file
HCURSOR	HICON;	A handle to a Windows mouse cursor object
COLORREF	DWORD;	A Windows color reference value, containing values for the red, green, and blue components of a color

Handles

An important concept in Windows programming is the concept of an object handle. Many functions return a handle to an object that the function created or loaded from a resource. Functions like CreateWindowEx return a window handle. Other functions, like CreateFile, return a handle to an open file or, like HeapCreate, return a handle to a newly allocated heap. Internally, Windows keeps track of all these handles, and the handle serves as the link through the operating system between the object and the application. Using these handles, an application can easily refer to any of these objects, and the operating system instantly knows which object a piece of code wants to manipulate.

Constants

The Windows API functions declare literally thousands upon thousands of different constants to be used as parameter values. Constants for everything from color values to return values have been defined in the Windows.pas, Types.pas, ActiveX.pas, ShlObj.pas, ComObj.pas, and System.pas files. The constants that are defined for each API function are listed with that function within the text. However, the Windows.pas file may yield more information concerning the constants for any particular function, and it is a good rule of thumb to check this Delphi source code file when using complicated functions.

Strings

All Windows API functions that use strings require a pointer to a null-terminated string type. Windows is written in C, which does not have the Pascal string type. Earlier versions of Delphi required the application to allocate a string buffer and convert the string type to a PChar. However, since Delphi 3, we have a string conversion mechanism that allows a string to be used as a PChar by simply typecasting it (i.e., PChar(MyString), where MyString is declared as MyString: string). For the most part, this conversion will work with almost all Windows API functions that require a string parameter.

Importing Windows Functions

The Windows API is huge. It defines functions for almost every kind of utility or comparison or action that a programmer could think of. Due to the sheer volume of Windows API functions, some functions simply fell through the cracks and were not imported by the Delphi source code. Since all Windows API functions are simply functions exported from a DLL, importing a new Windows API function is a relatively simple process, if the function parameters are known.

Importing a new Windows API function is exactly like importing any other function from a DLL. For example, in earlier versions of Delphi, the BroadcastSystemMessage function was not imported by the Delphi source code. (It is now imported and available for use, but we'll use this function as an example.) In order to import this function for use within an application, it is simply declared as a function from within a DLL as:

```
function BroadcastSystemMessage(Flags: DWORD; Recipients: PDWORD;
    uiMessage: UINT; wParam: WPARAM; lParam: LPARAM): Longint; stdcall;

implementation

function BroadcastSystemMessage; external user32 name 'BroadcastSystemMessage';
```

As long as the parameters required by the function and the DLL containing the function are known, any Windows API function can be imported and used by a Delphi application. It is important to note that the stdcall directive must be appended to the prototype for the function, as this defines the standard mechanism by which Windows passes parameters to a function on the stack.

Note: Use the stdcall directive, appended to the end of the function prototype, when importing Windows API functions.

Incorrectly Imported Functions

Some functions have been incorrectly imported by the Delphi source code. These exceptions are noted in the individual function descriptions. For the most part, the functions that have been imported incorrectly deal with the ability to pass NIL as a value to a pointer parameter, usually to retrieve the required size of a buffer so the buffer can be dynamically allocated to the exact length before calling the function to retrieve the real data. In Delphi, some of these functions have been imported with parameters defined as VAR or CONST. These types of parameters can accept a pointer to a buffer but can never be set to NIL, thus limiting the use of the function within the Delphi environment. As is the case with almost anything in Delphi, it is a simple matter to fix. Simply reimport the function as if it did not exist, as outlined above. Functions that have been imported incorrectly are identified in their individual function descriptions throughout this book.

Callback Functions

Another very important concept in Windows programming is that of a callback function. A callback function is a function within the developer's application that is never called directly by any other function or procedure within that application. Instead, it is called by the Windows operating system. This allows Windows to communicate directly with the application, passing it various parameters as defined by the individual callback function. Most of the enumeration functions require some form of application-defined callback function that receives the enumerated information.

Individual callback functions have specific parameters that must be declared exactly by the application. This is required so that Windows passes the correct information to the application in the correct order. A good example of a function that uses a callback function is EnumWindows. The EnumWindows function parses through all top-level windows on the screen, passing the handle of each window to an application-defined callback function. This continues until all top-level windows have been enumerated or the callback function returns FALSE. The callback function used by EnumWindows is defined as:

```
EnumWindowsProc(
hWnd: HWND;                {a handle to a top-level window}
lParam: LPARAM             {the application-defined data}
): BOOL;                   {returns TRUE or FALSE}
```

A function matching this function prototype is created within the application, and a pointer to the function is passed as one of the parameters to the EnumWindows function. The Windows operating system calls this callback function for each top-level window, passing the window's handle in one of the callback function's parameters. It is important to note that the stdcall directive must be appended to the prototype for the callback function, as this defines the standard mechanism by which Windows passes parameters to a function on the stack. For example, the above callback function would be prototyped as:

```
EnumWindowsProc(hWnd: HWND; lParam: LPARAM); stdcall;
```

Without the stdcall directive, Windows will not be able to access the callback function. This powerful software mechanism, in many cases, allows an application to retrieve information about the system that is only stored internally by Windows and would otherwise be unreachable. For a complete example of callback function usage, see the EnumWindows function and many other functions throughout the book.

Function Parameters

The vast majority of Windows API functions simply take the static parameters handed to them and perform some function based on the value of the parameters. However, certain functions return values that must be stored in a buffer, and that buffer is passed to the function in the form of a pointer. In most cases, when the function description specifies that it returns some value in a buffer, null-terminated string buffer, or pointer to a data structure, these buffers and data structures must be allocated by the application before the function is called.

In many cases, a parameter may state that it can contain one or more values from some table. These values are defined as constants, and they are combined using the Boolean OR operator. The actual value passed to the function usually identifies a bitmask, where the state of each bit has some significance to the function. This is why the constants can be combined using Boolean operations. For example, the CreateWindowEx function has a parameter called dwStyle which can accept a number of constants combined with the Boolean OR operator. To pass more than one constant to the function, the parameter would be set to something like "WS_CAPTION or WS_CHILD or WS_CLIPCHILDREN." This would create a child window that includes a caption bar and would clip around its child windows during painting.

Conversely, when a function states that it returns one or more values that are defined as specific constants, the return value can be combined with one of the constants using the Boolean AND operator to determine if that constant is contained within the return value. If the result of the combination equals the value of the constant, then that constant is included in the return value.

Unicode

Originally, software only needed a single byte to define a character within a character set. This allowed for up to 256 characters, which was more than plenty for the entire alphabet, numbers, punctuation symbols, and common mathematical symbols. However, due to the shrinking of the global community and the subsequent internationalization of Windows and Windows software, a new method of identifying characters was needed. Many languages have well over 256 characters used for writing, much more than a single byte can describe. Therefore, Unicode was invented. A Unicode character is 16 bits long and can therefore identify 65,535 characters within a language's alphabet. To accommodate the new character set type, many Windows API functions come in two flavors: ANSI and Unicode. When browsing the Windows.pas source code, many functions are defined with an A or W appended to the end of the function name, identifying them as an ANSI function or Wide character (Unicode) function. The functions within this book cover only the ANSI functions. However, the Unicode functions usually differ only in the type of string information passed to a function, and the text within this book should adequately describe the Unicode function's behavior.

Delphi vs. the Windows API

The Delphi development team did a world-class job of encapsulating the vast majority of important Windows API functionality into the VCL. However, due to the vastness of the Windows API, it would be impossible and impractical to wrap every API function in an Object Pascal object. To achieve certain goals or solve specific problems, a developer may be forced to use lower-level Windows API functions that are simply not encapsulated by a Delphi object. It may also be necessary to extend the functionality of a Delphi object, and if this object encapsulates some part of the Windows API, it will be the API that the developer will likely have to use to extend the functionality by any great amount.

Indeed, there are literally hundreds of APIs out there that dramatically extend Windows' functionality, and due to the sheer numbers of API functions and the ever changing, ever expanding functionality being introduced by Microsoft, it would be nearly impossible to actively import every last function from every available API. Therefore, it is important that the well-prepared and capable Delphi programmer is familiar with hardcore Windows programming, as it is highly likely that you'll be called upon sometime in your Delphi career to make use of some Windows API functionality that is not encapsulated by the VCL.

There may even be situations where it is impractical to use the Delphi components that encapsulate Windows functionality. The VCL makes Windows programming easy, but by its very nature, Delphi applications tend to be a minimum 350 KB in size. Bypassing the VCL and using direct Windows API calls, on the other hand, can yield a

Delphi application as small as 10 KB. Every situation is different, and fortunately, as Delphi programmers, we have a lot of flexibility in this area. Using direct Windows API calls may not always be necessary, but when it is, it's good to know that we have that option available to us.

Chapter 2

Window Movement Functions

The Win32 API includes a group of functions that allow a developer to programmatically control the size and positioning of windows. While a window can be moved or resized easily using its Left, Top, Width, or Height properties, these functions give the developer extended control above and beyond what Delphi encapsulates.

Z-order

Many of the window movement functions are concerned with modifying the z-order of a window. The z-order of a window refers to the order in which the windows overlap each other. It is based on the z-axis, which can be thought of as an imaginary line running into the screen at a 90-degree angle. The windows are stacked according to their position on this z-axis. Those windows that are said to be closer to the top of the z-order overlap and appear on top of other windows, while those that are overlapped are said to be closer to the bottom of the z-order. Figure 2-1 illustrates this concept.

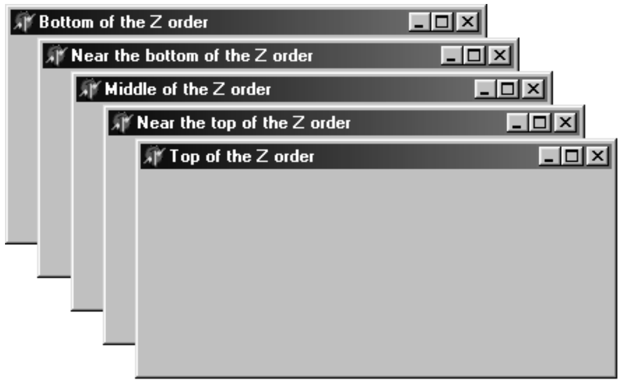


Figure 2-1: The window z-order

Windows maintains the z-order of all windows in a single list. A window's z-order is determined by the order in which it appeared on the screen in relation to the other windows. When a window is created, it is placed above the previously created windows in the z-order, so the first window created is at the bottom of the z-order and the last window created is at the top. However, the window's position in the z-order list is dependent upon its type. All windows can be classified as follows:

- **Topmost windows**: A topmost window overlaps all other non-topmost windows. This is true even if it is not the active or foreground window. This type of window contains the WS_EX_TOPMOST extended style and appears in the z-order before all non-topmost windows.
- **Top-level windows**: A top-level window is any normal window that is not a child window. This type of window does not contain the WS_EX_TOPMOST extended style, and it is always overlapped by and appears below any topmost window in the z-order.
- **Child windows**: A child window contains the WS_CHILD style. Child windows have a z-order amongst themselves within a parent window, but otherwise they reflect the same z-order position as their parent.

When a window is activated, it is brought to the top of the z-order of all windows of the same type, bringing any child windows with it. If a window owns any other windows, those windows are positioned above the activated window in the z-order so they are always displayed above their owners.

Special Effects

Imaginative utilization of the window movement functions can give an application that professional touch. For example, if an application occupies a relatively small amount of screen space but has floating toolbars or other pop-up windows that are constantly open, a developer can use the window movement functions to cause the toolbar windows to move with the main window when the user drags it to a new location. This is a nice effect at the cost of only a few lines of code. The following example demonstrates this technique.

Listing 2-1: Moving a toolbar with its owner window

```
unit WinMoveU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Unit2;

type
  TForm1 = class(TForm)
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
    {we must override the WM_MOVE message}
    procedure WMMove(var Msg: TWMMove); message WM_MOVE;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
```

```
implementation

{$R *.DFM}

procedure TForm1.FormShow(Sender: TObject);
begin
  {show the toolbar window}
  Form2.Show;
end;

{this is fired every time the main window is moved}
procedure TForm1.WMMove(var Msg: TWMMove);
begin
  {if the toolbar window exists...}
  if Form2<>NIL then
    {...move the toolbar window alongside the main window.}
    MoveWindow(Form2.Handle, Form1.Left+Form1.Width+5, Form1.Top, Form2.Width,
               Form2.Height, TRUE);
end;

end.
```

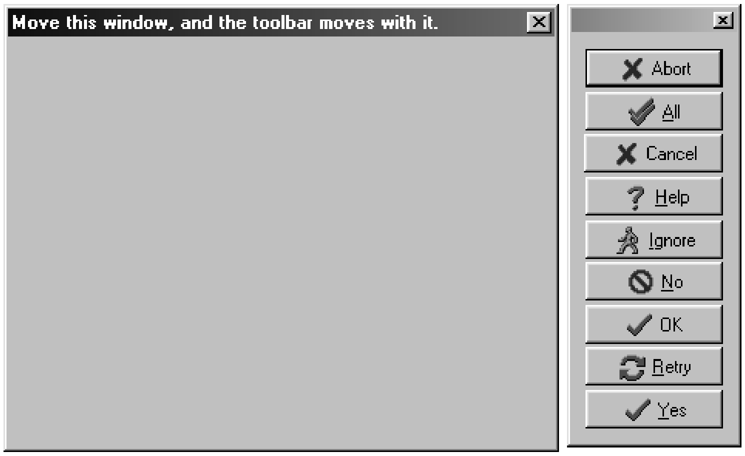


Figure 2-2: The main window and its toolbar

Delphi vs. the Windows API

In most cases, the average Delphi developer may never have a need for these functions. It is easy enough to move and resize a Delphi form by manipulating the left, right, width, and height properties. The TForm object even provides methods that present the same functionality as CascadeWindows, TileWindows, and other MDI child window manipulations. However, there may be times when using the VCL is not practical or is impossible (such as when an application is created using nothing but Windows API function calls), in which case you have to use these functions. You may also be dealing with windows provided by other applications or DLLs, and these functions are the only way to move and resize a window when all you have is a window handle. Plus, several of these functions provide extended functionality that is not available in TForm methods or other Delphi functions and procedures.

Window Movement Functions

The following window movement functions are covered in this chapter.

Table 2-1: Window movement functions

Function	Description
AdjustWindowRect	Calculates the window size based on the desired client area size.
AdjustWindowRectEx	Calculates the size of a window with an extended style based on the desired client area size.
BeginDeferWindowPos	Begins a process of moving multiple windows simultaneously.
BringWindowToTop	Brings the specified window to the top of the z-order.
CascadeWindows	Arranges the specified windows in a cascading format.
CloseWindow	Minimizes the specified window.
DeferWindowPos	Defines a new size and position for the specified window.
EndDeferWindowPos	Ends the process of moving multiple windows simultaneously.
GetWindowPlacement	Retrieves the show state and positioning of the specified window.
MoveWindow	Moves a window.
OpenIcon	Restores a window from a minimized state.
SetWindowPlacement	Sets the show state and positioning of the specified window.
SetWindowPos	Changes the size, position, and z-order of the specified window.
ShowOwnedPopups	Toggles the visibility of all popups owned by the specified window.
ShowWindow	Displays a window.
ShowWindowAsync	Displays a window and immediately returns to the calling function.
TileWindows	Arranges the specified windows in a tiled format.

AdjustWindowRect   Windows.pas

Syntax

```
AdjustWindowRect(
var lpRect: TRect;      {a pointer to the client rectangle structure}
dwStyle: DWORD;         {window style flags}
bMenu: BOOL             {menu flag}
): BOOL;                {returns TRUE or FALSE}
```

Description

AdjustWindowRect calculates a window rectangle size based on the specified client rectangle size in lpRect. The window rectangle will include the size of the border, caption bar, and menu bar. This rectangle can be used with the CreateWindow or CreateWindowEx functions to create a window with the exact desired client area size. The returned coordinates are in terms of top-left and bottom-right screen coordinates, but the CreateWindow function needs these parameters in terms of a top and left coordinate and a window width and height. Therefore, the developer must subtract the left coordinate from the right to get the appropriate width and subtract the top coordinate from the bottom to get the appropriate height.

Parameters

lpRect: The address of a TRect structure that contains the top-left and bottom-right coordinates of the desired client area, relative to the screen. If this function succeeds, this information will be modified to contain the top-left and bottom-right coordinates of a window rectangle containing the specified client area, also relative to the screen.

dwStyle: A 32-bit number representing the window style used by the specified window.

bMenu: If this window has a menu, this parameter should be TRUE; otherwise it should be FALSE.

Return Value

If this function succeeds, it returns TRUE; otherwise it returns FALSE. To get extended error information, call the GetLastError function.

See Also

AdjustWindowRectE, CreateWindow*, CreateWindowEx*

An asterisk (*) indicates the function is covered in *The Tomes of Delphi: Win32 Core API — Windows 2000 Edition*.

Example

Listing 2-2: Creating a window with a client area 300 pixels high and 300 pixels wide

```
procedure TForm1.CreateParams(var Params: TCreateParams);
var
   TheRect: TRect; // stores our rectangle coordinates
begin
   {fill in the standard parameters}
   inherited CreateParams(Params);

   {our window will start at coordinates 100,100 and our client
    rectangle will be 300 pixels high and 300 pixels wide}
   TheRect.Left:=100;
   TheRect.Top:=100;
   TheRect.Right:=400;
   TheRect.Bottom:=400;

   {adjust our rectangular coordinates to get a window with a
    300 by 300 pixel client area}
   AdjustWindowRect(TheRect, Params.Style, FALSE);

   {the results from AdjustWindowRect are in terms of exact coordinates,
    but the CreateWindowEx function needs this in terms of a top and left
    coordinate and a width and height measurement}
   Params.X:=TheRect.Left;
   Params.Y:=TheRect.Top;
   Params.Width:=TheRect.Right-TheRect.Left;     // determine window width
   Params.Height:=TheRect.Bottom-TheRect.Top;    // determine window height

end;
```

Figure 2-3: The result of the AdjustWindowRect function on the window

AdjustWindowRectEx   Windows.pas

Syntax

```
AdjustWindowRectEx(
var lpRect: TRect;          {a pointer to the client rectangle structure}
dwStyle: DWORD;             {window style flags}
bMenu: BOOL                 {menu flag}
dwExStyle: DWORD            {extended style flags}
): BOOL;                    {returns TRUE or FALSE}
```

Description

This calculates a window rectangle size based on the specified client rectangle size in lpRect. The window rectangle will include the size of the border, caption bar, and menu bar. This rectangle can be used with the CreateWindow or CreateWindowEx functions to create a window with the exact desired client area size. The returned coordinates are in terms of top-left and bottom-right screen coordinates, but the CreateWindow function needs these parameters in terms of a top and left coordinate and a window width and height. Therefore, the developer must subtract the left coordinate from the right to get the appropriate width and subtract the top coordinate from the bottom to get the appropriate height. This is functionally equivalent to AdjustWindowRect.

Parameters

lpRect: The address of a TRect structure that contains the top-left and bottom-right coordinates of the desired client area, relative to the screen. If this function succeeds, this information will be modified to contain the top-left and bottom-right coordinates of a window rectangle containing the specified client area, also relative to the screen.

dwStyle: A 32-bit number representing the window style used by the specified window.

bMenu: If this window will have a menu, this parameter should be TRUE; otherwise, it should be FALSE.

dwExStyle: A 32-bit number representing the extended window style used by the specified window.

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

AdjustWindowRect, CreateWindowEx*

Example

Listing 2-3: Giving an extended window style window a client area of 300 x 300 pixels

```
procedure TForm1.CreateParams(var Params: TCreateParams);
var
   TheRect: TRect; // stores our rectangle coordinates
begin
   {fill in the standard parameters}
   inherited CreateParams(Params);

   {our window will start at coordinates 100,100 and our client
    rectangle will be 300 pixels high and 300 pixels wide}
   TheRect.Left:=100;
   TheRect.Top:=100;
   TheRect.Right:=400;
   TheRect.Bottom:=400;

   {adjust our rectangular coordinates to get a window with a
    300 by 300 pixel client area}
   AdjustWindowRectEx(TheRect, Params.Style, FALSE, Params.ExStyle);

   {the results from AdjustWindowRectEx are in terms of exact coordinates,
    but the CreateWindowEx function needs this in terms of a top and left
    coordinate and a width and height measurement}
   Params.X:=TheRect.Left;
   Params.Y:=TheRect.Top;
   Params.Width:=TheRect.Right-TheRect.Left;     // determine window width
   Params.Height:=TheRect.Bottom-TheRect.Top;    // determine window height

end;
```

BeginDeferWindowPos   Windows.pas

Syntax

```
BeginDeferWindowPos(
nNumWindows: Integer       {the number of windows to be moved}
): HDWP;                   {returns a handle to a position structure}
```

Description

This is the first function in a series of functions used to reposition and resize multiple windows simultaneously with a minimum of screen refresh. It allocates memory to an internal structure that tracks the target position and size for the windows to be modified. The DeferWindowPos function fills this structure with information on the new size and position for each window. The EndDeferWindowPos then uses the information to move and resize the windows simultaneously. The screen is not updated until the EndDeferWindowPos function is called.

Parameters

nNumWindows: Specifies the number of windows that will have position information stored in the multiple window position structure. The DeferWindowPos function can increase the size of this structure if necessary, but if there is not enough memory to increase the size, the entire sequence is failed.

Return Value

If this function succeeds, it returns a handle to the multiple window position structure; otherwise, it returns zero.

See Also

DeferWindowPos, EndDeferWindowPos, GetWindowPlacement, MoveWindow, SetWindowPlacement, SetWindowPos, ShowWindow, WM_MOVE, WM_SIZE, WM_WINDOWPOSCHANGED, WM_WINDOWPOSCHANGING

Example

Listing 2-4: Repositioning multiple windows

```
procedure TForm1.FormShow(Sender: TObject);
begin
   {display the other two forms}
   Form2.Show;
   Form3.Show;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
   WindowPosInfo: HDWP;  // holds the internal window position structure
begin
   {allocate memory for moving three windows}
   WindowPosInfo:=BeginDeferWindowPos(3);

   {set up the first window}
   WindowPosInfo:=DeferWindowPos(WindowPosInfo, Form1.Handle, HWND_NOTOPMOST,
                                 50,50,400,100,SWP_SHOWWINDOW);

   {set up the second window}
   WindowPosInfo:=DeferWindowPos(WindowPosInfo, Form2.Handle, HWND_NOTOPMOST,
                                 50,150,400,100,SWP_SHOWWINDOW);

   {set up the third window}
   WindowPosInfo:=DeferWindowPos(WindowPosInfo, Form3.Handle, HWND_NOTOPMOST,
                                 50,250,400,100,SWP_SHOWWINDOW);

   {complete the sequence and reposition the windows}
   EndDeferWindowPos(WindowPosInfo);
end;
```

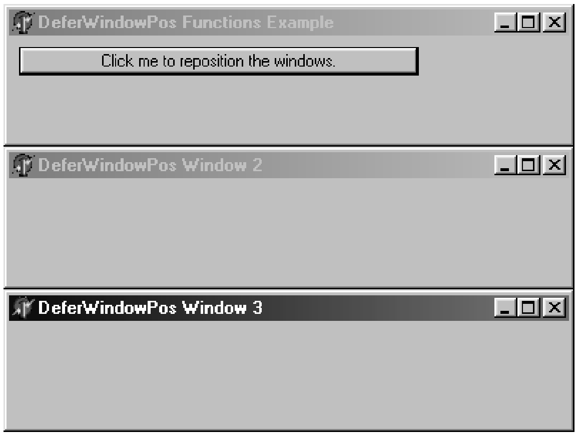

DeferWindowPos Functions Example
Click me to reposition the windows.
DeferWindowPos Window 2
DeferWindowPos Window 3

Figure 2-4: The repositioned windows

BringWindowToTop   Windows.pas

Syntax

```
BringWindowToTop(
hWnd: HWND          {a handle to a window}
): BOOL;            {returns TRUE or FALSE}
```

Description

This function will bring the specified window to the top of its relative z-order, bringing it in front of other windows in the same z-order (i.e., a child window is in front of other child windows, a top-level window is in front of other top-level windows, and a topmost window is in front of other topmost windows). If the window is a top-level or topmost window, it will be activated, but it will not be restored from a minimized state. This function cannot be used to make a window a topmost window. An application should call SetForegroundWindow to make itself the foreground application.

Parameters

hWnd: The handle of the window to bring to the top of the z-order.

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

EnableWindow, IsWindowVisible, SetActiveWindow, SetFocus, SetForegroundWindow, SetWindowPos, WM_ENABLE, WM_SETFOCUS

Example

Listing 2-5: Rearranging the z-order of child windows

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   {bring Panel1 to the top of the child window z-order}
   BringWindowToTop(Panel1.Handle);
end;
```

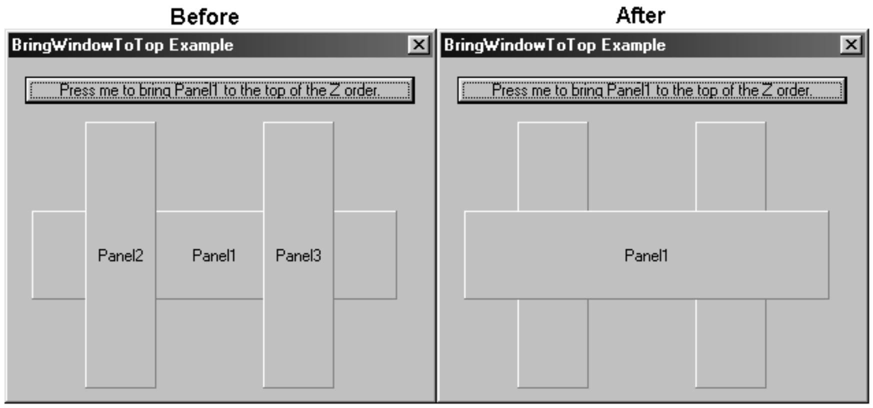

Before
After
BringWindowToTop Example
BringWindowToTop Example
Press me to bring Panel1 to the top of the Z order.
Press me to bring Panel1 to the top of the Z order.
Panel2
Panel1
Panel3
Panel1

Figure 2-5: The rearranged z-order

CascadeWindows Windows.pas

Syntax

```
CascadeWindows(
hwndParent: HWND;   {a handle to the parent window}
wHow: UINT;         {control flags}
lpRect: PRect;      {rectangular area to arrange windows in}
cKids: UINT;        {the number of windows to arrange}
lpKids: Pointer     {the address of an array of window handles}
): WORD;            {returns the number of windows arranged}
```

Description

This function arranges the windows associated by the handles in the lpKids array, or the child windows of the specified window, by cascading them.

Parameters

hwndParent: A handle to the parent window. If this parameter is zero, the desktop window is used as the parent window. If this is being used to cascade MDI child windows, this parameter should be set to the ClientHandle property of the particular form.

wHow: This can be set to MDITILE_SKIPDISABLED to bypass cascading any disabled child windows. If this parameter is set to zero, all child windows are cascaded.

lpRect: A pointer to a TRect structure describing a rectangular area in screen coordinates in which the windows are arranged. If this parameter is NIL, the client area of the parent window is used.

cKids: Specifies the number of elements in the lpKids array. If the lpKids parameter is NIL, this parameter is ignored.

lpKids: A pointer to an array of window handles identifying the windows to be arranged. Specifying NIL for this parameter will arrange all of the child windows of the parent window.

Return Value

If the function succeeds, it returns the number of windows that were arranged; otherwise, it returns zero.

See Also

BeginDeferWindowPos, DeferWindowPos, EndDeferWindowPos, MoveWindow, SetWindowPlacement, TileWindows, WM_MDICASCADE, WM_MDITILE

Example

Listing 2-6: Cascading MDI child windows

```
procedure TForm1.Cascade1Click(Sender: TObject);
begin
   {this will tile all of the MDI child windows except the one that is disabled}
   CascadeWindows(Form1.ClientHandle,MDITILE_SKIPDISABLED,nil,0,nil);
end;
```

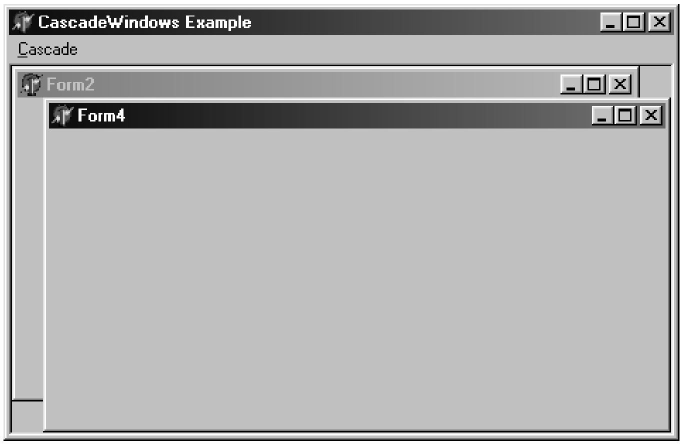


Figure 2-6: Cascaded MDI child windows

CloseWindow   Windows.pas

Syntax

CloseWindow(	
hWnd: HWND	{a handle to a window}
): BOOL;	{returns TRUE or FALSE}

Description

CloseWindow minimizes the specified window but does not destroy it.

Parameters

hWnd: The handle of the window to be minimized.

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

DestroyWindow, IsIconic, IsWindowVisible, IsZoomed, OpenIcon, ShowWindow, WM_SIZE

Example

Listing 2-7: CloseWindow example using OpenIcon and IsIconic

```
{this continually minimizes and restores the form}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  {if our form is minimized...}
  if IsIconic(Form1.Handle) then
     {...restore it...}
     OpenIcon(Form1.Handle)
  else
     {...otherwise minimize it}
     CloseWindow(Form1.Handle);
end;
```

DeferWindowPos Windows.pas

Syntax

```
DeferWindowPos(
hWinPosInfo: HDWP;          {the handle to a position structure}
hWnd: HWND;                 {the handle of a window to position}
hWndInsertAfter: HWND;      {the handle of the preceding window}
X: Integer;                 {the horizontal coordinate}
Y: Integer;                 {the vertical coordinate}
CX: Integer;                {the width, in pixels}
CY: Integer;                {the height, in pixels}
uFlags: UINT                {size and position flags}
): HDWP;                    {returns a handle to a position structure}
```

Description

This function updates the specified multiple window position structure for the new size and position of the indicated window. Use the BeginDeferWindowPos function to allocate memory for this structure. The DeferWindowPos function can increase the size of this structure if necessary, but if there is not enough memory for the increased size, the entire sequence fails. When the EndDeferWindowPos function is called, it uses this structure to move and resize multiple windows simultaneously. The screen is not refreshed until after the EndDeferWindowPos function is completed.

Note that owned windows of a topmost window are also made topmost so that they are displayed above their owner, but owner windows of the specified window are not changed. Thus, a non-topmost window can own a topmost window, but a topmost window cannot own a non-topmost window. If a topmost window is repositioned to a non-topmost window, its owned windows are also changed to non-topmost.

Parameters

hWinPosInfo: The handle to the multiple window position structure that was returned by BeginDeferWindowPos or the last call to DeferWindowPos.

hWnd: The handle of the window to be moved or resized.

hWndInsertAfter: Identifies the window that will precede the repositioned window in the z-order. This is either a window handle or a value from Table 2-2. This parameter is ignored if the SWP_NOZORDER flag is set in the Flags parameter. If this parameter is set to zero, the window will be placed at the top of the z-order. If a window's z-order position is placed above all other topmost windows, that window becomes a topmost window. This has the same effect as specifying the HWND_TOPMOST flag for this parameter.

X: The horizontal coordinate of the window's upper-left corner. If this is a child window, the coordinates are relative to the parent window's client area.

Y: The vertical coordinate of the window's upper-left corner. If this is a child window, the coordinates are relative to the parent window's client area.

CX: The window's new width, in pixels.

CY: The window's new height, in pixels.

uFlags: Specifies a combination of values from Table 2-3 that will affect the size and position of the window.

Return Value

If this function succeeds, it returns a handle to the updated multiple window position structure. This structure could be different from the one passed to the function and should be used in subsequent calls to DeferWindowPos and EndDeferWindowPos. Otherwise, this function returns zero. If the function fails, the application should abandon the window positioning operation and should not call EndDeferWindowPos.

See Also

BeginDeferWindowPos, EndDeferWindowPos, GetWindowPlacement, MoveWindow, SetWindowPlacement, SetWindowPos, ShowWindow, WM_MOVE, WM_SIZE, WM_WINDOWPOSCHANGED, WM_WINDOWPOSCHANGING

Example

Please see Listing 2-4 under BeginDeferWindowPos.

Table 2-2: DeferWindowPos hWndInsertAfter values

Value	Description
HWND_BOTTOM	Places the window at the bottom of the z-order. If this window was a topmost window, it loses its topmost status and is placed below all other windows.
HWND_NOTOPMOST	Places the window above all non-topmost windows but behind all topmost windows. If the window is already a non-topmost window, this flag has no effect.
HWND_TOP	Places the window at the top of the z-order.
HWND_TOPMOST	Places the window above all non-topmost windows. It will retain its topmost position even when deactivated.

Table 2-3: DeferWindowPos uFlags values

Value	Description
SWP_DRAWFRAME	Draws the frame defined in the window's class description around the window.
SWP_FRAMECHANGED	Causes a WM_NCCALCSIZE message to be sent to the window, even if the window size is not changing.
SWP_HIDEWINDOW	Hides the window.
SWP_NOACTIVATE	Does not activate the window. If this flag is not set, the window is activated and moved to the top of the topmost or non-topmost group depending on the hWndInsertAfter parameter.
SWP_NOCOPYBITS	Discards the entire client area. If this flag is not set, the valid area of the client area is saved and copied back into the client area after all movement and positioning is completed.
SWP_NOMOVE	Retains the current position, ignoring the X and Y parameters.
SWP_NOOWNERZORDER	Does not change the owner window's position in the z-order.
SWP_NOREDRAW	When this flag is set, no repainting occurs, and the application must explicitly invalidate or redraw any parts of the window that need to be redrawn, including the non-client area and scroll bars.
SWP_NOREPOSITION	The same as the SWP_NOOWNERZORDER flag.
SWP_NOSENDCHANGING	The window will not receive WM_WINDOWPOSCHANGING messages.
SWP_NOSIZE	Retains the current size, ignoring the CX and CY parameters.
SWP_NOZORDER	Retains the current z-order, effectively causing the WndInsertAfter parameter to be ignored.
SWP_SHOWWINDOW	Displays the window.

EndDeferWindowPos Windows.pas

Syntax

```
EndDeferWindowPos(
hWinPosInfo: HDWP          {the handle of a position structure}
): BOOL;                   {returns TRUE or FALSE}
```

Description

This is the last function called in a series of functions used to simultaneously move and resize multiple windows with a minimum of screen refresh. The BeginDeferWindow-Pos function is called first, which allocates memory for a multiple-window position internal structure that tracks the new position and size of each window to be moved. The DeferWindowPos function is then called for each window to be modified. The EndDeferWindowPos function is called last. This function sends the WM_WINDOWPOSCHANGING and WM_WINDOWPOSCHANGED messages to each window and updates the screen only when all windows have been modified.

Parameters

hWinPosInfo: A handle to the multiple window position internal structure. This handle is returned from the BeginDeferWindowPos and DeferWindowPos functions.

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

BeginDeferWindowPos, DeferWindowPos, GetWindowPlacement, MoveWindow, SetWindowPlacement, SetWindowPos, ShowWindow, WM_MOVE, WM_SIZE, WM_WINDOWPOSCHANGED, WM_WINDOWPOSCHANGING

Example

Please see Listing 2-4 under BeginDeferWindowPos.

GetWindowPlacement Windows.pas

Syntax

```
GetWindowPlacement(
hWnd: HWND;                              {a handle of a window}
WindowPlacement: PWindowPlacement {a pointer to a position data structure}
): BOOL;                                 {returns TRUE or FALSE}
```

Description

This function retrieves the show state and the normal, minimized, and maximized positions of the specified window.

Parameters

hWnd: A handle to the window whose placement information is to be retrieved.

WindowPlacement: A pointer to a TWindowPlacement data structure that will receive the show state and window placement information. This structure is defined as:

```
TWindowPlacement = packed record
    Length: UINT;                {the size of the structure in bytes}
    Flags: UINT;                 {positioning flags}
    ShowCmd: UINT;               {show state flags}
    ptMinPosition: TPoint;       {minimized coordinates}
    ptMaxPosition: TPoint;       {maximized coordinates}
    rcNormalPosition: TRect;     {restored position coordinates}
end;
```

Before calling this function, the Length member must be set to SizeOf(TWindow-Placement). The members of this structure are filled with the placement information after this function is called. Please refer to the SetWindowPlacement function for a description of this data structure.

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

SetWindowPlacement, SetWindowPos, ShowWindow

Example

Please see Listing 2-9 under SetWindowPlacement.

MoveWindow ***Windows.pas***

Syntax

```
MoveWindow(
hWnd: HWND;            {a handle to a window to be moved}
X: Integer;            {the new horizontal coordinate}
Y: Integer;            {the new vertical coordinate}
nWidth: Integer;       {the new window width}
nHeight: Integer;      {the new window height}
bRepaint: BOOL         {the repaint flag}
): BOOL;               {returns TRUE or FALSE}
```

Description

This function changes the position and dimensions of the specified window. If the specified window is a top-level window, the coordinates are relative to the screen. If the specified window is a child window, coordinates are relative to the parent window's client area.

This function sends the following messages to the specified window: WM_WINDOW-POSCHANGING, WM_WINDOWPOSCHANGED, WM_MOVE, WM_SIZE, and WM_NCCALCSIZE.

Parameters

hWnd: A handle to the window to be modified.

X: The new horizontal coordinate for the upper-left corner of the window.

Y: The new vertical coordinate for the upper-left corner of the window.

nWidth: Specifies the new width of the window.

nHeight: Specifies the new height of the window.

bRepaint: Determines how this window will be repainted. If this parameter is TRUE, the MoveWindow function calls the UpdateWindow function. This sends a WM_PAINT message to the window, causing it to be repainted immediately after the window is moved. If this parameter is FALSE, no repainting will occur, including the entire non-client area and any part of the parent window uncovered by a child window. The application must explicitly invalidate or redraw any areas that need to be updated as a result of the MoveWindow function. A WM_PAINT message is placed in the message queue of the specified window, but its message loop will only dispatch the WM_PAINT message after all other messages have been dispatched.

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE.

See Also

BeginDeferWindowPos, DeferWindowPos, EndDeferWindowPos, SetWindowPlacement, SetWindowPos

Example

Listing 2-8: Moving a window

```
var
   XPos: Integer = 5;  // our initial horizontal position

procedure TForm1.Timer1Timer(Sender: TObject);
begin
   {increment the horizontal position}
   Inc(XPos);

   {move the edit box to the right}
   MoveWindow(Edit1.Handle, XPos, Edit1.Top, Edit1.Width, Edit1.Height, TRUE);
end;
```

OpenIcon Windows.pas

Syntax

```
OpenIcon(
hWnd: HWND          {a handle to a minimized window}
): BOOL;            {returns TRUE or FALSE}
```

Description

This function restores and activates the specified minimized window. A WM_QUERYOPEN message is sent to the window when this function is called.

Parameters

hWnd: A handle to the window to be restored and activated.

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CloseWindow, DestroyWindow*, IsIconic, IsWindowVisible, IsZoomed, ShowWindow, WM_SIZE

Example

Please see Listing 2-7 under CloseWindow.

SetWindowPlacement Windows.pas

Syntax

```
SetWindowPlacement(
hWnd: HWND;                              {a handle to a window}
WindowPlacement: PWindowPlacement        {a pointer to a window placement structure}
): BOOL;                                 {returns TRUE or FALSE}
```

Description

This function sets the show state and the normal, minimized, and maximized coordinates of the specified window.

Parameters

hWnd: A handle to the window whose placement information is to be set.

WindowPlacement: A pointer to a TWindowPlacement data structure that contains the show state and window placement information. This structure is defined as:

```
TWindowPlacement = packed record
      length: UINT;                 {the size of the structure in bytes}
      flags: UINT;                  {positioning flags}
      showCmd: UINT;                {show state flags}
      ptMinPosition: TPoint;        {minimized coordinates}
      ptMaxPosition: TPoint;        {maximized coordinates}
      rcNormalPosition: TRect;      {restored position coordinates}
end;
```

length: The size of the structure, in bytes. Before calling this function, this member must be set to SizeOf(TWindowPlacement).

flags: Specifies flags that control the position of a minimized window and the method by which the window is restored. This member can be one or more of the flags in Table 2-4.

showCmd: Specifies the current show state of the window and can be one of the values in Table 2-5.

ptMinPosition: The coordinates of the upper-left corner of the window when it is minimized, stored in the members of a TPoint structure.

ptMaxPosition: The coordinates of the upper-left corner of the window when it is maximized, stored in the members of a TPoint structure.

rcNormalPosition: The coordinates of the window in a normal, restored position, stored in the members of a TRect structure.

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

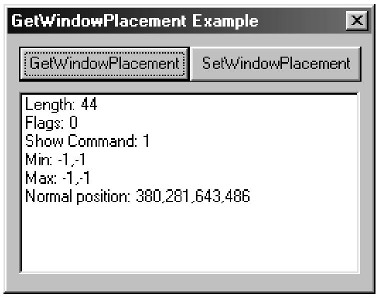
See Also

GetWindowPlacement, SetWindowPos, ShowWindow

Example

Listing 2-9: Window placement information

```
const
  WPF_ASYNCWINDOWPLACEMENT = $0004;   // This constant is not defined in Delphi

{get the window placement}
procedure TForm1.Button1Click(Sender: TObject);
var
   PlacementInfo: TWindowPlacement;
begin
   {we must set the length to the size of the data structure first}
   PlacementInfo.Length:=SizeOf(TWindowPlacement);

   {get the window placement information}
   GetWindowPlacement(Form1.Handle, @PlacementInfo);

   {empty the list box}
   ListBox1.Items.Clear;

   {display all of the information in the window placement structure}
   ListBox1.Items.Add('Length: '+IntToStr(PlacementInfo.length));
   ListBox1.Items.Add('Flags: '+IntToStr(PlacementInfo.Flags));
   ListBox1.Items.Add('Show Command: '+IntToStr(PlacementInfo.showCmd));
   ListBox1.Items.Add('Min: '+IntToStr(PlacementInfo.ptMinPosition.X)+','+
                      IntToStr(PlacementInfo.ptMinPosition.Y));
   ListBox1.Items.Add('Max: '+IntToStr(PlacementInfo.ptMaxPosition.X)+','+
                      IntToStr(PlacementInfo.ptMaxPosition.Y));
   ListBox1.Items.Add('Normal position: '+
                      IntToStr(PlacementInfo.rcNormalPosition.Left)+','+
                      IntToStr(PlacementInfo.rcNormalPosition.Top)+','+
                      IntToStr(PlacementInfo.rcNormalPosition.Right)+','+
                      IntToStr(PlacementInfo.rcNormalPosition.Bottom));
end;

{set the window placement}
procedure TForm1.Button2Click(Sender: TObject);
var
   PlacementInfo: TWindowPlacement;
begin
   {we must set the length to the size of the data structure first}
   PlacementInfo.Length:=SizeOf(TWindowPlacement);

   {fill in the rest of the window structure members}
   PlacementInfo.flags:=WPF_SETMINPOSITION;
   PlacementInfo.showCmd:=SW_SHOW;
   PlacementInfo.ptMinPosition.X:=100;
   PlacementInfo.ptMinPosition.Y:=100;
   PlacementInfo.ptMaxPosition.X:=50;
   PlacementInfo.ptMaxPosition.Y:=50;
   PlacementInfo.rcNormalPosition.Left:=100;
   PlacementInfo.rcNormalPosition.Top:=100;
   PlacementInfo.rcNormalPosition.Right:=250;
   PlacementInfo.rcNormalPosition.Bottom:=250;

   {set the window placement information}
   SetWindowPlacement(Form1.Handle, @PlacementInfo);
end;
```



```
Length: 44
Flags: 0
Show Command: 1
Min: -1,-1
Max: -1,-1
Normal position: 380,281,643,486
```

Figure 2-7: Getting the window placement information

Table 2-4: SetWindowPlacement WindowPlacement.flags values

Value	Description
WPF_ASYNCWINDOWPLACEMENT	**Windows 2000 or later**: The system posts the request to the message queue of the thread owning the affected window and returns immediately.
WPF_RESTORETOMAXIMIZED	The window will be maximized the next time it is restored, regardless of whether or not it was maximized before being minimized. This is valid only the next time that the window is restored and when the SW_SHOWMINIMIZED flag is set for the showCmd member. This does not change the default restoration behavior.
WPF_SETMINPOSITION	The coordinates of the minimized window may be specified. This flag must be included if coordinates are set in the ptMinPosition member.

Table 2-5: SetWindowPlacement WindowPlacement.showCmd values

Value	Description
SW_HIDE	The window is hidden and another window is activated.
SW_MINIMIZE	The window is minimized and the next top-level window in the system window list is activated.
SW_RESTORE	The window is activated and displayed in its original size and position.
SW_SHOW	The window is activated and displayed in its current size and position.
SW_SHOWDEFAULT	The window is shown based on the wShowWindow member of the TStartupInfo structure passed to the CreateProcess function by the program that started the application. This is used to set the initial show state of an application's main window. This flag should be used when showing the window for the first time if the application can be run from a shortcut. This flag will cause the window to be shown using the Run settings under the shortcut properties.
SW_SHOWMAXIMIZED	The window is activated and displayed in a maximized state.
SW_SHOWMINIMIZED	The window is activated and displayed as an icon.
SW_SHOWMINNOACTIVE	The window is displayed as an icon. The active window remains active.
SW_SHOWNA	The window is displayed in its current state. The active window remains active.
SW_SHOWNOACTIVE	The window is displayed in its most recent state. The active window remains active.
SW_SHOWNORMAL	This is the same as SW_RESTORE.

SetWindowPos Windows.pas

Syntax

```
SetWindowPos(
hWnd: HWND;                {a handle to a window}
hWndInsertAfter: HWND;     {a window handle or positioning flag}
X: Integer;                {the horizontal position}
Y: Integer;                {the vertical position}
CX: Integer;               {the width of the window}
CY: Integer;               {the height of the window}
uFlags: UINT               {size and positioning flags}
): BOOL;                   {returns TRUE or FALSE}
```

Description

This function changes the size, position, and z-order of the specified window. The z-order of child, pop-up, and top-level windows is determined by the order in which these windows appeared on the screen. The topmost window is the first window in the z-order.

Note that owned windows of a topmost window are also made topmost so that they are displayed above their owner, but owner windows of the specified window are not changed. Thus, a non-topmost window can own a topmost window, but a topmost window cannot own a non-topmost window. If a topmost window is repositioned to a non-topmost window, its owned windows are also changed to non-topmost.

Parameters

hWnd: The handle of the window to be moved or resized.

hWndInsertAfter: Identifies the window that will precede the repositioned window in the z-order. This is either a window handle or a value from Table 2-6. This parameter is ignored if the SWP_NOZORDER flag is set in the uFlags parameter. If this parameter is set to zero, the window will be placed at the top of the z-order. If a window's z-order position is placed above all other topmost windows, that window becomes a topmost window. This has the same effect as specifying the HWND_TOPMOST flag for this parameter.

X: The horizontal coordinate of the window's upper-left corner. If this is a child window, the coordinates are relative to the parent window's client area.

Y: The vertical coordinate of the window's upper-left corner. If this is a child window, the coordinates are relative to the parent window's client area.

CX: The window's new width, in pixels.

CY: The window's new height, in pixels.

uFlags: Specifies a combination of values from Table 2-7 that will affect the size and position of the window.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

BeginDeferWindowPos, DeferWindowPos, EndDeferWindowPos, MoveWindow, SetActiveWindow, SetForegroundWindow, SetWindowPlacement, ShowWindow, WM_MOVE, WM_SIZE

Example

Listing 2-10: Setting the window position

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   {resize the memo so that it takes up the entire client area below the button}
   SetWindowPos(Memo1.Handle,0,0,Button1.Top+Button1.Height+5,Form1.ClientWidth,
                Form1.ClientHeight-(Button1.Top+Button1.Height+5),
                SWP_SHOWWINDOW);
end;
```

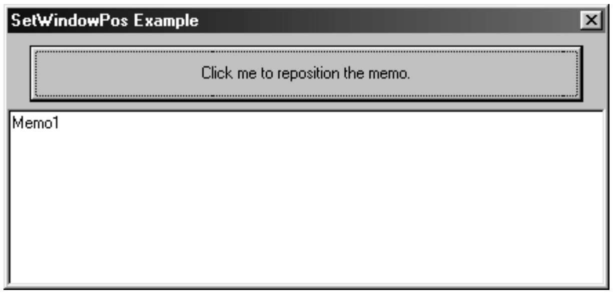


Figure 2-8: The repositioned memo

Table 2-6: SetWindowPos hWndInsertAfter values

Value	Description
HWND_BOTTOM	Places the window at the bottom of the z-order. If this window was a topmost window, it loses its topmost status and is placed below all other windows.
HWND_NOTOPMOST	Places the window above all non-topmost windows but behind all topmost windows. If the window is already a non-topmost window, this flag has no effect.
HWND_TOP	Places the window at the top of the z-order.
HWND_TOPMOST	Places the window above all non-topmost windows. It will retain its topmost position even when deactivated.

Table 2-7: SetWindowPos uFlags values

Value	Description
SWP_DRAWFRAME	Draws the frame defined in the window's class description around the window.
SWP_FRAMECHANGED	Causes a WM_NCCALCSIZE message to be sent to the window, even if the window size is not changing.
SWP_HIDEWINDOW	Hides the window.
SWP_NOACTIVATE	Does not activate the window. If this flag is not set, the window is activated and moved to the top of the topmost or non-topmost group, depending on the hWndInsertAfter parameter.
SWP_NOCOPYBITS	Discards the entire client area. If this flag is not set, the valid area of the client area is saved and copied back into the client area after all movement and positioning is completed.
SWP_NOMOVE	Retains the current position, ignoring the X and Y parameters.
SWP_NOOWNERZORDER	Does not change the owner window's position in the z-order.
SWP_NOREDRAW	When this flag is set, no repainting occurs, and the application must explicitly invalidate or redraw any parts of the window that need to be redrawn, including the non-client area and scroll bars.
SWP_NOREPOSITION	The same as the SWP_NOOWNERZORDER flag.
SWP_NOSENDCHANGING	The window will not receive WM_WINDOWPOSCHANGING messages.
SWP_NOSIZE	Retains the current size, ignoring the CX and CY parameters.

Value	Description
SWP_NOZORDER	Retains the current z-order, effectively causing the WndInsertAfter parameter to be ignored.
SWP_SHOWWINDOW	Displays the window.

ShowOwnedPopups Windows.pas

Syntax

```
ShowOwnedPopups(
hWnd: HWND;          {a handle to a window}
fShow: BOOL          {the window visibility flag}
): BOOL;             {returns TRUE or FALSE}
```

Description

This function will show or hide all pop-up windows owned by the specified window. Pop-up windows will only be shown if hidden by a previous call to ShowOwnedPopups (a window hidden with the ShowWindow function will not be displayed when ShowOwnedPopups is called).

Parameters

hWnd: A handle to the window owning the pop-ups to be shown.

fShow: Determines if pop-up windows are shown or hidden. A value of TRUE displays all hidden pop-up windows owned by the specified window. A value of FALSE will hide all visible pop-up windows.

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

IsWindowVisible, SetWindowPos, ShowWindow

Example

Listing 2-11: Toggling the show state of owned pop-up windows

This code belongs in the unit for the main form:

```
var
  Form1: TForm1;
  ShowIt: Boolean;   // our toggle variable

implementation

uses Unit2;

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
   {toggle our show state variable}
   ShowIt:=not ShowIt;

   {show or hide all pop-ups owned by the main form}
   ShowOwnedPopups(Form1.Handle, ShowIt);
end;

procedure TForm1.FormShow(Sender: TObject);
begin
   {show the second form when the program starts}
   Form2.Show;
end;

initialization
   {initialize our toggle variable}
   ShowIt:=TRUE;
```

This code goes in the unit for Form2:

```
uses Unit2;

{we must override CreateParams to set this window's owner}
procedure TForm2.CreateParams(var Params: TCreateParams);
begin
   {fill in the default creation parameters}
   inherited CreateParams(Params);

   {set this form's owner to the main form}
   Params.WndParent:=Form1.Handle;
end;
```

ShowWindow   Windows.pas

Syntax

```
ShowWindow(
hWnd: HWND;            {a handle to a window}
nCmdShow: Integer      {the show state of the window}
): BOOL;               {returns TRUE or FALSE}
```

Description

This function sets the specified window's display state. When displaying an application's main window, the developer should specify the SW_SHOWDEFAULT flag. This will display the window as instructed by application startup information. For example, if a Windows 95 shortcut has its properties set to run the application minimized, the SW_SHOWDEFAULT flag will show the window minimized. Without this flag, these shortcut properties are ignored.

Parameters

hWnd: A handle to the window to be shown.

nCmdShow: Specifies how the window will be shown and can be one of the values from the following table.

Return Value

If the function succeeds and the window was previously visible, it returns TRUE. If the function fails or the window was previously hidden, it returns FALSE.

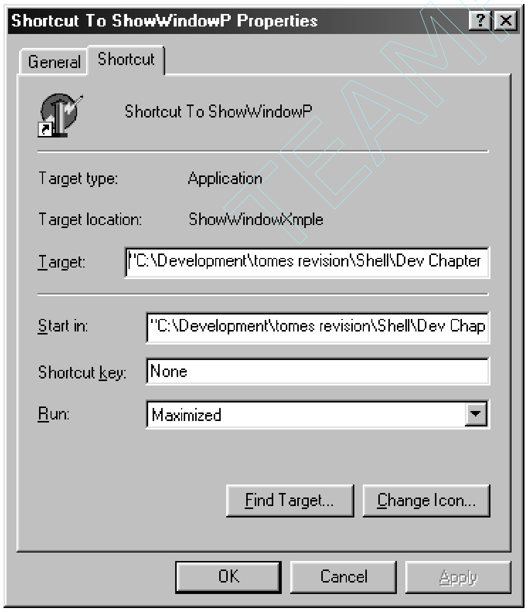
See Also

CreateProcess*, SetWindowPlacement, SetWindowPos, ShowOwnedPopups, ShowWindowAsync, WM_SHOWWINDOW

Example

Listing 2-12: Showing a window based on shortcut properties

```
const
  {Delphi 6 does not define all available constants}
  SW_FORCEMINIMIZE = 11;

procedure TForm1.FormCreate(Sender: TObject);
begin
   {this example is run from a shortcut, and this line will
    show the window based on the shortcut properties}
   ShowWindow(Form1.Handle, SW_SHOWDEFAULT);
end;
```



Shortcut To ShowWindowP

Target type: Application

Target location: ShowWindowXmple

Target: "C:\Development\tomes revision\Shell\Dev Chapter

Start in: "C:\Development\tomes revision\Shell\Dev Chap

Shortcut key: None

Run: Maximized

Figure 2-9: The shortcut settings start this application maximized

Table 2-8: ShowWindow nCmdShow values

Value	Description
SW_FORCEMINIMIZE	**Windows 2000/XP only**: Minimizes a window even if the owning thread is hung.
SW_HIDE	The window is hidden and another window is activated.

Value	Description
SW_MINIMIZE	The window is minimized and the next top-level window in the system window list is activated.
SW_RESTORE	The window is activated and displayed in its original size and position.
SW_SHOW	The window is activated and displayed in its current size and position.
SW_SHOWDEFAULT	The window is shown based on the wShowWindow member of the TStartupInfo structure passed to the CreateProcess function by the program that started the application. This is used to set the initial show state of an application's main window. This flag should be used when showing the window for the first time if the application can be run from a shortcut. This flag will cause the window to be shown using the Run settings under the shortcut properties.
SW_SHOWMAXIMIZED	The window is activated and displayed in a maximized state.
SW_SHOWMINIMIZED	The window is activated and displayed as an icon.
SW_SHOWMINNOACTIVE	The window is displayed as an icon. The active window remains active.
SW_SHOWNA	The window is displayed in its current state. The active window remains active.
SW_SHOWNOACTIVE	The window is displayed in its most recent state. The active window remains active.
SW_SHOWNORMAL	This is the same as SW_RESTORE.

ShowWindowAsync Windows.pas

Syntax

```
ShowWindowAsync(
hWnd: HWND;                {a handle to a window}
nCmdShow: Integer          {the show state of the window}
): BOOL;                   {returns TRUE or FALSE}
```

Description

This function is similar to ShowWindow. Its purpose is to set the display state of a window created by a different thread. This function posts a WM_SHOWWINDOW message to the message queue of the specified window. This allows the calling application to continue execution if the application associated with the specified window is hung.

Parameters

hWnd: A handle to the window to be shown.

nCmdShow: Specifies how the window will be shown and can be one of the values from the following table.

Return Value

If the function succeeds and the window was previously visible, it returns TRUE. If the function fails or the window was previously hidden, it returns FALSE.

See Also

CreateProcess*, SetWindowPlacement, SetWindowPos, ShowOwnedPopups, ShowWindow, WM_SHOWWINDOW

Example

Listing 2-13: Showing a window asynchronously

```
procedure TForm1.Button1Click(Sender: TObject);
var
   TheWindow: HWND;
begin
   {find a handle to the Windows Explorer window. Windows Explorer must be running}
   TheWindow:=FindWindow('ExploreWClass',nil);

   {show it}
   ShowWindowAsync(TheWindow, SW_MAXIMIZE);
end;
```

Table 2-9: ShowWindowAsync nCmdShow values

Value	Description
SW_HIDE	The window is hidden and another window is activated.
SW_MINIMIZE	The window is minimized, and the next top-level window in the system window list is activated.
SW_RESTORE	The window is activated and displayed in its original size and position.
SW_SHOW	The window is activated and displayed in its current size and position.
SW_SHOWDEFAULT	The window is shown based on the wShowWindow member of the TStartupInfo structure passed to the CreateProcess function by the program that started the application. This is used to set the initial show state of an application's main window. This flag should be used when showing the window for the first time if the application can be run from a shortcut. This flag will cause the window to be shown using the Run settings under the shortcut properties.
SW_SHOWMAXIMIZED	The window is activated and displayed in a maximized state.
SW_SHOWMINIMIZED	The window is activated and displayed as an icon.
SW_SHOWMINNOACTIVE	The window is displayed as an icon. The active window remains active.
SW_SHOWNA	The window is displayed in its current state. The active window remains active.
SW_SHOWNOACTIVE	The window is displayed in its most recent state. The active window remains active.
SW_SHOWNORMAL	This is the same as SW_RESTORE.

TileWindows *Windows.pas*

Syntax

```
TileWindows(
hwndParent: HWND;           {a handle to a parent window}
wHow: UINT;                 {tiling flags}
lpRect: PRect;              {the area to arrange the windows in}
cKids: UINT;                {the number of windows to tile}
lpKids: Pointer             {the address to an array of window handles}
): WORD;                    {returns the number of windows arranged}
```

Description

This function arranges the windows associated by the handles in the lpKids array, or the child windows of the specified window, by tiling them. The windows can be tiled in a horizontal or vertical fashion and can be restricted to a rectangular area within the specified parent window.

Parameters

hwndParent: A handle to the parent window. If this parameter is zero, the desktop window is assumed to be the parent window.

wHow: Specifies how the windows are tiled. MDITILE_HORIZONTAL tiles windows horizontally, and MDITILE_VERTICAL tiles windows vertically. The MDITILE_SKIPDISABLED flag can be combined with either of the previous flags to exclude any windows that are disabled from the tiling process.

lpRect: A pointer to a TRect structure containing the coordinates of the area in which the windows are arranged. If this parameter is NIL, the entire client area of the parent window is used.

cKids: Specifies the number of elements in the array pointed to by the lpKids parameter. If the lpKids parameter is NIL, this parameter is ignored.

lpKids: A pointer to an array of window handles identifying the windows to be tiled. If this parameter is NIL, all of the child windows of the specified parent window are tiled.

Return Value

If the function succeeds, it returns the number of windows tiled; otherwise, it returns zero.

See Also

BeginDeferWindowPos, CascadeWindows, DeferWindowPos, EndDeferWindowPos, MoveWindow, SetWindowPlacement, WM_MDICASCADE, WM_MDITILE

Example

Listing 2-14: Vertically tiling MDI child windows

```
procedure TForm1.TileWindows1Click(Sender: TObject);
begin
   {this will tile all of the MDI child windows vertically}
   TileWindows(Form1.ClientHandle,MDITILE_VERTICAL,nil,0,nil);
end;
```

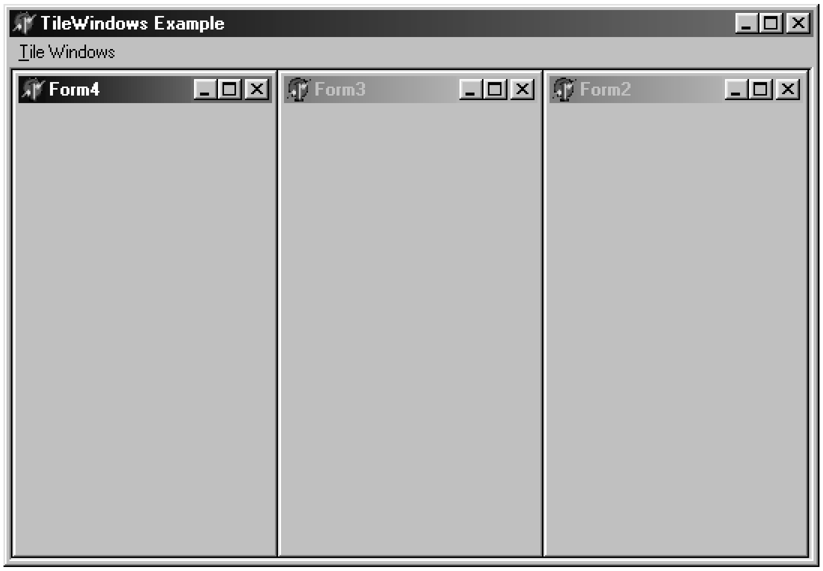


Figure 2-10: Tiled windows

Table 2-10: TileWindow wHow values

Value	Description
MDITILE_HORIZONTAL	The windows are tiled horizontally.
MDITILE_SKIPDISABLED	Any disabled windows are not tiled.
MDITILE_VERTICAL	The windows are tiled vertically.

Chapter 3

Window Information Functions

A window, by its very nature, has a lot of specific information associated with it. Details such as a window's dimensions, position, parent, or even style flags may need to be retrieved by an application, or even modified. The class that the window itself is based on may contain information that an application needs to retrieve on the fly. Fortunately, the Win32 API has a collection of functions that allow the application to retrieve, and sometimes modify, almost any detail concerning a window or its class.

Information Storage

Every window has an information storage mechanism known as a property list. This property list is intended solely for user-required data and is not used directly by the Windows operating system. Every window has one, including forms and any controls descending from TWinControl. The property list is stored in a memory area associated with each specific window that Windows manages automatically.

A property list works in a manner similar to INI files, in that a string is associated with a specific value. The SetProp function takes a string and a 32-bit integer number. If the string does not already exist in the property list, the specified string and data are added. If the string does exist, then the data for that string is changed to the specified number. The GetProp function provides the method to extract these properties, and the RemoveProp function deletes them from the property list. An application should not remove properties that another application assigned to the list, but it should remove its own properties before halting execution.

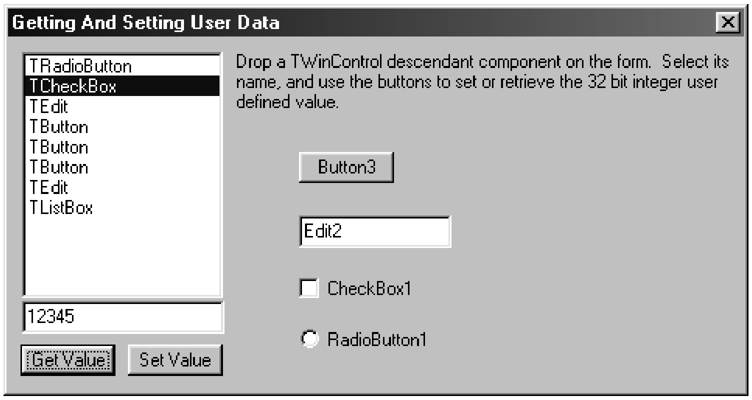
Property lists give the developer a good alternative to global variables, which can sometimes cause problems such as scoping issues or name collisions. It allows the developer to store any amount of information for any purpose, while giving Windows the job of managing the memory required for the storage. It is also a very flexible way to communicate information; a calling function does not need to know the number of properties in the list or even a specific offset, only the string associated with the desired data. See the EnumProps function description for an example of using window property lists.

Window-specific Information

In addition to the property list, each window automatically has a storage area for a single 32-bit number. This is also intended for user-defined data requirements and is not used by the Windows operating system. A developer could use this to store a 32-bit pointer to a data structure, an object, etc. The 32-bit user data area is accessed through the GetWindowLong and SetWindowLong functions. The following example demonstrates setting and retrieving this value for Delphi components descended from TWinControl.

Listing 3-1: Setting and retrieving the 32-bit user data value

```
{the enumeration callback function}
procedure EnumerateChildWindows(hWnd: HWND; lParam: LPARAM); stdcall;

var
  Form1: TForm1;

implementation

{this is called for every existing child window on the form}
procedure EnumerateChildWindows(hWnd: HWND; lParam: LPARAM);
var
  TheClassName: Array[0..255] of char;
begin
  {retrieve the window text of the child window...}
  GetClassName(hWnd, TheClassName, 255);

  {...and display it}
  Form1.ListBox1.Items.AddObject(TheClassName,TObject(hWnd));
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
  {display the class names of all child windows upon activation}
  EnumChildWindows(Form1.Handle,@EnumerateChildWindows,0);
end;

procedure TForm1.ListBox1Click(Sender: TObject);
begin
  {empty the edit box when another control is clicked on}
  Edit1.Clear;
end;

{notice in these two procedures that the window handle for the control
 is stored in the Objects array. since the Objects array holds pointers,
 and a pointer is just a 32-bit number, we can cast the window handle
 as a TObject, and cast the TObject back into a 32-bit integer (which is
 what an HWND is defined as) to store window handles with their
 associated window text.}

procedure TForm1.Button1Click(Sender: TObject);
begin
  {retrieve the 32-bit user-defined value}
  Edit1.Text := IntToStr(GetWindowLong(Longint(ListBox1.Items.Objects[ListBox1.
                                     ItemIndex]), GWL_USERDATA));
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  {set the 32-bit user-defined value for the selected window}
  SetWindowLong(Longint(ListBox1.Items.Objects[ListBox1.ItemIndex]),
                GWL_USERDATA, StrToInt(Edit1.Text));

  {empty the edit box to indicate the function completed}
  Edit1.Clear;
end;
```


Getting And Setting User Data

TRadioButton
TCheckBox
TEdit
TButton
TButton
TButton
TEdit
TListBox

Drop a TWinControl descendant component on the form. Select its name, and use the buttons to set or retrieve the 32 bit integer user defined value.

Figure 3-1: The 32-bit user data value

Subclassing a Window

The GetWindowLong and SetWindowLong functions (and their sister functions for classes) provide the developer with a method to change other window properties programmatically. One of the most powerful tricks a developer can perform with these functions is to replace the window procedure for a window, creating what is known as a subclass. All windows of a specific class share the window procedure defined for that class when the window was registered. The window procedure for a class can be replaced with a new window procedure by using the SetClassLong function, affecting all windows created using that class; to replace the window procedure for one specific window, use the SetWindowLong function. Messages for the subclassed window go to the new window procedure first, allowing the developer to drastically change the behavior of a window on the fly. The following example demonstrates how to use the SetWindowLong function to replace the window procedure of the main form at run time. The new window procedure intercepts the WM_NCHITTEST message. If the user is trying to move the form by clicking on the caption bar and dragging, the new window procedure replaces the result of the message with a result that indicates the user clicked on the client area. This prevents the user from moving the window with the mouse.

Listing 3-2: Replacing the window procedure at run time

```
{the prototype for the subclassed window procedure}
  function SubclassedWndProc(hWnd: HWND; Msg: UINT; wParam: WPARAM;
                              lParam: LPARAM): LResult; stdcall;
var
  Form1: TForm1;
  OldWndProc: Pointer;  // a pointer to the old window procedure

implementation

function SubclassedWndProc(hWnd: HWND; Msg: UINT; wParam: WPARAM;
                              lParam: LPARAM): LResult;
begin
  {pass all messages to the previous window procedure. note that it
   is very important to pass all unhandled messages back to the
   original procedure, or the application and the entire system may crash.}
  Result := CallWindowProc(OldWndProc, Form1.Handle, Msg, wParam, lParam);

  {if the user is clicking on the caption bar, change the
   result to indicate that the user clicked on the client area}
  if ((Msg=WM_NCHITTEST) and (Result=HTCAPTION)) then
    Result := HTCLIENT;

end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  {subclass the form upon creation}
  OldWndProc := Pointer(SetWindowLong(Form1.Handle, GWL_WNDPROC,
                        longint(@SubclassedWndProc)));
end;
```

Knowing It All

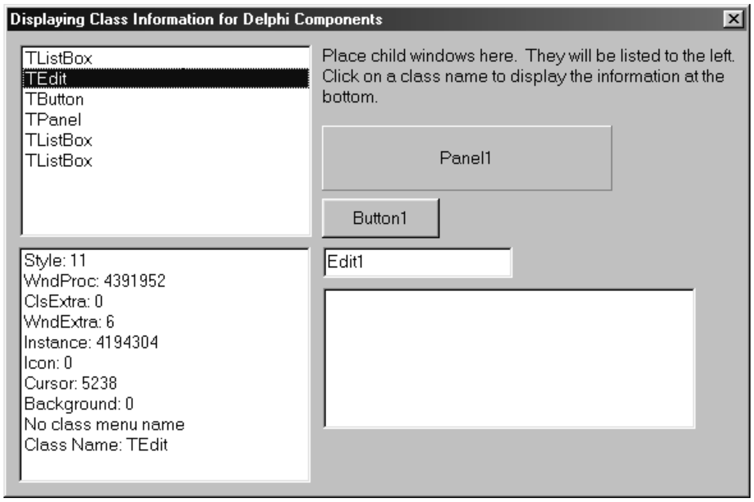
The Win32 API includes a number of enumeration functions. These functions allow the developer to retrieve information on every window without knowing how many windows currently exist. These functions, coupled with the other window information functions, give the application the ability to change dynamically as the system environment changes.

The ClassInfo example on the companion CD demonstrates the use of the EnumChildWindows and GetClassInfo functions to display run-time class information on standard Delphi components. This example iterates through every TWinControl component on the form to retrieve its class name. When a specific class name is selected from a list box, its class information is displayed.

Listing 3-3: Displaying class information for Delphi components

```
{the enumeration callback function}
procedure EnumerateChildWindows(hWnd: HWND; lParam: LPARAM); stdcall;
```

```
var
  Form1: TForm1;

implementation

{this is called once for each child window}
procedure EnumerateChildWindows(hWnd: HWND; lParam: LPARAM);
var
   TheClassName: Array[0..255] of char;   // holds the child window class name
begin
   {retrieve the name of the child window class...}
   GetClassName(hWnd, TheClassName, 255);

   {...and display it}
   Form1.ListBox1.Items.Add(TheClassName);
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
   {retrieve the class names when the form becomes active}
   EnumChildWindows(Form1.Handle,@EnumerateChildWindows,0);
end;

procedure TForm1.ListBox1Click(Sender: TObject);
var
   ClassInfo: TWndClass;               // a class information structure
   ClassName: array[0..255] of char;   // holds the class name
begin
   {get the class information for the selected class}
   StrPCopy(ClassName,ListBox1.Items[ListBox1.ItemIndex]);
   GetClassInfo(hInstance,ClassName,ClassInfo);

   {display the information in the TWndClass structure
    retrieved from the specified class}
   ListBox2.Items.Clear;

   ListBox2.Items.Add(Format('Style: %d',[ClassInfo.style]));
   ListBox2.Items.Add(Format('WndProc: %d',[integer(ClassInfo.lpfnWndProc)]));
   ListBox2.Items.Add(Format('ClsExtra: %d',[ClassInfo.cbClsExtra]));
   ListBox2.Items.Add(Format('WndExtra: %d',[ClassInfo.cbWndExtra]));
   ListBox2.Items.Add(Format('Instance: %d',[integer(ClassInfo.hInstance)]));
   ListBox2.Items.Add(Format('Icon: %d',[integer(ClassInfo.hIcon)]));
   ListBox2.Items.Add(Format('Cursor: %d',[integer(ClassInfo.hCursor)]));
   ListBox2.Items.Add(Format('Background: %d',[integer(ClassInfo.hbrBackground)]));
   if (ClassInfo.lpszMenuName<>nil) then
      ListBox2.Items.Add('Menu Name: '+ClassInfo.lpszMenuName)
   else
      ListBox2.Items.Add('No class menu name');
   if (ClassInfo.lpszClassName<>nil) then
      ListBox2.Items.Add('Class Name: '+ClassInfo.lpszClassName);

end;
```


Displaying Class Information for Delphi Components
TListBox
TEdit
TButton
TPanel
TListBox
TListBox
Place child windows here. They will be listed to the left. Click on a class name to display the information at the bottom.
Panel1
Button1
Edit1

```
Style: 11
WndProc: 4391952
ClsExtra: 0
WndExtra: 6
Instance: 4194304
Icon: 0
Cursor: 5238
Background: 0
No class menu name
Class Name: TEdit
```

Figure 3-2: Delphi object class information

These functions can be combined with other enumeration functions to provide almost any detail about any window in the system. The following example is from the WindowInfo application included on the companion CD. It demonstrates multiple enumeration and window information functions. This application can be used as a complement to the WinSight32 application that ships with Delphi to provide a complete source of information for every window in the system.

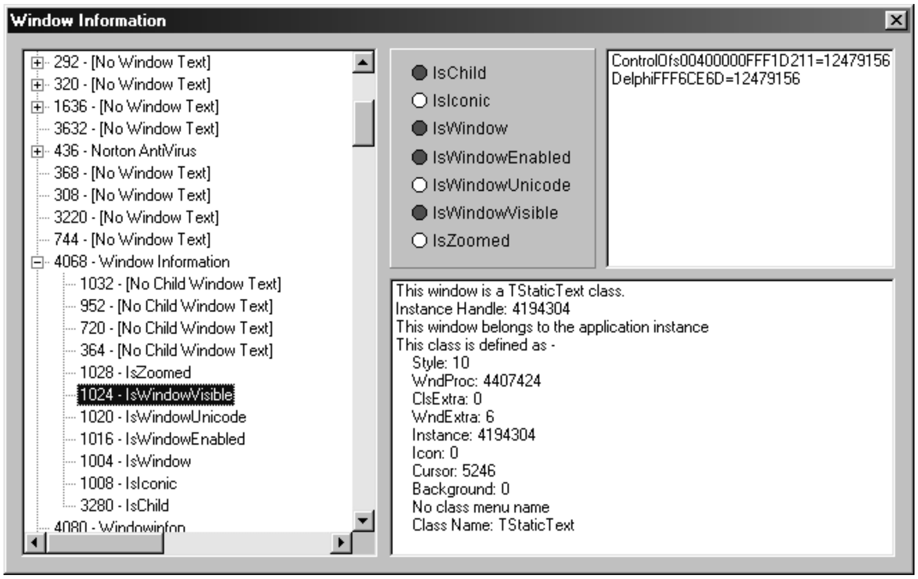
Listing 3-4: Displaying window and class information

```
{prototypes for enumeration functions. these must all have the stdcall
 keyword at the end.}
function EnumerateWindows(hWnd: HWND; lParam: LPARAM): BOOL; stdcall;
function EnumerateChildWindows(hWnd: HWND;lParam: LPARAM):BOOL; stdcall;
function EnumProperties(hWnd: HWND; lpszString: PChar; hData: THandle): BOOL; stdcall;

var
  Form1: TForm1;

implementation

function EnumerateChildWindows(hWnd: HWND; lParam: LPARAM): BOOL;
var
   WindowText: array[0..255] of char;
begin
   {get the text displayed in the child window}
   GetWindowText(hWnd, WindowText, 255);

   {indicate if the child window does not have any text}
   if (WindowText='') then WindowText:='[No Child Window Text]';

   {add an item to the treeview object for a child window}
   Form1.Treeview1.Items.AddChild(Form1.Treeview1.Items[lParam],IntToStr(hWnd)+
                               ' - '+WindowText);

   {continue enumeration}
   Result:=TRUE;
end;

function EnumerateWindows(hWnd: HWND; lParam: LPARAM): BOOL;
var
   WindowText: array[0..255] of char;
begin
   {get the text displayed in the window}
   GetWindowText(hWnd, WindowText, 255);

   {indicate if the window does not have any text}
   if (WindowText='') then WindowText:='[No Window Text]';

   {add an item to the treeview object for a top level window}
   Form1.TreeView1.Items.Add(nil,IntToStr(hWnd)+' - '+WindowText);

   {now, enumerate all of the child windows of this top level window}
   EnumChildWindows(hWnd, @EnumerateChildWindows, Form1.TreeView1.Items.Count-1);

   {continue enumeration of top level windows}
   Result:=TRUE;
end;

function EnumProperties(hWnd: HWND; lpszString: PChar; hData: THandle): BOOL;
begin
   {add the property and its value to the listbox}
   Form1.ListBox1.Items.Add(Format('%s=%d',[lpszString,hData]));

   {continue property enumeration}
   Result:=TRUE;
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
   {clear the treeview object...}
   TreeView1.Items.Clear;

   {...and fill it with information about every window in the system}
   EnumWindows(@EnumerateWindows,0);
end;

procedure TForm1.TreeView1Click(Sender: TObject);
var
   TheWindow: HWND;                        // holds a window handle
   ParentWindow: HWND;                     // holds a parent window handle
   TheInstance: Longint;                   // holds an instance handle
   TheClassName: array[0..255] of char;    // holds the name of the class of a window
   TheClassInfo: TWndClass;                // holds information about a window class
   ErrorCode: Integer;                     // general error code variable
   BoolError: Boolean;                     // boolean error code variable
begin
   {get the window handle of the window selected in the treeview object}
   TheWindow:=HWND(StrToInt(Copy(TreeView1.Selected.Text,0,Pos
      ('-',TreeView1.Selected.Text)-2)));

   {if this window is a child window, retrieve a handle to its parent}
```

```
if (TreeView1.Selected.Parent<>nil) then
   ParentWindow:=HWND(StrToInt(Copy(TreeView1.Selected.Parent.Text,0,
                      Pos('-',TreeView1.Selected.Parent.Text)-2)))
else
   ParentWindow:=0;

{indicate if this window is a child window}
if IsChild(ParentWindow,TheWindow) then
   Shape1.Brush.Color := clRed
else
   Shape1.Brush.Color := clWhite;

{indicate if this window is minimized}
if IsIconic(TheWindow) then
   Shape2.Brush.Color := clRed
else
   Shape2.Brush.Color := clWhite;

{indicate if the TheWindow variable contains a valid window handle}
if IsWindow(TheWindow) then
   Shape3.Brush.Color := clRed
else
   Shape3.Brush.Color := clWhite;

{indicate if this window is enabled}
if IsWindowEnabled(TheWindow) then
   Shape4.Brush.Color := clRed
else
   Shape4.Brush.Color := clWhite;

{indicate if this window is a Unicode window}
if IsWindowUnicode(TheWindow) then
   Shape5.Brush.Color := clRed
else
   Shape5.Brush.Color := clWhite;

{indicate if this window is visible}
if IsWindowVisible(TheWindow) then
   Shape6.Brush.Color := clRed
else
   Shape6.Brush.Color := clWhite;

{indicate if this window is maximized}
if IsZoomed(TheWindow) then
   Shape7.Brush.Color := clRed
else
   Shape7.Brush.Color := clWhite;

{clear the property display list box...}
ListBox1.Items.Clear;

{...and display all of the property entries for the selected window}
EnumProps(TheWindow, @EnumProperties);

{clear the class information list box...}
ListBox2.Items.Clear;
```

```
{...and retrieve the class name of the selected window}
ErrorCode:=GetClassName(TheWindow,TheClassName,255);

{if there was an error retrieving the class name...}
if (ErrorCode=0) then
begin
  {...display and error message...}
  ShowMessage('GetClassName failed.  No class name available.');
  Exit;
end
else
  {...or display the class name of the selected window}
  ListBox2.Items.Add('This window is a '+string(TheClassName)+' class.');

{retrieve the instance handle associated with the selected window}
TheInstance:=GetWindowLong(TheWindow,GWL_HINSTANCE);

{if there was an error retrieving the instance handle...}
if (TheInstance=0) then
begin
  {...display an error message...}
  ShowMessage('GetWindowLong failed.  No application instance available.');
  Exit;
end
else
  {...or display the instance handle}
  ListBox2.Items.Add('Instance Handle: '+IntToStr(TheInstance));

{indicate if the retrieved instance handle is the same as the current instance}
if (TheInstance=hInstance) then
   ListBox2.Items.Add('This window belongs to the application instance');

{retrieve the class information for the class that the selected window belongs to}
BoolError:=GetClassInfo(TheInstance,TheClassName,TheClassInfo);

{if there was an error retrieving the class info...}
if (not BoolError) then
begin
     {...display an error message...}
     ListBox2.Items.Add('GetClassInfo failed.  No class information available.');
     Exit;
end;

{...otherwise, display the information on this class}
ListBox2.Items.Add('This class is defined as -');
ListBox2.Items.Add(Format('    Style: %d',[TheClassInfo.style]));
ListBox2.Items.Add(Format('    WndProc: %d',[integer(TheClassInfo.lpfnWndProc)]));
ListBox2.Items.Add(Format('    ClsExtra: %d',[TheClassInfo.cbClsExtra]));
ListBox2.Items.Add(Format('    WndExtra: %d',[TheClassInfo.cbWndExtra]));
ListBox2.Items.Add(Format('    Instance: %d',[integer(TheClassInfo.hInstance)]));
ListBox2.Items.Add(Format('    Icon: %d',[integer(TheClassInfo.hIcon)]));
ListBox2.Items.Add(Format('    Cursor: %d',[integer(TheClassInfo.hCursor)]));
ListBox2.Items.Add(Format('    Background: %d',[integer(TheClassInfo.hbrBackground)]));
if (TheClassInfo.lpszMenuName<>nil) then
  ListBox2.Items.Add('    Menu Name: '+TheClassInfo.lpszMenuName)
else
```

```
      ListBox2.Items.Add('    No class menu name');
    if (TheClassInfo.lpszClassName<>nil) then
      ListBox2.Items.Add('    Class Name: '+TheClassInfo.lpszClassName);

end;
```



```
ControlOfs00400000FFF1D211=12479156
DelphiFFF6CE6D=12479156
```

```
This window is a TStaticText class.
Instance Handle: 4194304
This window belongs to the application instance
This class is defined as -
   Style: 10
   WndProc: 4407424
   ClsExtra: 0
   WndExtra: 6
   Instance: 4194304
   Icon: 0
   Cursor: 5246
   Background: 0
   No class menu name
   Class Name: TStaticText
```

Figure 3-3: Information on every window in the system

Delphi vs. the Windows API

While the TForm class and other classes descended from the TWinControl class have many properties and methods that are useful for retrieving information about the window, there is a lot of functionality that is not encapsulated. To have the flexibility and functionality to manipulate windows in every manner available, programmers must use the Windows API.

Of particular interest are the get/set window/class long functions, as well as those functions that deal with windows properties. These groups of functions give the developer incredible power when manipulating a window. The get and set functions allow a developer to dynamically change window styles, which can ultimately give users more control over customizing the appearance of the application. The window property functions are also incredibly useful because they allow developers to attach an entire list of information to a window, as opposed to using the Tag property for tracking a single value. While several functions in this chapter are encapsulated by various methods and properties of Delphi components, many are not, and using the Windows API gives the developer maximum access to everything Windows has to offer when manipulating a window.

Window Information Functions

The following window information functions are covered in this chapter.

Table 3-1: Window information functions

Function	Description
AnyPopup	Indicates if any pop-up windows exist anywhere earlier on the screen.
ChildWindowFromPoint	Determines if a specific coordinate lies within any child windows.
ChildWindowFromPointEx	Determines if a specific coordinate lies within any child windows. This function can ignore invisible, disabled, or transparent child windows.
EnableWindow	Toggles the enable state of a window.
EnumChildWindows	Passes the handle of every child window belonging to the specified window to an application-defined callback function.
EnumProps	Passes the entries in a window property list to an application-defined callback function.
EnumPropsEx	Passes the entries in a window property list to an application-defined callback function. A user-defined value can be passed along with the property entry.
EnumThreadWindows	Passes the handle to every non-child window associated with a thread to an application-defined callback function.
EnumWindows	Passes the handle to every top-level window on the screen to an application-defined callback function.
FindWindow	Retrieves the handle to a top-level window.
FindWindowEx	Retrieves the handle to a child window.
FlashWindow	Toggles the caption bar color of a window.
GetActiveWindow	Retrieves the handle of the currently active window.
GetClassInfo	Retrieves information about the specified window's class.
GetClassInfoEx	Retrieves information about the specified window's class. This function can retrieve extended window styles and small cursor handles.
GetClassLong	Retrieves the value of the specified window's class.
GetClassName	Retrieves the name of the specified window's class.
GetClientRect	Retrieves the rectangular coordinates of the specified window's client area.
GetDesktopWindow	Retrieves a handle to the desktop window.
GetFocus	Retrieves the handle of the window with the keyboard focus.
GetForegroundWindow	Retrieves the handle of the current foreground window.
GetNextWindow	Retrieves the handle of the next or previous window in its relative z-order.
GetParent	Retrieves the handle of the specified window's parent window.
GetProp	Retrieves a property from the specified window's property list.
GetTopWindow	Retrieves the handle of the child window at the top of its relative z-order.
GetWindow	Retrieves the handle of the window with the specified relationship to the given window.
GetWindowLong	Retrieves a value of the window.
GetWindowRect	Retrieves the overall rectangular coordinates of the specified window.

Function	Description
GetWindowText	Retrieves the text displayed in the window.
GetWindowTextLength	Retrieves the length of the text displayed in the window.
IsChild	Determines if the specified window is a child window.
IsIconic	Determines if the specified window is minimized.
IsWindow	Determines if the specified handle is a valid window handle.
IsWindowEnabled	Determines if the specified window is enabled.
IsWindowUnicode	Determines if the specified window is a Unicode window.
IsWindowVisible	Determines if the specified window is visible.
IsZoomed	Determines if the specified window is maximized.
RemoveProp	Removes a property entry from the specified window's property list.
SetActiveWindow	Activates a window.
SetClassLong	Sets a value in the specified window's class.
SetFocus	Gives the specified window the keyboard input focus.
SetForegroundWindow	Activates a window and puts its thread into the foreground.
SetParent	Sets the parent window of the specified window.
SetProp	Adds a property entry into the specified window's property list.
SetWindowLong	Sets a specified value in the window.
SetWindowText	Sets the specified windows text to the given string.
WindowFromPoint	Retrieves the handle of the window containing the specified coordinates.

AnyPopup Windows.pas

Syntax

```
AnyPopup: BOOL;      {returns TRUE or FALSE}
```

Description

This will indicate whether an owned, visible, top-level pop-up, or overlapped window exists anywhere on the entire screen. However, it will not detect unowned pop-up windows or windows that do not have the WS_VISIBLE style specified. This is a function used mainly in earlier Windows applications and is retained for compatibility purposes.

Return Value

If this function succeeds and a pop-up window is found, this function returns TRUE, even if the pop-up is completely covered by other windows. If the function fails, or it does not find a pop-up window, it returns FALSE.

See Also

EnumWindows, FindWindow, FindWindowEx, GetTopWindow, GetWindow, ShowOwnedPopups

Example

Listing 3-5: Finding any pop-up window

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   if (AnyPopup) then
      Label1.Caption:='Pop-ups found: TRUE'
   else
      Label1.Caption:='Pop-ups found: FALSE';
end;
```

ChildWindowFromPoint Windows.pas

Syntax

ChildWindowFromPoint(	
hWndParent: HWND;	{the handle of the parent window}
Point: TPoint	{a data structure containing coordinates to check}
): HWND;	{returns a handle to a child window}

Description

This function determines if the specified point, containing coordinates relative to the parent window, falls inside the boundaries of any child window. It returns the handle to this child window even if it is disabled or hidden.

Parameters

hWndParent: A handle to the parent window.

Point: A variable of type TPoint defining the coordinates to be checked. These coordinates are relative to the parent window's client area.

Return Value

If this function succeeds, a handle to the child window containing the point is returned. If the point is within the boundaries of the parent window but not a child window, the return value is the handle to the parent window. If more than one child window contains the point, the return value is the first child window in the z-order. If the function fails, or the point is outside of the parent window boundaries, the return value is zero.

See Also

ChildWindowFromPointEx, WindowFromPoint, WM_LBUTTONDOWN, WM_MOUSEMOVE, WM_RBUTTONDOWN

Example

Listing 3-6: Finding a child window at a specific coordinate

The form for this example has a panel whose Align property is set to alTop. This is the panel that the following code will find.

```
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
   WindowText: array[0..255] of char;  // holds the text of the child window
   TheChildWnd: HWND;                  // holds the handle to the child window
   ThePoint: TPoint;                   // our coordinate structure
begin
   {we want to find the child window at coordinates 5,5 relative to the main form}
   ThePoint.X:=5;
   ThePoint.Y:=5;

   {retrieve the child window handle at these coordinates, if any}
   TheChildWnd:=ChildWindowFromPoint(Form1.Handle,ThePoint);

   {if we found a child window...}
   if (TheChildWnd<>0) then
   begin
      {...display its text...}
      GetWindowText(TheChildWnd, WindowText, 255);
      Button1.Caption:=WindowText;
   end
   else
      {...or display a message}
      Button1.Caption:='No Child Window Found.';
end;
```

ChildWindowFromPointEx Windows.pas

Syntax

```
ChildWindowFromPointEx(
hWnd: HWND;          {the handle of the parent window}
Point: TPoint;       {a data structure with coordinates to be checked}
Flags: UINT          {disregard flags}
): HWND;             {returns a handle to a child window}
```

Description

This function determines if the specified point, containing coordinates relative to the parent window, falls inside the boundaries of any child window. Functionally equivalent to ChildWindowFromPoint, this function can skip invisible, disabled, or transparent child windows.

Parameters

hWndParent: A handle to the parent window.

Point: A variable of type TPoint defining the coordinates to be checked. These coordinates are relative to the parent window's client area.

Flags: A 32-bit number specifying which child windows to skip. This parameter can be one or more values from the following table.

Return Value

If this function succeeds, a handle to the child window containing the point and meeting the criteria in Flags is returned. If the point is within the boundaries of the parent window but not any child window meeting the criteria in Flags, the return value is the handle to the parent window. If more than one child window contains the point, the return value is the first child window in the z-order. If the function fails, or the point is outside of the parent window boundaries, the return value is zero.

See Also

ChildWindowFromPoint, WindowFromPoint, WM_LBUTTONDOWN, WM_MOUSEMOVE, WM_RBUTTONDOWN

Example

Listing 3-7: Finding a child window at specific coordinates

The form for this example has a panel whose Align property is set to alTop. This is the panel that the following code will find.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   WindowText: array[0..255] of char;  // holds the text of the child window
   TheChildWnd: HWND;                  // holds the handle to the child window
   ThePoint: TPoint;                   // our coordinate structure
begin
   {we want to find the child window at coordinates 5,5 relative to the main form}
   ThePoint.X:=5;
   ThePoint.Y:=5;

   {retrieve the child window handle at these coordinates, if any}
   TheChildWnd:=ChildWindowFromPointEx(Form1.Handle,ThePoint,CWP_ALL);

   {if we found a child window...}
   if (TheChildWnd<>0) then
   begin
      {...display its text...}
      GetWindowText(TheChildWnd, WindowText, 255);
      Button1.Caption:=WindowText;
   end
   else
      {...or display a message}
      Button1.Caption:='No Child Window Found.';
end;
```

Table 3-2: ChildWindowFromPointEx flags values

Value	Description
CWP_ALL	Do not skip any child windows.
CWP_SKIPINVISIBLE	Skip invisible child windows.
CWP_SKIPDISABLED	Skip disabled child windows.
CWP_SKIPTRANSPARENT	Skip transparent child windows.

EnableWindow Windows.pas

Syntax

```
EnableWindow(
hWnd: HWND;           {a handle to a window}
bEnable: BOOL         {enable/disable flag}
): BOOL;              {returns TRUE or FALSE}
```

Description

This function enables or disables mouse and keyboard input to the specified window or control. When disabled, a window or control will not receive any input, such as mouse clicks or keypresses, and generally cannot be accessed by the user. If the enabled state of a window or control is changing, the WM_ENABLE message is sent before this function returns. If a disabled window contains child windows, all of those child windows are disabled, but they are not sent the WM_ENABLE message. A disabled window must be enabled before it can be activated.

Parameters

hWnd: A handle to the window to be enabled or disabled.

bEnable: If this parameter is TRUE, the window is enabled; if it is FALSE, the window will be disabled.

Return Value

This function returns TRUE if the window was already disabled; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetActiveWindow, GetFocus, IsWindowEnabled, SetActiveWindow, SetFocus, WM_ENABLE

Example

Listing 3-8: Enabling and disabling a window

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   {if the edit box is currently enabled...}
   if (IsWindowEnabled(Edit1.Handle)) then
   begin
      {...disable it and modify the appropriate captions...}
      EnableWindow(Edit1.Handle,FALSE);
      Button1.Caption:='Enable Window';
      Edit1.Text:='This window is disabled';
   end
   else
   begin
      {...otherwise enable it and modify the appropriate captions}
      EnableWindow(Edit1.Handle,TRUE);
      Button1.Caption:='Disable Window';
      Edit1.Text:='This window is enabled';
   end;
end;
```

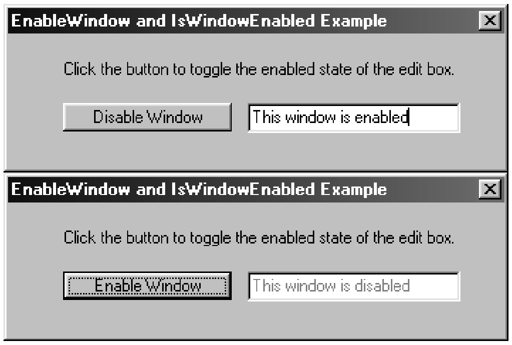

EnableWindow and IsWindowEnabled Example
Click the button to toggle the enabled state of the edit box.
Disable Window
This window is enabled
EnableWindow and IsWindowEnabled Example
Click the button to toggle the enabled state of the edit box.
Enable Window
This window is disabled

Figure 3-4: Enabled/ disabled window states

EnumChildWindows *Windows.pas*

Syntax

```
EnumChildWindows(
hWndParent: HWND;               {the handle of the parent window}
lpEnumFunc: TFNWndEnumProc;     {a pointer to the callback function}
lParam: LPARAM                  {an application-defined 32-bit value}
): BOOL;                        {returns TRUE or FALSE}
```

Description

EnumChildWindows parses through all of the child windows belonging to the parent window, sending the handle of each child window to an application-defined callback function. It continues until all child windows have been enumerated or the callback function returns FALSE. If a child window has created child windows of its own, these child windows are enumerated as well. This function will ignore child windows that have been destroyed and those that have been created during the enumeration process.

Parameters

hWndParent: The handle of the parent window whose child windows are to be enumerated.

lpEnumFunc: The address of the application-defined callback function.

lParam: A 32-bit application-defined value that will be passed to the callback function.

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE.

Callback Syntax

```
EnumChildProc(
hWnd: HWND;        {a handle to a child window}
lParam: LPARAM     {an application-defined 32-bit value}
): BOOL;           {returns TRUE or FALSE}
```

Description

This function receives the window handle for each child window belonging to the parent window specified in the call to EnumChildWindows. It may perform any desired task.

Parameters

hWnd: The handle of a child window.

lParam: A 32-bit application-defined number. This value is intended for application-specific use inside of the callback function, and it is the value of the lParam parameter passed to the EnumChildWindows function.

Return Value

The callback function should return TRUE to continue enumeration; otherwise, it should return FALSE.

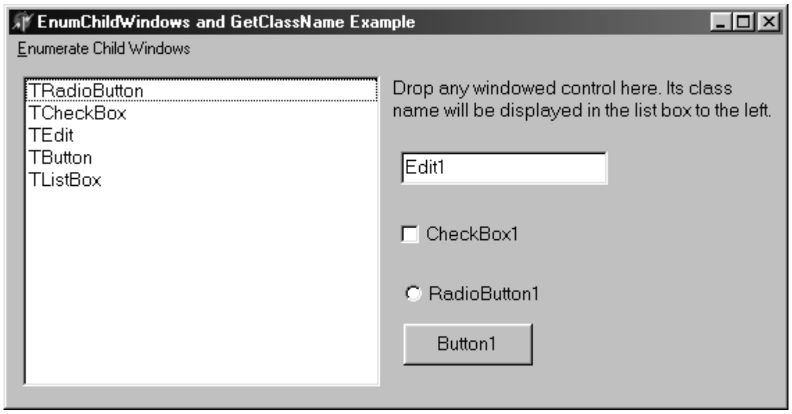
See Also

EnumThreadWindows, EnumWindows, FindWindow, FindWindowEx, GetWindow, GetParent, IsChild

Example

Listing 3-9: Enumerating child windows

```
{our callback function prototype}
function EnumerateChildWindows(hWnd:HWND; lParam:LPARAM): BOOL; stdcall;

var
  Form1: TForm1;

implementation

procedure TForm1.EnumerateChildWindows1Click(Sender: TObject);
begin
   {empty our list box}
   ListBox1.Items.Clear;

   {enumerate all child windows belonging to Form1}
   EnumChildWindows(Form1.Handle,@EnumerateChildWindows,0);
end;

{these steps execute for every child window belonging to the parent}
function EnumerateChildWindows(hWnd: HWND; lParam: LPARAM): BOOL;
var
   ClassName: Array[0..255] of char;  // this holds the class name of our child windows
begin
   {get the class name of the given child window}
   GetClassName(hWnd,ClassName,255);

   {display it in the list box}
   Form1.ListBox1.Items.Add(ClassName);

   {continue enumeration}
   Result:=TRUE;
end;
```



```
TRadioButton
TCheckBox
TEdit
TButton
TListBox
```

Drop any windowed control here. Its class name will be displayed in the list box to the left.

Figure 3-5: Child window class names

EnumProps Windows.pas

Syntax

EnumProps(	
hWnd: HWND;	{a handle to a window}
lpEnumFunc: TFNPropEnumProc	{the address of the enumeration callback function}
): Integer;	{returns the value returned from callback function}

Description

This function passes each entry in the property list of the specified window to an application-defined callback function. This continues until all properties have been enumerated or the callback function returns FALSE. This function is intended to be used to find the data associated with a window without knowing how many property entries exist.

Parameters

hWnd: The handle of the window whose property list is to be enumerated.

lpEnumFunc: The address of the application-defined callback function that receives the property list entries.

Return Value

This function returns the last value returned by the callback function. If the function fails, or the callback function did not find a property for the specified window, the value is –1. Note that the callback function returns a value of type BOOL, which the Windows.pas file defines as a LongBool. This type exists for compatibility reasons and holds a Longint value, where a value of 0 is assumed to mean FALSE and non-zero values are assumed to mean TRUE.

Callback Syntax

```
EnumPropProc(
hWnd: HWND;                 {the handle to the window with properties}
lpszPropString: PChar;      {a pointer to a null-terminated string}
hData: THandle              {the data component of a property list entry}
): BOOL;                    {returns TRUE or FALSE}
```

Description

This function receives property entry information for each property in the property list of the specified window. While this function is running, it should not yield control to any other process or attempt to add a property entry. It can call RemoveProp to remove a property entry, but it can only remove the current entry passed to the function.

Parameters

hWnd: The handle of the window whose property list is being enumerated.

lpszPropString: A pointer to a null-terminated string. This is the string component of the property list entry that was added by a call to the SetProp function.

hData: The 32-bit value that is the data component of the property list entry that was added by a call to the SetProp function.

Return Value

The callback function should return TRUE to continue enumeration; otherwise, it should return FALSE.

See Also

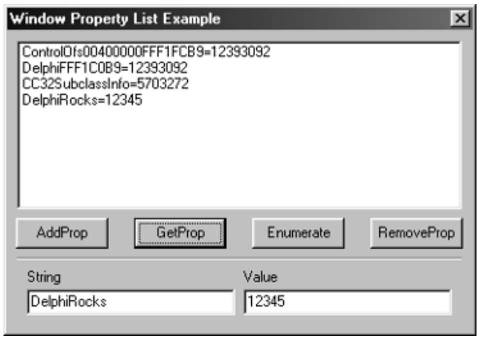
EnumPropsEx, GetProp, RemoveProp, SetProp

Example

Listing 3-10: Enumerating the property entries in a window property list

```
{our callback function prototype}
function EnumWinProps(hWnd: HWND; pString: PChar; Data: THandle): BOOL; stdcall;

var
  Form1: TForm1;

implementation

{these steps will be executed for each property entry in the window's property list}
function EnumWinProps(hWnd: HWND; pString: PChar; Data: THandle): BOOL;
begin
   {add the string and associated value to the list box}
   Form1.ListBox1.Items.Add(Format('%s=%d',[pString,Data]));

   {continue enumeration}
   Result:=TRUE;
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   {add a new property to the window's property list}
   SetProp(Form1.Handle,PChar(Edit1.Text),StrToInt(Edit2.Text));

   {clear the edit boxes}
   Edit1.Text:='';
   Edit2.Text:='';

   {clear the list box}
   Form1.ListBox1.Items.Clear;

   {list all of the properties associated with the window}
   EnumProps(Form1.Handle, @EnumWinProps);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
   {clear the list box}
   Form1.ListBox1.Items.Clear;

   {list all of the properties associated with the window}
   EnumProps(Form1.Handle, @EnumWinProps);
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
   {remove the selected property from the property list}
   RemoveProp(Form1.Handle,PChar(Copy(ListBox1.Items[ListBox1.ItemIndex],
              0,Pos('=',ListBox1.Items[ListBox1.ItemIndex])-1)));

   {clear the list box}
   Form1.ListBox1.Items.Clear;

   {list all of the properties associated with the window}
   EnumProps(Form1.Handle, @EnumWinProps);
end;

procedure TForm1.Button4Click(Sender: TObject);
var
   Data: THandle;  // this stores the property entry data
begin
   {get property entry data associated with the given string}
   Data:=GetProp(Form1.Handle,PChar(Edit1.Text));

   {if there was a property value returned...}
   if (Data<>0) then
      {...display it...}
      Edit2.Text:=IntToStr(Data)
   else
      {...otherwise display an error message}
      Edit2.Text:='No property found.';
end;
```



```
ControlOfs00400000FFF1FCB9=12393092
DelphiFFF1C0B9=12393092
CC32SubclassInfo=5703272
DelphiRocks=12345
```

String: DelphiRocks
Value: 12345

Figure 3-6: Window property list

EnumPropsEx Windows.pas

Syntax

```
EnumPropsEx(
hWnd: HWND;                       {a handle to a window}
lpEnumFunc: TFNPropEnumProcEx;    {the enumeration callback function address}
lParam: LPARAM                    {a 32-bit application-defined value}
): Integer;                       {returns the value returned from callback
                                   function}
```

Description

This function passes each entry in the property list of the specified window to an application-defined callback function. This continues until all properties have been enumerated or the callback function returns FALSE. This function is intended to be used to find the data associated with a window without knowing how many property entries exist. This is functionally equivalent to EnumProps, except there is an extra parameter for user-defined values that are passed to the callback function.

Parameters

hWnd: The handle of the window whose property list is to be enumerated.

lpEnumFunc: The address of the application-defined callback function that receives the property list entries.

lParam: A 32-bit application-defined value that is passed to the callback function.

Return Value

This function returns the last value returned by the callback function. If the function fails, or the callback function did not find a property for the specified window, the value is –1. Note that the callback function returns a value of type BOOL, which the Windows.pas file defines as a LongBool. This type exists for compatibility reasons and holds a Longint value, where a value of 0 is assumed to mean FALSE and non-zero values are assumed to mean TRUE.

Callback Syntax

EnumPropProcEx(	
hWnd: HWND;	{the handle to the window with properties}
lpszPropString: PChar;	{a pointer to a null-terminated string}
hData: Handle;	{the data component of a property list entry}
dwData: DWORD	{the application-defined data}
): BOOL;	{returns TRUE or FALSE}

Description

This function receives property entry information for each property in the property list of the specified window. While this function is running, it should not yield control to any other process or attempt to add a property entry. It can call RemoveProp to remove a property entry, but it can only remove the current entry passed to the function.

Parameters

hWnd: The handle of the window whose property list is being enumerated.

lpszPropString: A pointer to a null-terminated string. This is the string component of the property list entry that was added by a call to the SetProp function.

hData: The 32-bit value that is the data component of the property list entry that was added by a call to the SetProp function.

dwData: A 32-bit application-defined value. This value is intended for application-specific use inside of the callback function and is the value of the lParam parameter passed to the EnumPropsEx function.

Return Value

The callback function should return TRUE to continue enumeration; otherwise, it should return FALSE.

See Also

EnumProps, GetProp, RemoveProp, SetProp

Example

Listing 3-11: Enumerating window property entries with user data

```
{our callback function prototype}
function EnumWinPropsEx(hWnd: HWND; pString: PChar; Data: THandle;
                        dwData: DWORD): BOOL; stdcall;

var
  Form1: TForm1;

implementation

{these steps will be executed for each property entry in the window's property list}
function EnumWinPropsEx(hWnd: HWND; pString: PChar; Data: THandle; dwData: DWORD): BOOL;
begin
   {add the string and associated value to the list box}
   Form1.ListBox1.Items.Add(Format('%s=%d, User Data: %d', [pString,Data,dwData]));

   {continue enumeration}
   Result:=TRUE;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
   {add a new property to the window's property list}
   SetProp(Form1.Handle,PChar(Edit1.Text),StrToInt(Edit2.Text));

   {clear the edit boxes}
   Edit1.Text:='';
   Edit2.Text:='';

   {clear the list box}
   Form1.ListBox1.Items.Clear;

   {list all of the properties associated with the window}
   EnumPropsEx(Form1.Handle, @EnumWinPropsEx, 1);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
   {clear the list box}
   Form1.ListBox1.Items.Clear;

   {list all of the properties associated with the window}
   EnumPropsEx(Form1.Handle, @EnumWinPropsEx, 2);
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
   {remove the selected property from the property list}
   RemoveProp(Form1.Handle,PChar(Copy(ListBox1.Items[ListBox1.ItemIndex],
              0,Pos('=',ListBox1.Items[ListBox1.ItemIndex])-1)));

   {clear the list box}
   Form1.ListBox1.Items.Clear;

   {list all of the properties associated with the window}
   EnumPropsEx(Form1.Handle, @EnumWinPropsEx, 3);
end;

procedure TForm1.Button4Click(Sender: TObject);
var
   Data: THandle;  // this stores the property entry data
begin
   {get property entry data associated with the given string}
   Data:=GetProp(Form1.Handle,PChar(Edit1.Text));

   {if there was a property value returned...}
   if (Data<>0) then
      {...display it...}
      Edit2.Text:=IntToStr(Data)
   else
```

```
      {...otherwise display an error message}
      Edit2.Text:='No property found.';
end;
```

EnumThreadWindows Windows.pas

Syntax

EnumThreadWindows(	
dwThreadId: DWORD;	{the thread identification number}
lpfn: TFNWndEnumProc;	{the address of the enumeration callback function}
lParam: LPARAM	{a 32-bit application-defined value}
): BOOL;	{returns TRUE or FALSE}

Description

This function enumerates all of the non-child windows associated with the specified thread. Each window handle associated with the specified thread is passed to an application-defined callback function. This function will continue until all of the windows are enumerated or the callback function returns FALSE.

Parameters

dwThreadId: The thread whose windows are to be enumerated.

lpfn: The address of the application-defined callback function.

lParam: A 32-bit application-defined value that is passed to the callback function.

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE.

Callback Syntax

EnumThreadWndProc(	
hWnd: HWND;	{a handle to a window}
lParam: LPARAM	{the application-defined data}
): BOOL;	{returns TRUE or FALSE}

Description

This function receives a window handle for every window associated with the given thread and can perform any desired task.

Parameters

hWnd: The handle of a window associated with the specified thread.

lParam: A 32-bit application-defined value. This value is intended for application-specific use inside of the callback function and is the value of the lParam parameter of the EnumThreadWindows function.

Return Value

The callback function should return TRUE to continue enumeration; otherwise, it should return FALSE.

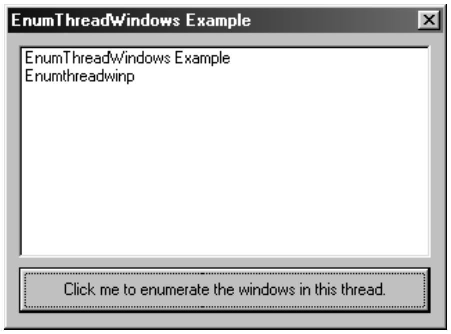
See Also

EnumChildWindows, EnumWindows, GetCurrentThreadID*, GetWindowThreadProcessId*

Example

Listing 3-12: Finding all windows belonging to a thread

```
{our callback function prototype}
function EnumerateThreadWindows(Wnd: HWND; Data: LPARAM): BOOL; stdcall;

var
  Form1: TForm1;

implementation

procedure TForm1.Button1Click(Sender: TObject);
begin
   {clear the listbox}
   ListBox1.Items.Clear;

   {enumerate all windows that belong to the current thread}
   EnumThreadWindows(GetCurrentThreadID, @EnumerateThreadWindows, 0);
end;

{theses steps are performed for every window belonging to the current thread}
function EnumerateThreadWindows(Wnd: HWND; Data: LPARAM): BOOL;
var
   WindowText: array[0..255] of char; // holds the text of the window
begin
   {get the text from the window...}
   GetWindowText(Wnd, WindowText, 255);

   {...and display it in the listbox}
   Form1.ListBox1.Items.Add(WindowText);

   {continue enumeration}
   Result:=TRUE;
end;
```


EnumThreadWindows Example

EnumThreadWindows Example
Enumthreadwinp

Figure 3-7: Windows belonging to the current thread

EnumWindows Windows.pas

Syntax

```
EnumWindows(
lpEnumFunc: TFNWndEnumProc;      {address of enumeration callback function}
lParam: LPARAM                   {a 32-bit application-defined value}
): BOOL;                         {returns TRUE or FALSE}
```

Description

This function parses through all top-level windows on the screen, passing the handle of each window to an application-defined callback function. This continues until all top-level windows have been enumerated or the callback function returns FALSE. The EnumWindows function does not enumerate child windows.

Parameters

lpEnumFunc: The address of the application-defined callback function.

lParam: A 32-bit application-defined value that will be passed to the callback function.

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE.

Callback Syntax

```
EnumWindowsProc(
hWnd: HWND;          {a handle to a top-level window}
lParam: LPARAM       {the application-defined data}
): BOOL;             {returns TRUE or FALSE}
```

Description

This function receives the window handle for each top-level window in the system, and it may perform any desired task.

Parameters

hWnd: The handle of a top-level window being enumerated.

lParam: A 32-bit application-defined value. This value is intended for application-specific use inside of the callback function and is the value of the lParam parameter passed to the EnumWindows function.

Return Value

The callback function should return TRUE to continue enumeration; otherwise, it should return FALSE.

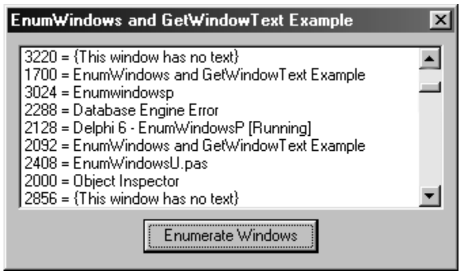
See Also

EnumChildWindows, EnumThreadWindows, FindWindow, FindWindowEx, GetTopWindow, GetWindow

Example

Listing 3-13: Listing the window text for every top-level window in the system

```
{our callback function prototype}
function EnumerateWindows(hWnd: HWND; lParam: LPARAM): BOOL; stdcall;

var
  Form1: TForm1;

implementation

procedure TForm1.Button1Click(Sender: TObject);
begin
   {empty the listbox that will hold the window names}
   ListBox1.Items.Clear;

   {enumerate all the top-level windows in the system}
   EnumWindows(@EnumerateWindows,0);
end;

{these steps execute for every top-level window in the system}
function EnumerateWindows(hWnd: HWND; lParam: LPARAM): BOOL;
var
   TheText: Array[0..255] of char;  // this holds the window text
begin
   {if the window does not have any text...}
   if (GetWindowText(hWnd, TheText, 255)=0) then
      {...display the window handle and a note...}
      Form1.ListBox1.Items.Add(Format('%d = {This window has no text}',[hWnd]))
   else
      {otherwise display the window handle and the window text}
      Form1.ListBox1.Items.Add(Format('%d = %s',[hWnd,TheText]));

   {continue enumeration}
   Result:=TRUE;
end;
```



```
3220 = {This window has no text}
1700 = EnumWindows and GetWindowText Example
3024 = Enumwindowsp
2288 = Database Engine Error
2128 = Delphi 6 - EnumWindowsP [Running]
2092 = EnumWindows and GetWindowText Example
2408 = EnumWindowsU.pas
2000 = Object Inspector
2856 = {This window has no text}
```

Figure 3-8: All top-level windows

FindWindow Windows.pas

Syntax

FindWindow(	
lpClassName: PChar;	{a pointer to a null-terminated class name string}
lpWindowName: PChar	{a pointer to a null-terminated window name string}
): HWND;	{returns a handle to a window}

Description

FindWindow retrieves the handle of the top-level window with the specified class name and window name. Child windows are not searched.

Parameters

lpClassName: A pointer to a case-sensitive, null-terminated string that specifies the class name, or an integer atom identifying the class name string. If this specifies an atom, the atom must be created with a call to GlobalAddAtom. The atom, a 16-bit value, must be in the low-order word of ClassName and the high-order word must be zero.

lpWindowName: A pointer to a case-sensitive, null-terminated string that specifies the window's name, which is the title in the caption bar. If this parameter is NIL, all window names match.

Return Value

If this function succeeds, the return value is the handle of the window with the specified class name and window name; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

EnumWindows, FindWindowEx, GetClassName, GetWindow

Example

Listing 3-14: Finding a window

```
procedure TForm1.Button1Click(Sender: TObject);
var
   TheWindow: HWND;                   // holds the window handle found
   WindowText: array[0..255] of char; // holds the window's text
begin
   {find a handle to the Delphi IDE window}
   TheWindow := FindWindow('TAppBuilder', nil);

   {retrieve its text}
   GetWindowText(TheWindow, @WindowText[0], 255);

   {display the text}
   Button1.Caption := WindowText;
end;
```

FindWindowEx Windows.pas

Syntax

FindWindowEx(	
Parent: HWND;	{a handle to a parent window}
Child: HWND;	{a handle to a child window}
ClassName: PChar;	{a pointer to a null-terminated class name string}
WindowName: PChar	{a pointer to a null-terminated window name string}
): HWND;	{returns a handle to a window}

Description

This function retrieves the handle of the window with the specified class name and window name. Unlike FindWindow, this function searches child windows, starting with the one following the given child window.

Parameters

Parent: The handle of the parent window whose child windows are to be searched. If this parameter is zero, the desktop window is used as the parent and the child windows of the desktop are searched.

Child: The handle of a child window. The search will begin with the next child window in the z-order of the specified child window. The specified child window must be a direct child window of the window defined by the Parent parameter. If this parameter is zero, the search will start with the first child window in the parent window. Note that if this parameter and the Parent parameter are both zero, this function searches all top-level windows.

ClassName: A pointer to a case-sensitive, null-terminated string that specifies the class name or an integer atom identifying the class name string. If this specifies an atom, the atom must be created with a call to GlobalAddAtom. The atom, a 16-bit value, must be in the low-order word of ClassName, and the high-order word must be zero.

WindowName: A pointer to a case-sensitive, null-terminated string that specifies the window's name (the window text). If this parameter is NIL, all window names match.

Return Value

If this function succeeds, the return value is the handle of the window with the specified class name and window name; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

EnumChildWindows, EnumWindows, FindWindow, GetClassName, GetWindow

Example

Listing 3-15: Using FindWindowEx to find a child window

```
procedure TForm1.Button1Click(Sender: TObject);
var
   FoundWindow: HWND;                  // holds a window handle
   WindowText: array[0..255] of char;   // holds the window text
begin
   {find a TEdit child window}
   FoundWindow := FindWindowEx(Form1.Handle, 0, 'TEdit', nil);

   {get its text...}
   GetWindowText(FoundWindow, WindowText, 255);

   {...and display it}
   Label1.Caption:='FindWindowEx found window handle '+IntToStr(FoundWindow)+
                   ': '+WindowText;
end;
```

FlashWindow Windows.pas

Syntax

FlashWindow(	
hWnd: HWND;	{the handle to the window to flash}
bInvert: BOOL	{flash flag}
): BOOL;	{returns TRUE or FALSE}

Description

This function will flash the window from an inactive to active state, or vice versa. It is flashed only once, and the window can be opened or minimized.

Parameters

hWnd: The handle of the window to be flashed.

bInvert: A Boolean value specifying how the window is to be flashed. A value of TRUE will cause the window to be flashed from one state to the other (i.e., inactive to active). A value of FALSE causes the window to flash back to its original state.

Return Value

If the function succeeds and the window was active before the call to this function, the return value is TRUE. If the function fails, or if the function succeeds and the window was inactive before calling this function, it returns FALSE.

See Also

GetActiveWindow, GetFocus, SetActiveWindow, SetFocus

Example

Listing 3-16: Flashing a window

Note that this code is put into an OnTimer event of a TTimer set to fire once every 1,000 milliseconds.

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
   {flash the main form}
   FlashWindow(Form1.Handle, TRUE);
   {this is necessary under Delphi to get the icon on the taskbar to flash}
   FlashWindow(Application.handle, TRUE);
end;
```

GetActiveWindow Windows.pas

Syntax

GetActiveWindow: HWND; {returns a handle to the active window}

Description

This function returns a handle to the active window associated with the thread that calls the function.

Return Value

If this function succeeds, the return value is a handle to the active window associated with the thread that called the function. If the function fails, or if the thread does not have an active window, the return value is zero.

See Also

GetFocus, GetForegroundWindow, GetTopWindow, SetActiveWindow, SetFocus, SetForegroundWindow

Example

Listing 3-17: Retrieving a handle to the currently active window

```
procedure TForm1.Button1Click(Sender: TObject);
var
   TheWindow: HWND;                   // this will hold the active window handle
   WindowText: array[0..255] of char; // this will hold the text of that window
begin
   {get the handle to the active window associated with this thread}
   TheWindow := GetActiveWindow;

   {get the text of that window}
   GetWindowText(TheWindow, WindowText, 255);

   {display the text}
   Label1.Caption := 'Active Window Text: ' + string(WindowText);
end;
```

GetClassInfo Windows.pas

Syntax

GetClassInfo(
hInstance: HINST; {a handle to an application instance}
lpClassName: PChar; {a pointer to a null-terminated class name string}
var lpWndClass: TWndClass {a pointer to a TWndClass structure}
): BOOL; {returns TRUE or FALSE}

Description

This function returns information about the given window class. This information is returned in the members of the lpWndClass variable, a TWndClass data structure, and is the same information passed to the RegisterClass function that created the class.

Parameters

hInstance: The instance handle of the application that created the class. To get information about classes defined by Windows, such as buttons or list boxes, set this parameter to zero.

lpClassName: A pointer to a null-terminated string that contains the name of the class, either an application-defined name used in the RegisterClass function or the name of a preregistered window class. This can also be an integer atom, created with a call to GlobalAddAtom. The atom, a 16-bit value less than $C000, must be in the low-order word, and the high-order word must be zero.

lpWndClass: A pointer to a TWndClass structure that will receive the information about the specified class. The TWndClass structure is defined by Delphi as:

```
TWndClass = packed record
     Style: UINT;                    {class style flags}
     lpfnWndProc: TFNWndProc;        {a pointer to the window procedure}
     cbClsExtra: Integer;            {extra class memory bytes}
     cbWndExtra: Integer;            {extra window memory bytes}
     hInstance: HINST;               {a handle to the module instance}
     hIcon: HICON;                   {a handle to an icon}
     hCursor: HCURSOR;               {a handle to a cursor}
     hbrBackground: HBRUSH;          {a handle to the background brush}
     lpszMenuName: PAnsiChar;        {the menu name}
     lpszClassName: PAnsiChar;       {the class name}
end;
```

The TWndClass structure is described under the RegisterClass function in *The Tomes of Delphi: Win32 Core API — Windows 2000 Edition*.

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

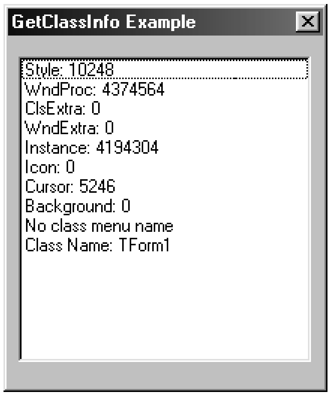
GetClassInfoEx, GetClassLong, GetClassName, RegisterClass*

Example

Listing 3-18: Retrieving information about the main form's class

```
procedure TForm1.FormActivate(Sender: TObject);
var
   ClassInfo: TWndClass;  // this will hold our class information
begin
```

```
   {get the information for our main form's class}
   GetClassInfo(hInstance,'TForm1',ClassInfo);

   {empty the list box}
   ListBox1.Items.Clear;

   {display all of the information about the main form's class}
   ListBox1.Items.Add(Format('Style: %d',[ClassInfo.style]));
   ListBox1.Items.Add(Format('WndProc: %d',[integer(ClassInfo.lpfnWndProc)]));
   ListBox1.Items.Add(Format('ClsExtra: %d',[ClassInfo.cbClsExtra]));
   ListBox1.Items.Add(Format('WndExtra: %d',[ClassInfo.cbWndExtra]));
   ListBox1.Items.Add(Format('Instance: %d',[integer(ClassInfo.hInstance)]));
   ListBox1.Items.Add(Format('Icon: %d',[integer(ClassInfo.hIcon)]));
   ListBox1.Items.Add(Format('Cursor: %d',[integer(ClassInfo.hCursor)]));
   ListBox1.Items.Add(Format('Background: %d',[integer(ClassInfo.hbrBackground)]));
   if (ClassInfo.lpszMenuName<>nil) then
      ListBox1.Items.Add('Menu Name: '+ClassInfo.lpszMenuName)
   else
      ListBox1.Items.Add('No class menu name');
   if (ClassInfo.lpszClassName<>nil) then
      ListBox1.Items.Add('Class Name: '+ClassInfo.lpszClassName);
end;
```



```
Style: 10248
WndProc: 4374564
ClsExtra: 0
WndExtra: 0
Instance: 4194304
Icon: 0
Cursor: 5246
Background: 0
No class menu name
Class Name: TForm1
```

Figure 3-9: The main form's class information

GetClassInfoEx Windows.pas

Syntax

GetClassInfoEx(	
Instance: HINST;	{a handle to an application instance}
ClassName: PChar;	{a pointer to a null-terminated class name string}
var WndClass: TWndClassEx	{a pointer to a TWndClassEx structure}
): BOOL;	{returns TRUE or FALSE}

Description

This function returns information about the given window class. This information is returned in the members of the WndClass variable, a TWndClassEx data structure, and is the same information passed to the RegisterClassEx function that created the class. This function is equivalent to GetClassInfo, except it returns a handle to the small icon associated with the given class.

Parameters

Instance: The instance handle of the application that created the class. To get information about classes defined by Windows, such as buttons or list boxes, set this parameter to zero.

ClassName: A pointer to a null-terminated string that contains the name of the class, either an application-defined name used in the RegisterClass function or the name of a preregistered window class. This can also be an integer atom, created with a call to GlobalAddAtom. The atom, a 16-bit value less than $C000, must be in the low-order word and the high-order word must be zero.

WndClass: A pointer to a TWndClassEx structure that will receive the information about the specified class. The TWndClassEx structure is defined by Delphi as:

```
TWndClassEx = packed record
    cbSize: UINT;                    {the size of this structure}
    Style: UINT;                     {class style flags}
    lpfnWndProc: TFNWndProc;         {a pointer to the window procedure}
    cbClsExtra: Integer;             {extra class memory bytes}
    cbWndExtra: Integer;             {extra window memory bytes}
    hInstance: HINST;                {a handle to the module instance}
    hIcon: HICON;                    {a handle to an icon}
    hCursor: HCURSOR;                {a handle to a cursor}
    hbrBackground: HBRUSH;           {a handle to the background brush}
    lpszMenuName: PAnsiChar;         {the menu name}
    lpszClassName: PAnsiChar;        {the class name}
    hIconSm: HICON;                  {a handle to a small icon}
end;
```

The TWndClassEx structure is described under the RegisterClassEx function in *The Tomes of Delphi: Win32 Core API — Windows 2000 Edition*. Before calling the GetClassInfoEx function, the cbSize member of this structure must be set to SizeOf(TWndClassEx).

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetClassInfo, GetClassLong, GetClassName, RegisterClassEx*

Example

Listing 3-19: Retrieving class information for all child windows

```
{function prototype for enumerating child windows}
function EnumerateChildWindows(hWnd:HWND; lParam:LPARAM): BOOL; stdcall;

var
  Form1: TForm1;

implementation

function EnumerateChildWindows(hWnd: HWND; lParam: LPARAM): BOOL;
var
   TheClassName: Array[0..255] of char;
begin
   {get the class name of this child window}
   GetClassName(hWnd, TheClassName, 255);

   {display it in the list box}
   Form1.ListBox1.Items.Add(TheClassName);

   {continue enumeration}
   Result:=TRUE;
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
   {populate the list box with the class names of all child windows}
   EnumChildWindows(Form1.Handle,@EnumerateChildWindows,0);
end;

procedure TForm1.ListBox1Click(Sender: TObject);
var
   ClassInfo: TWndClassEx;             // this holds our class info
   ClassName: array[0..255] of char;   // this holds the class name
begin
   {copy the class name to a PChar string that is passed to the GetClassInfoEx
    function. the classname parameter must be a PChar or the memory pointed
    to by ClassInfo becomes corrupted when accessing certain members of the
    data structure.}
   StrPCopy(ClassName,ListBox1.Items[ListBox1.ItemIndex]);

   {set the size of the data structure}
   ClassInfo.cbSize:=SizeOf(TWndClassEx);

   {get the class information for the selected class}
   GetClassInfoEx(hInstance,ClassName,ClassInfo);

   {clear the list box}
   ListBox2.Items.Clear;

   {display the class information}
   ListBox2.Items.Add(Format('Size: %d',[ClassInfo.cbSize]));
   ListBox2.Items.Add(Format('Style: %d',[ClassInfo.Style]));
   ListBox2.Items.Add(Format('WndProc: %d',[integer(ClassInfo.lpfnWndProc)]));
```

```
    ListBox2.Items.Add(Format('ClsExtra: %d',[ClassInfo.cbClsExtra]));
    ListBox2.Items.Add(Format('WndExtra: %d',[ClassInfo.cbWndExtra]));
    ListBox2.Items.Add(Format('Instance: %d',[integer(ClassInfo.hInstance)]));
    ListBox2.Items.Add(Format('Icon: %d',[integer(ClassInfo.hIcon)]));
    ListBox2.Items.Add(Format('Cursor: %d',[integer(ClassInfo.hCursor)]));
    ListBox2.Items.Add(Format('Background: %d',[integer(ClassInfo.hbrBackground)]));
    if (ClassInfo.lpszMenuName<>nil) then
       ListBox2.Items.Add('Menu Name: '+ClassInfo.lpszMenuName)
    else
       ListBox2.Items.Add('No class menu name');
    if (ClassInfo.lpszClassName<>nil) then
       ListBox2.Items.Add('Class Name: '+ClassInfo.lpszClassName);
    ListBox2.Items.Add(Format('Small Icon: %d',[ClassInfo.hIconSm]));

end;
```

GetClassLong   Windows.pas

Syntax

GetClassLong(	
hWnd: HWND;	{a handle to a window}
nIndex: Integer	{the offset of the value to retrieve}
): DWORD;	{returns a 32-bit value}

Description

This function returns the 32-bit value at the specified offset into the extra memory for the window class that the given window belongs to. This extra memory is reserved by specifying a value in the ClsExtra member of the TWndClass structure used when the RegisterClass function is called. In addition, this function can return information about the window class by using one of the values in the following table for the Index parameter.

Parameters

hWnd: The handle to the window with the class memory to be accessed.

nIndex: Specifies the zero-based byte offset for the 32-bit value to be retrieved. This can be a value between zero and the number of bytes of extra class memory minus four (i.e., if 16 bytes of extra class memory are allocated, a value of 8 would index into the third 32-bit value). In addition, one of the values in the following table can be used to access specific information about the class.

Return Value

If this function succeeds, it returns the 32-bit value at the specified index into the class memory area; otherwise, it returns a zero. To get extended error information, call the GetLastError function.

See Also

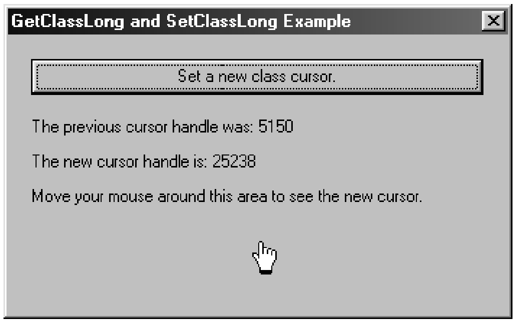
GetClassInfo, GetClassInfoEx, GetClassName, RegisterClass*, RegisterClassEx*, SetClassLong

Example

Listing 3-20: Modifying class settings

This example cycles through the default cursors available through Delphi. Note that the array elements of the Cursors property of the TScreen object are numbered backwards for the standard cursors.

```
var
   CursorIndex: Integer = 0;       // we will start with the first screen cursor

procedure TForm1.Button1Click(Sender: TObject);
var
   HandleCursor: HCURSOR;          // holds the handle to a cursor
begin
   {get a handle to the current cursor for this class}
   HandleCursor:=GetClassLong(Form1.Handle, GCL_HCURSOR);

   {display the cursor handle}
   Label1.Caption:='The previous cursor handle was: '+IntToStr(HandleCursor);

   {set a new cursor for this class from the list of built-in Delphi cursors}
   SetClassLong(Form1.Handle, GCL_HCURSOR, Screen.Cursors[CursorIndex]);

   {display what this new cursor handle is}
   Label2.Caption:='The new cursor handle is: '+IntToStr(Screen.Cursors[CursorIndex]);

   {go to the next cursor in the screen cursor list}
   Dec(CursorIndex);
end;
```



The previous cursor handle was: 5150

The new cursor handle is: 25238

Move your mouse around this area to see the new cursor.

Figure 3-10: A new default cursor

Table 3-3: GetClassLong nIndex values

Values	Description
GCL_CBCLSEXTRA	The size of the extra memory associated with this class, in bytes. Setting this value will not change the amount of memory already allocated.
GCL_CBWNDEXTRA	The size of the extra memory associated with each window of this class, in bytes. Setting this value will not change the amount of memory already allocated.
GCL_HBRBACKGROUND	The handle to the default background brush.

Values	Description
GCL_HCURSOR	The handle to the window class cursor.
GCL_HICON	The handle to the window class icon.
GCL_HICONSM	The handle to the window class small icon.
GCL_HMODULE	The handle of the module that registered the class.
GCL_MENUNAME	A pointer to the menu name string.
GCL_STYLE	The 32-bit style bits for this class.
GCL_WNDPROC	A pointer to the window procedure for this class. If a developer replaces the window procedure using this index, it must conform to the window procedure callback definition as outlined in the RegisterClass function. This subclass will affect all windows subsequently created with this class. An application should not subclass a window created by another process.
GCW_ATOM	An atom that uniquely identifies this class. This is the same atom returned by the RegisterClass and RegisterClassEx functions.

GetClassName Windows.pas

Syntax

```
GetClassName(
hWnd: HWND;              {a handle to a window}
lpClassName: PChar;      {a pointer to a buffer to receive the string}
nMaxCount: Integer       {the size of the buffer in characters}
): Integer;              {returns the number of characters copied}
```

Description

This function simply copies the class name of the specified window to the buffer pointed to by the ClassName parameter.

Parameters

hWnd: A handle to the window to get the class name from.

lpClassName: A pointer to a buffer that will receive the null-terminated class name string.

nMaxCount: Specifies the length of the buffer pointed to by the ClassName parameter. The class name string will be truncated if it is larger than the buffer.

Return Value

If this function succeeds, it returns the length of the class name string in bytes, excluding the null-terminating character; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

GetClassInfo, GetClassInfoEx, GetClassLong, RegisterClass*, SetClassLong

Example

This function is used in a number of examples throughout this chapter. Please see Listing 3-9 under EnumChildWindows and the Window Information application on the companion CD.

GetClientRect Windows.pas

Syntax

```
GetClientRect(
hWnd: HWND;          {a handle to a window}
var lpRect: TRect    {a pointer to a rectangle data structure}
): BOOL;             {returns TRUE or FALSE}
```

Description

This function returns the coordinates of the client area of the given window in the TRect structure pointed to by the Rect variable.

Parameters

hWnd: The handle of the window to get the client coordinates from.

lpRect: A pointer to a TRect structure that will receive the coordinates. These coordinates are in terms of the client area of the specified window. Thus, the Left and Top members will be zero, and the Right and Bottom members will contain the width and height of the client area.

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

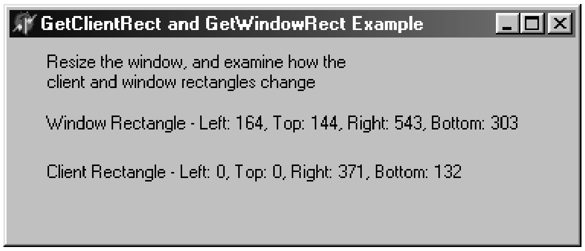
See Also

GetWindowPlacement, GetWindowRect, SetWindowPlacement

Example

Listing 3-21: Displaying the client and window rectangle coordinates

```
procedure TForm1.FormResize(Sender: TObject);
var
   WinRect, ClientRect: TRect;  // these hold the appropriate rectangle coordinates
begin
   {get the window rectangle coordinates}
   Windows.GetWindowRect(Form1.Handle, WinRect);

   {get the client rectangle coordinates}
   Windows.GetClientRect(Form1.Handle, ClientRect);

   {display the coordinates}
   Label1.Caption:=Format('Window Rectangle - Left: %d, Top: %d, Right: %d,
                          Bottom: %d', [WinRect.Left, WinRect.Top,
                          WinRect.Right, WinRect.Bottom]);
   Label2.Caption:=Format('Client Rectangle - Left: %d, Top: %d, Right: %d,
                          Bottom: %d', [ClientRect.Left, ClientRect.Top,
                          ClientRect.Right,ClientRect.Bottom]);
end;
```



Resize the window, and examine how the client and window rectangles change

Window Rectangle - Left: 164, Top: 144, Right: 543, Bottom: 303

Client Rectangle - Left: 0, Top: 0, Right: 371, Bottom: 132

Figure 3-11: Client and window rectangle coordinates

GetDesktopWindow Windows.pas

Syntax

GetDesktopWindow: HWND; {returns a handle to the desktop window}

Description

This function returns a handle to the desktop window. This window encompasses the entire screen and is the area on which all other windows and icons are painted. A developer can pass the handle returned from this function to the GetDC function to get a device context for drawing directly on the desktop surface.

Return Value

If this function succeeds, it returns the handle to the desktop window; otherwise, it returns zero.

See Also

EnumWindows, GetWindow

Example

Listing 3-22: Retrieving a handle to the desktop window

```
procedure TForm1.Button1Click(Sender: TObject);
var
   ClassName: array[0..255] of char;
   DesktopWindow: HWND;
begin
   {get the desktop window handle}
   DesktopWindow := GetDesktopWindow;

   {get the class name of the desktop window}
   GetClassName(DesktopWindow, ClassName, 255);

   {display the class name on the button}
   Button1.Caption := ClassName;
end;
```

GetFocus Windows.pas

Syntax

GetFocus: HWND; {returns a handle to a window}

Description

This function retrieves the handle of a window associated with the calling thread that has the input focus.

Return Value

If this function succeeds, it returns the handle to the window associated with the calling thread that has the input focus. If the function fails, or there is no window associated with the calling thread that has the input focus, it returns zero. If the return value is zero, another thread may have a window with the input focus.

See Also

GetActiveWindow, GetCapture, GetForegroundWindow, GetTopWindow, IsWindowEnabled, SetActiveWindow, SetFocus, WM_KILLFOCUS, WM_SETFOCUS

Example

Listing 3-23: Getting the window with the input focus

This code is put in the OnEnter event of multiple controls.

```
procedure TForm1.Button1Enter(Sender: TObject);
var
   FocusWnd: HWND;                    // this will hold the window handle
   ClassName: array[0..255] of char;  // this will hold the class name
begin
   {get the handle of the window that currently has input focus}
   FocusWnd := GetFocus;

   {get the class name of this window}
   GetClassName(FocusWnd, ClassName, 255);

   {display the class name of the window with input focus}
   Label2.Caption := string(ClassName) + ' has input focus.'
end;
```

GetForegroundWindow Windows.pas

Syntax

GetForegroundWindow: HWND; {returns a handle to a window}

Description

This function returns the handle of the window with which the user is currently working.

Return Value

If the function succeeds, it returns a handle to the foreground window; otherwise, it returns zero.

See Also

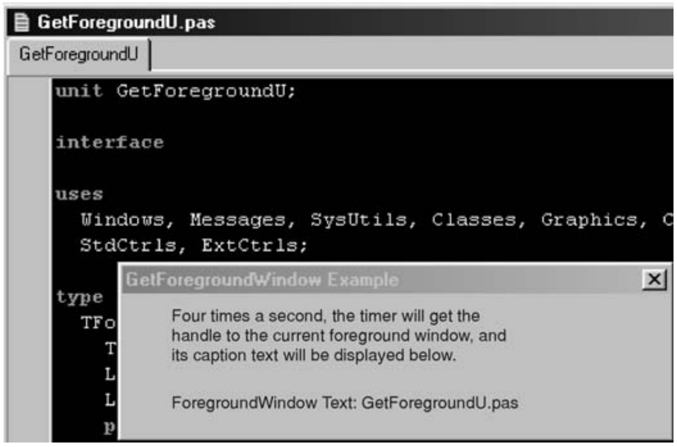
GetFocus, GetTopWindow, SetForegroundWindow

Example

Listing 3-24: Retrieving a handle to the foreground window

The form for this project has its FormStyle property set to fsStayOnTop, so it is visible when other applications have the focus. This code is fired from a timer set at 250 milliseconds.

```
procedure TForm1.Timer1Timer(Sender: TObject);
var
   TheWindowText: array[0..255] of char;
   TheForegroundWindow: HWND;
begin
   {get a handle to the foreground window}
   TheForegroundWindow := GetForegroundWindow;

   {get its caption text}
   GetWindowText(TheForegroundWindow, TheWindowText, 255);

   {display the foreground window's caption}
   Label2.Caption := 'Foreground Window Text: ' + TheWindowText;
end;
```



```
unit GetForegroundU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, C
  StdCtrls, ExtCtrls;

type
  TFo
    T
    L
    L
    p
```

Four times a second, the timer will get the handle to the current foreground window, and its caption text will be displayed below.

ForegroundWindow Text: GetForegroundU.pas

Figure 3-12: The foreground window

GetNextWindow Windows.pas

Syntax

```
GetNextWindow(
hWnd: HWND;          {a handle to the current window}
uCmd: UINT           {direction flags}
): HWND;             {returns a handle to a window}
```

Description

This function returns the handle to the next or previous window in the relative z-order of the specified window. The next window is below the specified window in the z-order, and the previous window is above it. Windows maintains a separate z-order for topmost windows, top-level windows, and child windows, and this function returns a handle to a window relative to the specified window in the appropriate z-order list.

Parameters

hWnd: A handle to the current window.

uCmd: Specifies whether the handle to the next window or previous window, relative to the current window, should be returned. It can be either value from the following table.

Return Value

If this function is successful, it returns the handle to the next or previous window in the relative z-order. If the function fails, or if there is no next or previous window relative to the given window, it returns zero. To get extended error information, call the GetLastError function.

See Also

BringWindowToTop, EnumWindows, FindWindow, FindWindowEx, GetTopWindow, GetWindow

Example

Listing 3-25: Finding the top sibling window and its nearest neighbor in the z-order

```
procedure TForm1.Button1Click(Sender: TObject);
var
   WindowText: array[0..255] of char;
   TheWindow: HWND;
   ThePreviousWindow: HWND;
begin
     {get the handle to the Form1 child window at the
      top of the z-order relative to its siblings}
     TheWindow := GetTopWindow(Form1.Handle);

     {get the text displayed on this window...}
     GetWindowText(TheWindow, WindowText, 255);

     {...and display it in the label}
     Label2.Caption := 'Top Window: ' + WindowText;
```

```
    {now get the window just under it in the z-order}
    ThePreviousWindow := GetNextWindow(TheWindow, GW_HWNDNEXT);

    {get the text displayed on this window...}
    GetWindowText(ThePreviousWindow, WindowText, 255);

    {...and display it in the label}
    Label3.Caption := 'Next To Top: ' + WindowText;
end;
```

Table 3-4: GetNextWindow uCmd values

Value	Description
GW_HWNDNEXT	Returns a handle to the window below the specified window in the relative z-order.
GW_HWNDPREV	Returns a handle to the window above the specified window in the relative z-order.

GetParent Windows.pas

Syntax

GetParent(	
hWnd: HWND	{a handle of a child window}
): HWND;	{returns a handle to a parent window}

Description

This function returns a handle to the given window's parent window, if any.

Parameters

hWnd: A handle to the window whose parent window handle is to be retrieved.

Return Value

If this function succeeds, it returns the handle to the parent window of the given window. If this function fails, or if the specified window does not have a parent window, it returns zero. To get extended error information, call the GetLastError function.

See Also

EnumWindows, FindWindow, FindWindowEx, GetWindow, SetParent

Example

Listing 3-26: Finding a control's parent window

```
procedure TForm1.Button1Click(Sender: TObject);
var
   TheText: array[0..255] of char;
   TheParent: HWND;
begin
   {get the button's parent window handle}
   TheParent:=GetParent(Button1.Handle);

   {get the parent window's text}
```

```
    GetWindowText(TheParent, TheText, 255);

    {display this text on the button}
    Button1.Caption:=TheText;
end;
```

GetProp Windows.pas

Syntax

```
GetProp(
hWnd: HWND;          {a handle to a window}
lpString: PChar      {a pointer to a string}
): THandle;          {returns a 32-bit value}
```

Description

This function retrieves the 32-bit value associated with the given string from the property list of the specified window.

Parameters

hWnd: The handle of the window whose property list is to be searched.

lpString: A pointer to a null-terminated string or an atom identifying a string. If this parameter is an atom, the atom must have been created with a call to GlobalAddAtom. The atom, a 16-bit value, must be in the low-order word and the high-order word must be zero.

Return Value

If this function succeeds and it contains the specified string, it returns the data value associated with that string in the property list of the given window. If the function fails, or the specified string is not in the property list of the window, it returns zero.

See Also

EnumProps, EnumPropsEx, RemoveProp, SetProp

Example

Please see either Listing 3-10 under EnumProps or Listing 3-11 under EnumPropsEx.

GetTopWindow Windows.pas

Syntax

```
GetTopWindow(
hWnd: HWND           {a handle of a parent window}
): HWND;             {returns a handle to a child window}
```

Description

This function examines the child windows of the specified parent window and returns a handle to the child window at the top of the z-order relative to its siblings. Only the siblings of child windows belonging to the parent window are searched. If the child windows have child windows themselves, they are excluded.

Parameters

hWnd: A handle to the parent window whose child windows are to be searched. If this value is zero, this function will return the first child window belonging to the desktop window.

Return Value

If this function succeeds, it returns a handle to the child window at the top of the z-order relative to its siblings. If the function fails, or the parent window has no child windows, it returns zero. To get extended error information, call the GetLastError function.

See Also

BringWindowToTop, EnumChildWindows, EnumWindows, GetActiveWindow, GetForegroundWindow, GetNextWindow, GetWindow, SetActiveWindow, SetForegroundWindow, ShowWindow

Example

Please see Listing 3-25 under GetNextWindow.

GetWindow *Windows.pas*

Syntax

```
GetWindow(
hWnd: HWND;          {a handle to a window}
uCmd: UINT           {relationship flags}
): HWND;             {returns a handle to a window}
```

Description

This function returns a handle to a window that has the specified relationship to the window in the hWnd parameter.

Parameters

hWnd: A handle to a window. The search starts from the window associated with this window handle.

uCmd: Specifies the relationship of the returned window to the specified window and can be one value from the following table.

Return Value

If this function is successful, it returns the handle of the related window. If the function fails, or there is no window with the specified relationship to the given window, it returns zero. To get extended error information, call the GetLastError function.

See Also

GetActiveWindow, GetNextWindow, GetTopWindow, EnumWindows, FindWindow

Example

Listing 3-27: Getting the child window at the top of the z-order

```
const
  GW_ENABLEDPOPUP = 6;                // Delphi does not define this constant

procedure TForm1.Button1Click(Sender: TObject);
var
   TheWindow: HWND;                   // identifies a window
   TheText: array[0..255] of char;    // holds the window text
begin
   {get the child window at the top of the z-order on our main form}
   TheWindow := GetWindow(Form1.Handle, GW_CHILD);

   {get its text...}
   GetWindowText(TheWindow, TheText, 255);

   {...and display it}
   Button1.Caption := TheText;
end;
```

Table 3-5: GetWindow uCmd values

Value	Description
GW_CHILD	Returns a handle to the child window at the top of the z-order if the specified window has child windows; otherwise, the function returns zero.
GW_ENABLEDPOPUP	**Windows 2000 and later**: Returns a handle to the next enabled pop-up window owned by the specified window (found by using GW_HWNDNEXT). If there are no enabled pop-up windows, the function returns the handle of the specified window.
GW_HWNDFIRST	Returns a handle to the window at the top of the z-order of the z-order group containing the specified window (i.e., if the specified window is a child window, the window at the top of the child window z-order is returned; if the specified window is a top-level window, the window at the top of the top-level window z-order is returned).
GW_HWNDLAST	Returns a handle to the window at the bottom of the z-order of the z-order group containing the specified window.
GW_HWNDNEXT	Returns a handle to the window below the specified window in the relative z-order.
GW_HWNDPREV	Returns a handle to the window above the specified window in the relative z-order.
GW_OWNER	Returns a handle to the specified window's owner.

GetWindowLong Windows.pas

Syntax

GetWindowLong(	
hWnd: HWND;	{a handle to a window}
nIndex: Integer	{the offset of the value to retrieve}
): Longint;	{returns a 32-bit value}

Description

This function returns the 32-bit value at the specified offset into the extra window memory for the specified window. This extra memory is reserved by specifying a value in the WndExtra member of the TWndClass structure used when the RegisterClass function is called. In addition, this function can return information about the window by using one of the values in the following table for the Index parameter.

Parameters

hWnd: A handle to the window with the extra window memory to be accessed.

nIndex: Specifies the zero-based byte offset for the value to be retrieved. This can be a value between zero and the number of bytes of extra window memory minus four (i.e., if 16 bytes of extra window memory are allocated, a value of 8 would index into the third 32-bit value). In addition, one of the values in the following table can be used to access specific information about the window.

Return Value

If this function succeeds, it returns the 32-bit value at the specified index into the window memory area; otherwise, it returns zero. To get extended error information, call the GetLastError function.

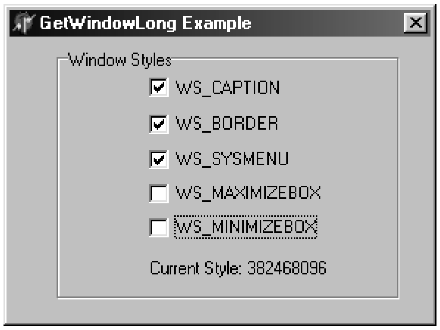
See Also

GetClassInfo, GetClassInfoEx, GetClassName, RegisterClass*, RegisterClassEx*, SetClassLong, SetWindowLong

Example

Listing 3-28: Modifying window styles at run time

```
procedure TForm1.CheckBox1Click(Sender: TObject);
var
   WindowStyle: Longint;   // holds the window style
begin
   {get the current styles used by this window}
   WindowStyle := GetWindowLong(Form1.Handle, GWL_STYLE);

   {toggle the WS_CAPTION style}
   if (CheckBox1.Checked) then
      WindowStyle := WindowStyle OR WS_CAPTION
   else
      WindowStyle := WindowStyle AND NOT WS_CAPTION;
   {toggle the WS_BORDER style}
   if (CheckBox2.Checked) then
      WindowStyle := WindowStyle OR WS_BORDER
   else
      WindowStyle : =WindowStyle AND NOT WS_BORDER;

   {toggle the WS_SYSMENU style}
   if (CheckBox3.Checked) then
      WindowStyle := WindowStyle OR WS_SYSMENU
   else
      WindowStyle : =WindowStyle AND NOT WS_SYSMENU;

   {toggle the WS_MAXIMIZEBOX style}
   if (CheckBox4.Checked) then
      WindowStyle := WindowStyle OR WS_MAXIMIZEBOX
   else
      WindowStyle := WindowStyle AND NOT WS_MAXIMIZEBOX;

   {toggle the WS_MINIMIZEBOX style}
   if (CheckBox5.Checked) then
      WindowStyle := WindowStyle OR WS_MINIMIZEBOX
   else
      WindowStyle := WindowStyle AND NOT WS_MINIMIZEBOX;


   {make the window use the new styles}
   SetWindowLong(Form1.Handle, GWL_STYLE, WindowStyle);

   {this little trick forces the entire window to redraw, including non-client areas}
   SetWindowPos(Handle, 0, 0, 0, 0, 0, SWP_DRAWFRAME or SWP_NOACTIVATE or
                SWP_NOMOVE or SWP_NOSIZE or SWP_NOZORDER);

   {display the current styles used by this window}
   Label1.Caption := 'Current Style: '+IntToStr(WindowStyle);
end;

procedure TForm1.FormCreate(Sender: TObject);
var
   WindowStyle: Longint;   // holds the window style information
begin
   {get the current styles used by this window}
   WindowStyle := GetWindowLong(Form1.Handle, GWL_STYLE);

   {initialize the checkboxes according to the styles that are present}
   if (WindowStyle AND WS_CAPTION) > 0 then CheckBox1.Checked:=TRUE;
   if (WindowStyle AND WS_BORDER) > 0 then CheckBox2.Checked:=TRUE;
   if (WindowStyle AND WS_SYSMENU) > 0 then CheckBox3.Checked:=TRUE;
   if (WindowStyle AND WS_MAXIMIZEBOX) > 0 then CheckBox4.Checked:=TRUE;
   if (WindowStyle AND WS_MINIMIZEBOX) > 0 then CheckBox5.Checked:=TRUE;

   {hook up the OnClick events for the check boxes. this step is necessary
    because the OnClick event is automatically fired when the Checked
    property is accessed.}
   CheckBox1.OnClick := CheckBox1Click;
   CheckBox2.OnClick := CheckBox1Click;
   CheckBox3.OnClick := CheckBox1Click;
   CheckBox4.OnClick := CheckBox1Click;
   CheckBox5.OnClick := CheckBox1Click;
end;
```



Current Style: 382468096

Figure 3-13: The window styles

Table 3-6: GetWindowLong nIndex values

Value	Description
GWL_EXSTYLE	The extended styles used by this window.
GWL_STYLE	The styles used by this window.
GWL_WNDPROC	A pointer to the window procedure for this window. If a developer replaces the window procedure using this index, it must conform to the window procedure callback definition as outlined in the RegisterClass function. The process of replacing a window procedure with a new one is called subclassing. An application should not subclass a window created by another process. A developer must pass any unhandled messages back to the original window procedure. This is accomplished by using the return value from this function with the CallWindowProc function to access the original window procedure.
GWL_HINSTANCE	The handle of the application instance.
GWL_HWNDPARENT	The handle to the parent window, if any.
GWL_ID	The identifier of the window.
GWL_USERDATA	The 32-bit user data value of this window. Every window has a 32-bit user data value that is intended for application-defined data associated with the window.

These values are available if the Wnd parameter specifies a dialog box:

Value	Description
DWL_DLGPROC	A pointer to the dialog box procedure for this dialog box. If a developer replaces the dialog box procedure using this index, it must conform to the dialog box procedure callback function as defined in the CreateDialog function. The process of replacing a dialog box procedure with a new one is called subclassing. An application should not subclass a dialog box created by another process. A developer must pass any unhandled messages back to the original

Value	Description
	window procedure. This is accomplished by using the return value from this function with the CallWindowProc function to access the dialog box procedure.
DWL_MSGRESULT	The return value of a message processed in the dialog box procedure.
DWL_USER	The 32-bit extra dialog box information.

GetWindowRect Windows.pas

Syntax

```
GetWindowRect(
hWnd: HWND;            {a handle of a window}
var lpRect: TRect      {a pointer to a rectangle coordinate structure}
): BOOL;               {returns TRUE or FALSE}
```

Description

This function stores the coordinates of the bounding rectangle for the given window in the structure pointed at by the Rect variable. The coordinates are relative to the upper-left corner of the screen and include the title bar, scroll bars, border, etc., of the specified window.

Parameters

hWnd: A handle to the window whose bounding rectangle is to be retrieved.

lpRect: A pointer to a TRect structure whose members contain the coordinates for the upper-left and lower-right corners of the specified window.

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetWindowPlacement, SetWindowPlacement

Example

Please see Listing 3-21 under GetClientRect.

GetWindowText Windows.pas

Syntax

```
GetWindowText(
hWnd: HWND;            {a handle to a window}
lpString: PChar;       {a pointer to a buffer to receive the string}
nMaxCount: Integer     {the maximum number of characters to copy}
): Integer;            {returns the length of the copied string}
```

Description

This function copies the specified window's title bar text into the given buffer. If the window is a control, the text within the control is copied to the buffer. This function sends a WM_GETTEXT message to the specified window.

Parameters

hWnd: A handle to the window containing the text to be copied to the buffer.

lpString: A pointer to the buffer that will receive the window text.

nMaxCount: Specifies the number of characters to be copied to the buffer. This number includes the terminating null character (i.e., if 21 is specified, 20 characters will be copied to the buffer, and the last character will be set to the null terminator). The window text is truncated if it contains more characters than what is specified in this parameter.

Return Value

If this function succeeds, it returns the length of the copied string, in bytes, excluding the terminating null character. If the function fails, or if there was no text in the specified window, it returns zero. To get extended error information, call the GetLastError function.

See Also

GetWindowTextLength, SetWindowText, WM_GETTEXT

Example

Listing 3-29: Getting and setting the window text

```
procedure TForm1.Button1Click(Sender: TObject);
var
   TheText: PChar;     // this will hold the window text
   TextLen: Integer;   // the length of the window text
begin
   {get the length of the window text}
   TextLen := GetWindowTextLength(Form1.Handle);

   {dynamically allocate space based on the window text length}
   GetMem(TheText, TextLen);

   {get the window text. we must add 1 to account for the terminating null character}
   GetWindowText(Form1.Handle, TheText, TextLen+1);

   {display this text in the edit box}
   Edit1.Text := string(TheText);

   {free the memory for the new string}
   FreeMem(TheText);

end;

procedure TForm1.Button2Click(Sender: TObject);
```

```
begin
   {set the text of the window to the string in the edit box}
   SetWindowText(Form1.Handle, PChar(Edit1.Text));
end;
```

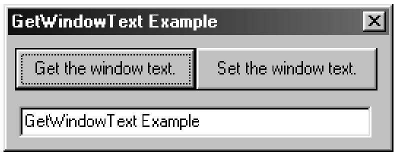


Figure 3-14: The window text has changed

GetWindowTextLength   Windows.pas

Syntax

GetWindowTextLength(
hWnd: HWND   {a handle to a window}
): Integer;   {returns the length of the window text}

Description

This function retrieves the length of the text in the given window's title bar, in bytes. If the window is a control, the length of the text within the control is returned. This function sends a WM_GETTEXTLENGTH message to the given window. It is possible that this function will return a result larger than the actual size of the text when using a mixture of ANSI and Unicode functions within an application.

Parameters

hWnd: A handle to the window from which to extract the text length.

Return Value

If this function succeeds, it returns the length of the text in the given window, in bytes, excluding the terminating null character; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

GetWindowText, SetWindowText, WM_GETTEXT, WM_GETTEXTLENGTH

Example

Please see Listing 3-29 under GetWindowText.

IsChild   Windows.pas

Syntax

IsChild(
hWndParent: HWND;   {a handle to a parent window}
hWnd: HWND   {a handle to the window to test}
): BOOL;   {returns TRUE or FALSE}

Description

This function tests a window to see if it is a child window of the specified parent window. The window is considered a child window if parentage can be traced from the window to the specified parent window.

Parameters

hWndParent: A handle to the parent window.

hWnd: A handle to the child window to be tested.

Return Value

If this function succeeds, and the window in the Wnd parameter is a child window of the window in the WndParent parameter, it returns TRUE. If the function fails, or the given window is not a child window of the specified parent window, it returns FALSE.

See Also

EnumChildWindows, GetParent, SetParent

Example

Listing 3-30: Testing child window status

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   {is this button a child of the main form?}
   if (IsChild(Form1.Handle,Button1.Handle)) then
      Button1.Caption := 'TRUE'
   else
      Button1.Caption := 'FALSE'
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
   {is this button a child of the main form?
    (this button is in a panel, so it is the child of a child window)}
   if (IsChild(Form1.Handle,Button2.Handle)) then
      Button2.Caption := 'TRUE'
   else
      Button2.Caption := 'FALSE'
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
   {is this button a child of the panel? (this button is outside of the panel)}
   if (IsChild(Panel1.Handle,Button3.Handle)) then
      Button3.Caption := 'TRUE'
   else
      Button3.Caption := 'FALSE'
end;
```

IsIconic Windows.pas

Syntax

```
IsIconic(
hWnd: HWND          {a handle to a window}
): BOOL;            {returns TRUE or FALSE}
```

Description

This function tests the specified window to see if it is minimized.

Parameters

hWnd: A handle to the window being tested.

Return Value

If this function succeeds and the specified window is minimized, it returns TRUE. If the function fails, or the specified window is not minimized, it returns FALSE.

See Also

CloseWindow, DestroyWindow*, IsWindowVisible, IsZoomed, OpenIcon, ShowWindow, WM_SIZE

Example

Please see Listing 2-7 under the CloseWindow function.

IsWindow Windows.pas

Syntax

```
IsWindow(
hWnd: HWND          {a potential handle to a window}
): BOOL;            {returns TRUE or FALSE}
```

Description

This function will test the given window handle to determine if it identifies a valid, existing window.

Parameters

hWnd: The window handle being tested.

Return Value

If this function succeeds and the handle identifies an existing window, it returns TRUE. If this function fails, or the given window handle does not identify an existing window, it returns FALSE.

See Also

EnumChildWindows, EnumWindows, FindWindow, FindWindowEx, GetWindow, IsWindowEnabled, IsWindowVisible

Example

Listing 3-31: Testing for a valid window handle

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   {see if the button has a valid window handle}
   if (IsWindow(Button1.Handle)) then
      Button1.Caption := 'TRUE'
   else
      Button1.Caption := 'FALSE';
end;
```

IsWindowEnabled Windows.pas

Syntax

```
IsWindowEnabled(
hWnd: HWND              {a handle to a window to test}
): BOOL;                {returns TRUE or FALSE}
```

Description

This function tests the specified window to see if it is enabled for mouse or keyboard input. A child window can receive input only if it is enabled and visible.

Parameters

hWnd: A handle to the window being tested.

Return Value

If this function succeeds and the specified window is enabled, it returns TRUE. If this function fails, or the specified window is disabled, it returns FALSE.

See Also

EnableWindow, GetActiveWindow, GetFocus, IsWindowVisible, SetActiveWindow, SetFocus, WM_ENABLE

Example

Please see Listing 3-8 under EnableWindow.

IsWindowUnicode Windows.pas

Syntax

```
IsWindowUnicode(
hWnd: HWND              {a handle to a window to test}
): BOOL;                {returns TRUE or FALSE}
```

Description

This function determines if the given window is a native Unicode window.

Parameters

hWnd: A handle to the window being tested.

Return Value

If the function succeeds and the specified window is a native Unicode window, it returns TRUE. If the function fails, or the specified window is not a native Unicode window, it returns FALSE.

See Also

IsWindow

Example

Listing 3-32: Determining if a window is a Unicode window

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   {determine if the window is a Unicode window}
   if (IsWindowUnicode(Form1.Handle)) then
      Button1.Caption := 'This window is a Unicode window'
   else
      Button1.Caption := 'This window is not a Unicode window'
end;
```

IsWindowVisible Windows.pas

Syntax

```
IsWindowVisible(
hWnd: HWND            {a handle to a window to test}
): BOOL;              {returns TRUE or FALSE}
```

Description

This function determines if the specified window has the WS_VISIBLE style flag set. This function will return TRUE as long as the WS_VISIBLE style flag is set, even if the window is completely obscured by other windows or is not visible because it has been clipped by its parent window.

Parameters

hWnd: A handle to the window being tested.

Return Value

If this function succeeds and the specified window has the WS_VISIBLE style flag set, it returns TRUE. If the function fails, or the specified window does not have the WS_VISIBLE style set, it returns FALSE.

See Also

BringWindowToTop, CloseWindow, FindWindow, GetWindowPlacement, SetWindowPlacement, ShowWindow

Example

Listing 3-33: Testing the visibility of a window

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   {test Edit1 for visibility}
   if (IsWindowVisible(Edit1.Handle)) then
      Button1.Caption := 'Edit1 is visible'
   else
      Button1.Caption := 'Edit1 is not visible';
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
   {test Edit2 for visibility}
   if (IsWindowVisible(Edit2.Handle)) then
      Button2.Caption := 'Edit2 is visible'
   else
      Button2.Caption := 'Edit2 is not visible';
end;
```

IsZoomed Windows.pas

Syntax

```
IsZoomed(
hWnd: HWND          {a handle to a window to test}
): BOOL;            {returns TRUE or FALSE}
```

Description

This function tests the specified window to see if it is maximized.

Parameters

hWnd: A handle to the window to be tested.

Return Value

If the function succeeds and the window is maximized, it returns TRUE. If the function fails, or the window is not maximized, it returns FALSE.

See Also

GetWindowPlacement, GetWindowRect, IsIconic, ShowWindow

Example

Listing 3-34: Testing for a maximized state

```
procedure TForm1.FormResize(Sender: TObject);
begin
   {indicate if the window is maximized or not}
   if (IsZoomed(Form1.Handle)) then
      Label1.Caption := 'This window is zoomed'
   else
```

```
    Label1.Caption := 'This window is not zoomed';
end;
```

RemoveProp Windows.pas

Syntax

```
RemoveProp(
hWnd: HWND;         {a handle to a window}
lpString: PChar     {a pointer to a string}
): THandle;         {returns a 32-bit value}
```

Description

This function removes the property associated with the specified string from the property list of the specified window. Before a window is destroyed, the application must remove all properties it has added to that window's property list. An application can only remove properties it has added and should never remove properties added by other applications or by Windows.

Parameters

hWnd: A handle to a window whose property list is to be modified.

lpString: A pointer to a null-terminated string or an atom identifying a string, that identifies the property entry to remove. If this parameter is an atom, the atom must have been created with a call to GlobalAddAtom. The atom, a 16-bit value, must be in the low-order word, and the high-order word must be zero.

Return Value

If the function succeeds, it returns the 32-bit value associated with the specified string. If the function fails, or the string does not exist in the property list, it returns zero.

See Also

EnumProps, EnumPropsEx, GetProp, SetProp

Example

Please see either Listing 3-10 under EnumProps or Listing 3-11 under EnumPropsEx.

SetActiveWindow Windows.pas

Syntax

```
SetActiveWindow(
hWnd: HWND          {a handle to a window to activate}
): HWND;            {returns a handle to the previously active window}
```

Description

This function activates the specified window, giving it input focus. The window will only be brought to the foreground if the thread calling this function owns the specified window. Use the SetForegroundWindow to activate a window and force its associated thread into the foreground.

Parameters

hWnd: A handle to the top-level window to be activated.

Return Value

If this function succeeds, it returns a handle to the previously active window; otherwise, it returns zero.

See Also

GetActiveWindow, SetForegroundWindow, WM_ACTIVATE

Example

Listing 3-35: Toggling active windows

This code is put into the OnTimer event of a timer set to fire every 1000 milliseconds.

```
procedure TForm1.Timer1Timer(Sender: TObject);
var
   ActiveWindow: HWND;  // holds the currently active form handle
begin
   {get the current active form}
   ActiveWindow := GetActiveWindow;

   {toggle the active form}
   if (ActiveWindow = Form1.Handle) then
      SetActiveWindow(Form2.Handle)
   else
      SetActiveWindow(Form1.Handle);
end;

procedure TForm1.FormShow(Sender: TObject);
begin
   {display form2}
   Form2.Show;
end;
```

SetClassLong Windows.pas

Syntax

SetClassLong(	
hWnd: HWND;	{a handle to a window}
nIndex: Integer;	{the index of the value to change}
dwNewLong: Longint	{the new value}
): DWORD;	{returns the previous value at the specified index}

Description

SetClassLong replaces the 32-bit value at the specified offset into the extra memory for the window class associated with the given window. This extra memory is reserved by specifying a value in the ClsExtra member of the TWndClass structure used when the RegisterClass function is called. In addition, it can modify information about the window class by using a value from the following table for the Index parameter.

Parameters

hWnd: The handle to the window with the class memory to be modified.

nIndex: Specifies the zero-based byte offset for the 32-bit value to be set. This can be a value between zero and the number of bytes of extra class memory minus four (i.e., if 16 bytes of extra class memory are allocated, a value of 8 would index into the third 32-bit value). In addition, one of the values in the following table can be used to modify specific information about the class.

dwNewLong: The new 32-bit value to be used at the specified index.

Return Value

If this function succeeds, it returns the previous 32-bit value at the specified offset; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

GetClassInfo, GetClassInfoEx, GetClassLong, GetClassName, RegisterClass*, RegisterClassEx*, SetWindowLong

Example

Please see Listing 3-20 under GetClassLong.

Table 3-7: SetClassLong nIndex values

Values	Description
GCL_CBCLSEXTRA	The size of the extra memory associated with this class, in bytes. Setting this value will not change the amount of memory already allocated.
GCL_CBWNDEXTRA	The size of the extra memory associated with each window of this class, in bytes. Setting this value will not change the amount of memory already allocated.
GCL_HBRBACKGROUND	The handle to the default background brush.
GCL_HCURSOR	The handle to the window class cursor.
GCL_HICON	The handle to the window class icon.
GCL_HICONSM	The handle to the window class small icon.
GCL_HMODULE	The handle of the module that registered the class.
GCL_MENUNAME	A pointer to the menu name string.
GCL_STYLE	The 32-bit style bits for this class.
GCL_WNDPROC	A pointer to the window procedure for this class. If a developer replaces the window procedure using this index, it must conform to the window procedure callback definition as outlined in the RegisterClass function. This subclass will affect all windows subsequently created with this class. An application should not subclass a window created by another process.

SetFocus Windows.pas

Syntax

```
SetFocus(
hWnd: HWND          {a handle to a window}
): HWND;            {returns a handle to the previous focus window}
```

Description

This function gives the specified window the keyboard input focus, activating the window or the parent of the window. It sends a WM_KILLFOCUS message to the window losing the keyboard input focus and a WM_SETFOCUS message to the window receiving the keyboard input focus. If a window is active but no window has the keyboard input focus (the hWnd parameter was set to zero), any keys pressed will send a WM_SYSCHAR, WM_SYSKEYDOWN, or WM_SYSKEYUP message, as appropriate, to the active window's window procedure. In the event that the VK_MENU key is also pressed, the lParam of the messages will have bit 30 set. If the calling thread created the window associated with the window handle in the hWnd parameter, its keyboard focus status is set to this window.

Parameters

hWnd: A handle to the window that will receive keyboard focus. If this parameter is zero, keyboard input is ignored (see above).

Return Value

If this function succeeds, it returns the handle of the window that previously had the keyboard input focus. If the function fails, the hWnd parameter has a handle to an invalid window, or there was no window that previously had keyboard focus, it returns zero.

See Also

GetActiveWindow, GetFocus, SetActiveWindow, SetForegroundWindow, WM_KILLFOCUS, WM_SETFOCUS, WM_SYSCHAR, WM_SYSKEYDOWN, WM_SYSKEYUP

Example

Listing 3-36: Changing the keyboard input focus

Note that this code is placed in the OnTimer event of a timer set to fire every 1000 milliseconds.

```
procedure TForm1.Timer1Timer(Sender: TObject);
var
   HasFocus: HWND;  // identifies a window
begin
   {determine which edit box has the keyboard focus}
   HasFocus := GetFocus;

   {switch focus to the other edit box}
   if (HasFocus = Edit1.Handle) then
```

```
      Windows.SetFocus(Edit2.Handle)
   else
      Windows.SetFocus(Edit1.Handle);
end;
```

SetForegroundWindow Windows.pas

Syntax

SetForegroundWindow(	
hWnd: HWND	{a handle to a window}
): BOOL;	{returns TRUE or FALSE}

Description

This function activates the specified window, brings it to the top of the window z-order, gives it keyboard input focus, and forces the thread that created the window into the foreground. Applications should use this function to force themselves into the foreground.

Parameters

hWnd: A handle to the window to be activated and brought to the foreground.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetForegroundWindow, SetActiveWindow, WM_ACTIVATE

Example

Listing 3-37: Bringing the Windows Explorer into the foreground

```
{note that the Windows Explorer must be running for this example to work}
procedure TForm1.Button1Click(Sender: TObject);
var
   TheWindow: HWND;
begin
   {find a handle to the Windows Explorer window}
   TheWindow := FindWindow('ExploreWClass', nil);

   {bring it into the foreground}
   SetForegroundWindow(TheWindow);
end;
```

SetParent Windows.pas

Syntax

SetParent(	
hWndChild: HWND;	{a handle to a window whose parent is changing}
hWndNewParent: HWND	{a handle to the new parent window}
): HWND;	{returns a handle to the previous parent window}

Description

This function sets the parent window of the hWndChild window to the hWndNew-Parent window. Both windows must belong to the same application. If the child window is visible, Windows performs any necessary redrawing.

Parameters

hWndChild: A handle to a child window.

hWndNewParent: A handle to the new parent window. If this parameter is zero, the desktop window is assumed to be the new parent window.

Return Value

If the function succeeds, it returns a handle to the previous parent window; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

GetParent

Example

Listing 3-38: Changing a button's parent

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   {the parent of button1 is currently the main form. this will set it to panel1, and the
    button will be displayed and clipped by the panel.}
   Windows.SetParent(Button2.Handle, Panel1.Handle);
end;
```

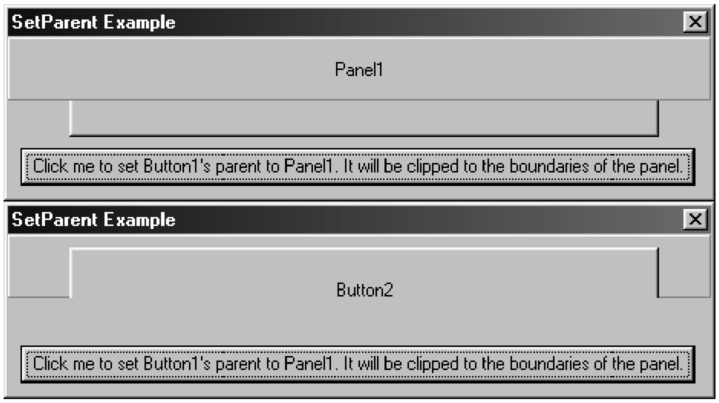

SetParent Example
Panel1
Click me to set Button1's parent to Panel1. It will be clipped to the boundaries of the panel.
SetParent Example
Button2
Click me to set Button1's parent to Panel1. It will be clipped to the boundaries of the panel.

Figure 3-15: The button's parent has changed

SetProp Windows.pas

Syntax

```
SetProp(
hWnd: HWND;          {a handle to a window}
lpString: PChar;     {a pointer to a string}
hData: THandle       {a 32-bit value}
): BOOL;             {returns TRUE or FALSE}
```

Description

This function will add or modify a property list entry of the specified window. If the specified string does not exist in the property list, a new property entry is created. If the string does exist, the data value associated with the specified string is replaced by the new data value. Before a window is destroyed, an application must remove all property entries it has added by using the RemoveProp function.

Parameters

hWnd: A handle to the window whose property list is to be modified.

lpString: A pointer to a null-terminated string or an atom identifying a string. This string will be associated with the data value once it is added to the property list of the window. If this parameter is an atom, the atom must have been created with a call to GlobalAddAtom. The atom, a 16-bit value, must be in the low-order word, and the high-order word must be zero.

hData: A 32-bit value that will be associated with the given string in the property list and can be any value of use to the application.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE.

See Also

EnumProps, EnumPropsEx, GetProp, RemoveProp

Example

Please see either Listing 3-10 under EnumProps or Listing 3-11 under EnumPropsEx.

SetWindowLong Windows.pas

Syntax

```
SetWindowLong(
hWnd: HWND;            {a handle to a window}
nIndex: Integer;       {the index of the value to change}
dwNewLong: Longint     {the new value}
): Longint;            {returns the previous value at the specified index}
```

Description

This function replaces the 32-bit value at the specified offset into the extra memory for the window. This extra memory is reserved by specifying a value in the cbWndExtra member of the TWndClass structure used when the RegisterClass function is called. In addition, this function can modify information about the window by using one of the values in the following table for the nIndex parameter.

Parameters

hWnd: A handle to the window with the extra window memory to be modified.

nIndex: Specifies the zero-based byte offset for the value to be modified. This can be a value between zero and the number of bytes of extra window memory minus four (i.e., if 16 bytes of extra window memory are allocated, a value of 8 would index into the third 32-bit value). In addition, one of the values in the following table can be used to modify specific information about the window.

dwNewLong: The new 32-bit value to be used at the specified index.

Return Value

If the function succeeds, it returns the previous 32-bit value at the specified index; otherwise, it returns zero. To get extended error information, call the GetLastError function.

If the function succeeds and the previous value at the specified index is zero, the return value will be zero. However, the last error information will not be cleared, making it difficult to determine if the function succeeded or failed. Developers should clear the last error information by calling the SetLastError function, passing it a value of 0, before calling the SetWindowLong function. If this is done, SetWindowLong failure will be indicated by a return value of zero and a non-zero return value from GetLastError.

See Also

CallWindowProc*, GetClassLong, GetWindowLong, RegisterClass*, SetClassLong, SetParent

Example

Please see Listing 3-28 under GetWindowLong.

Table 3-8: SetWindowLong nIndex values

Value	Description
GWL_EXSTYLE	The extended styles used by this window.
GWL_STYLE	The styles used by this window.
GWL_WNDPROC	A pointer to the window procedure for this window. If a developer replaces the window procedure using this index, it must conform to the window procedure callback definition as outlined in the RegisterClass function. The process of replacing a window procedure with a new one is called subclassing. An application

Value	Description
	should not subclass a window created by another process. A developer must pass any unhandled messages back to the original window procedure. This is accomplished by using the return value from this function with the CallWindowProc function to access the original window procedure.
GWL_HINSTANCE	The handle of the application instance.
GWL_HWNDPARENT	The handle to the parent window, if any.
GWL_ID	The identifier of the window.
GWL_USERDATA	The 32-bit user data value of this window. Every window has a 32-bit user data value that is intended for application-defined data associated with the window.

These values are available if the Wnd parameter specifies a dialog box:

Value	Description
DWL_DLGPROC	A pointer to the dialog box procedure for this dialog box. If a developer replaces the dialog box procedure using this index, it must conform to the dialog box procedure callback function as defined in the CreateDialog function. The process of replacing a dialog box procedure with a new one is called subclassing. An application should not subclass a dialog box created by another process. A developer must pass any unhandled messages back to the original window procedure. This is accomplished by using the return value from this function with the CallWindowProc function to access the dialog box procedure.
DWL_MSGRESULT	The return value of a message processed in the dialog box procedure.
DWL_USER	The 32-bit extra dialog box information.

SetWindowText Windows.pas

Syntax

```
SetWindowText(
hWnd: HWND;          {a handle to a window}
lpString: PChar      {a pointer to a string}
): BOOL;             {returns TRUE or FALSE}
```

Description

This function changes the text in the title bar of the specified window. If the window is a control, the text in the control is changed. This function sends a WM_SETTEXT message to the specified window. Tab characters are not expanded and will appear as a vertical bar.

Note that if the specified window is a list box control with the WS_CAPTION style specified, this function sets the text for the control, not for the list box entries.

Parameters

hWnd: A handle to a window whose text is to be changed.

lpString: A pointer to a null-terminated string. This string will become the text in the specified window or control.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetWindowText, GetWindowTextLength, WM_SETTEXT

Example

Please see Listing 3-29 under GetWindowText.

WindowFromPoint Windows.pas

Syntax

```
WindowFromPoint(
Point: TPoint      {coordinate information}
): HWND;           {returns a handle to a window}
```

Description

This function returns the handle of the window containing the specified point. This function does not work with hidden or disabled windows.

Parameters

Point: Specifies a TPoint structure containing the coordinates to check. These coordinates are relative to the screen.

Return Value

If this function succeeds, it returns a handle to the window containing the specified point. If it fails, or there is not a window containing the specified point, it returns zero.

See Also

ChildWindowFromPoint, ChildWindowFromPointEx

Example

Listing 3-39: Finding a window at a specific coordinate

```
procedure TForm1.Button1Click(Sender: TObject);
var
   WindowText: array[0..255] of char;  // holds the text of the window
   TheWindow: HWND;                    // holds the window handle
   ThePoint: TPoint;                   // holds the coordinates to check
begin
   {fill in the coordinates}
   ThePoint.X := 5;
   ThePoint.Y := 5;

   {retrieve the window}
   TheWindow := WindowFromPoint(ThePoint);

   {get the window text...}
   GetWindowText(TheWindow, WindowText, 255);

   {...and display it}
   Button1.Caption := WindowText;
end;
```

Chapter 4

File Input/Output Functions

It is rare to create an application that does not do some sort of file input and output, whether that be document creation or just the loading and saving of configuration data. From word processors to web browsers, almost every application deals with the file system in some manner. Fortunately, Windows provides developers with a rich selection of API functions to manage almost any type of file input and output functionality an application might require.

File Creation

Windows handles file creation through the CreateFile function. This function can also open an existing file for modification or simple data retrieval. When a file is opened or created, the system maintains a pointer into the file. This pointer changes location as data is read out of or written into the file. An application can change this file pointer to different locations within the file by using the SetFilePointer function. This allows an application to randomly access information in a file if the file structure is known at run time.

When a file is created or opened, a file handle is returned. This works similar to a window handle, as it identifies the file and allows other functions to manipulate the file. Any process that is started by the current process inherits all open file handles, if they were identified as inheritable. Although Windows will close any open files when the application terminates, information could be lost if an opened file is not explicitly closed by using the CloseHandle function.

Bear in mind that some of the file input/output functions covered in this chapter mention relative and qualified paths. A qualified path consists of the root drive followed by the directory name and each subdirectory name, including the filename (i.e., C:\Program Files\Borland\Delphi\Bin\Delphi32.exe). A relative path uses the relative path markers, such as "." and "..", to point to a directory using the current directory as the origin (i.e., ..\..\Database Desktop\Dbd32.exe).

File Times

Windows records file times in the coordinated universal time format (UTC). UTC is defined as the current date and time in Greenwich, England. Specifically, the file times are stored as a TFileTime structure. The TFileTime structure is defined as:

```
TFileTime = record
    dwLowDateTime: DWORD;      {the low-order 32 bits of the file time}
    dwHighDateTime: DWORD;     {the high-order 32 bits of the file time}
end;
```

The TFileTime structure combines to make a 64-bit value that specifies the number of 100 nanosecond intervals that have elapsed since 12:00 A.M., January 1, 1601 (per Coordinated Universal Time).

The file time stored by the system when the file is written to disk is based on the system time, which is in UTC format. However, an application will usually want to display a file time in local time format. Typically, an application converts a file time to the user's local time zone by calling FileTimeToLocalFileTime and passing the returned TFileTime structure to the FileTimeToSystemTime function. This function returns a data structure with the appropriate values for date and time in the local time zone. This is the method by which the Explorer displays file times in the local time zone format. There are three time values available for a file: creation time, last access time, and last modification time.

Delphi vs. the Windows API

Fortunately, much of the functionality available through these API functions has been wrapped within many various and sundry functions within the Sysutils unit. Doing a search of Sysutils will reveal the use of many functions outlined in this chapter, from opening and closing files to determining a file's age. This makes file input and output much more approachable by the neophyte Delphi programmer. However, as with almost anything that is wrapped into a simpler interface, many of the more advanced techniques or functionalities are simply not available unless the API is used directly. For most uses, the functions in the Sysutils unit will suffice, but when more sophisticated file input and output techniques are required, applications will find more functionality available from the Windows API.

File Input/Output Functions

The following file input/output functions are covered in this chapter:

Table 4-1: File input/output functions

Function	Description
CloseHandle	Closes an open handle.
CompareFileTime	Compares two TFileTime file times.
CopyFile	Copies a file to a new file.
CreateDirectory	Creates a new directory.
CreateDirectoryEx	Creates a directory with the attributes of a specified template directory.
CreateFile	Creates a new file or opens an existing one.
CreateFileMapping	Creates a file mapping object.
DeleteFile	Deletes a file.
DosDateTimeToFileTime	Converts a DOS-based date and time value into the system file time format.
FileTimeToDosDateTime	Converts a system file time value into the DOS date and time format.
FileTimeToLocalFileTime	Converts a UTC-based system file time value into a local file time value.
FileTimeToSystemTime	Converts a file time value into a TSystemTime data structure format.
FindClose	Closes a search handle.
FindCloseChangeNotification	Discontinues monitoring of a change notification handle.
FindFirstChangeNotification	Creates a change notification handle.
FindFirstFile	Searches a directory for a file or directory name.
FindNextChangeNotification	Restores a change notification handle for further monitoring.
FindNextFile	Continues a file search from a previous call to the FindFirstFile function.
FlushFileBuffers	Forces a file to be written to disk.
FlushViewOfFile	Forces a memory-mapped file to be written to disk.
GetCurrentDirectory	Retrieves a path for the current directory.
GetFileAttributes	Retrieves attributes for the specified file.
GetFileInformationByHandle	Retrieves file information from an open file handle.
GetFileSize	Retrieves the specified file's size in bytes.
GetFileTime	Retrieves the specified file's creation, last access, and last write times.
GetFileType	Retrieves the specified file's type.
GetFileVersionInfo	Retrieves the specified file's version information resource.
GetFileVersionInfoSize	Retrieves the size of the specified file's version information resource.
GetFullPathName	Retrieves the full path and long filename of the specified file.
GetShortPathName	Retrieves the short path (8.3 filename format) for the specified file.
GetTempFileName	Creates a temporary filename.
GetTempPath	Retrieves the environment-defined path for temporary file storage.
LocalFileTimeToFileTime	Converts a local file time value to a system UTC-based file time.
LockFile	Locks a portion of a file.

Function	Description
MapViewOfFile	Maps the specified file into the address space of the calling process.
MoveFile	Moves a file from one directory to another.
OpenFileMapping	Opens an existing file mapping object.
ReadFile	Reads information from a file.
RemoveDirectory	Deletes the specified directory.
SearchPath	Searches for a filename on the environment-defined path.
SetCurrentDirectory	Changes directories to the specified directory.
SetEndOfFile	Explicitly sets the end of the specified file.
SetFileAttributes	Sets file attributes.
SetFilePointer	Moves the file pointer within an open file.
SetFileTime	Sets the creation, last access, and last write times of the specified file.
SystemTimeToFileTime	Converts the system time information to the UTC-based system file time.
UnlockFile	Unlocks a previously locked file.
UnmapViewOfFile	Removes a mapped file from the calling process's address space.
VerQueryValue	Retrieves a value from the file's information resource.
WriteFile	Writes information to a file.

CloseHandle Windows.pas

Syntax

```
CloseHandle(
hObject: THandle        {an object handle}
): BOOL;                {returns TRUE or FALSE}
```

Description

The CloseHandle function closes an open device or object and should be used to close handles for console input and output, event files, mapped files, mutexes, named pipes, processes, semaphores, threads, files created from a call to CreateFile, and tokens (Windows NT and later). This function invalidates the specified handle and decrements the handle count of the object associated with the handle by one. Once the object's handle count reaches zero, the object is removed from memory. Attempting to close an invalidated handle will raise an exception

Parameters

hObject: Specifies an open handle.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CreateFile, DeleteFile, FindClose, FindFirstFile

Example

Please see Listing 4-4 under CreateFile.

CompareFileTime Windows.pas

Syntax

CompareFileTime(	
const lpFileTime1: TFileTime;	{a pointer to a TFileTime record}
const lpFileTime2: TFileTime	{a pointer to a TFileTime record}
): Longint;	{returns a file time equality indicator}

Description

This function compares lpFileTime1 to lpFileTime2 and returns the result indicating their difference. This function could be used with GetFileTime to determine if a file was written to when it was last accessed.

Parameters

lpFileTime1: A pointer to a TFileTime structure containing the 64-bit time of the first file to compare.

lpFileTime2: A pointer to a TFileTime structure containing the 64-bit time of the first file to compare.

Return Value

If the function succeeds, it returns a –1, indicating that the first file time is older than the second; a 0, indicating that the file times are equal; or a 1, indicating that the first file time is newer than the second. The function does not indicate an error upon failure.

See Also

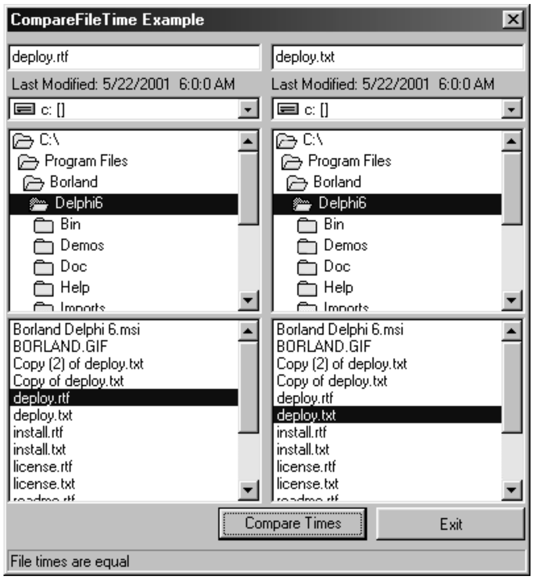
FileTimeToLocalFileTime, FileTimeToSystemTime, GetFileTime

Example

Listing 4-1: Comparing two file times

```
var
  Form1: TForm1;
  File1AccessTime: TFileTime;   // holds the file times to be compared
  File2AccessTime: TFileTime;

implementation

{converts the file time into the proper system time}
procedure TForm1.DisplayTime(TheTime: TFileTime; TheLabel: TLabel);
var
  SystemTime: TSystemTime;    // holds the system time information
  Intermediate: TFileTime;    // holds the local file time
  AMPM: string;               // indicates morning or evening
begin
   {we must first convert the file time into the local file time, and then convert this
    into the system time to get the correct modification time}
   FileTimeToLocalFileTime(TheTime, Intermediate);
   FileTimeToSystemTime(Intermediate, SystemTime);

   {indicate morning or evening, and modify the time so we are
    not displaying military standard}
   if SystemTime.wHour>11 then AMPM := ' PM' else AMPM := ' AM';
   if SystemTime.wHour>12 then SystemTime.wHour := SystemTime.wHour-12;

   {display the time}
   TheLabel.Caption := IntToStr(SystemTime.wMonth) + '/' +
                       IntToStr(SystemTime.wDay) +
                       '/' + IntToStr(SystemTime.wYear) + '  ' +
                       IntToStr(SystemTime.wHour) + ':' +
                       IntToStr(SystemTime.wMinute) + ':' +
                       IntToStr(SystemTime.wSecond) + AMPM;
end;

procedure TForm1.FileListBox1Click(Sender: TObject);
var
 Security: TSecurityAttributes;   // holds file security information
 hFile:THandle;                   // holds a handle to the file
begin
   {initialize the security attributes}
   Security.nLength:=SizeOf(TSecurityAttributes);
   Security.bInheritHandle := FALSE;

   {open the file so we can retrieve a handle to it}
   hFile := CreateFile(PChar(TFileListBox(Sender).FileName), GENERIC_READ,
                       FILE_SHARE_READ, @Security, OPEN_EXISTING,
                       FILE_ATTRIBUTE_NORMAL, 0);
   if hFile = INVALID_HANDLE_VALUE then
   begin
     ShowMessage('Error Opening File');
     Exit;
   end;

   {retrieve the file time and display it}
   if Sender = FileListBox1 then
   begin
     GetFileTime(hFile, nil, nil, @File1AccessTime);
     DisplayTime(File1AccessTime, Label7);
   end
   else
   begin
     GetFileTime(hFile, nil, nil, @File2AccessTime);
     DisplayTime(File2AccessTime, Label2);
   end;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
   lResult: Longint;  // holds the result of the time comparison
begin
   {compare the file times}
   lResult := CompareFileTime(File1AccessTime, File2AccessTime);
```

```
    {display the file comparison result}
    case lResult of
      -1: StatusBar1.SimpleText := 'First file is older than second file';
       0: StatusBar1.SimpleText := 'File times are equal';
       1: StatusBar1.SimpleText := 'First file is younger than second file';
    end;
end;
```



CompareFileTime Example

deploy.rtf | deploy.txt

Last Modified: 5/22/2001 6:0:0 AM | Last Modified: 5/22/2001 6:0:0 AM

```
c: []
C:\
  Program Files
    Borland
      Delphi6
        Bin
        Demos
        Doc
        Help
        Imports

Borland Delphi 6.msi
BORLAND.GIF
Copy (2) of deploy.txt
Copy of deploy.txt
deploy.rtf
deploy.txt
install.rtf
install.txt
license.rtf
license.txt
```

File times are equal

Figure 4-1: The file times are equal

CopyFile Windows.pas

Syntax

CopyFile(	
lpExistingFileName: PChar;	{a pointer to an existing filename}
lpNewFileName: PChar;	{a pointer to a new filename}
bFailIfExists: BOOL	{existing file flags}
): BOOL;	{returns TRUE or FALSE}

Description

This function copies an existing file to a new file. The security attributes of a file are not copied, but the file attributes are copied (i.e., if the existing file is read only, the new file will also be read only).

Parameters

lpExistingFileName: A null-terminated string containing the name of the file to be copied.

lpNewFileName: A null-terminated string containing the name of the new file.

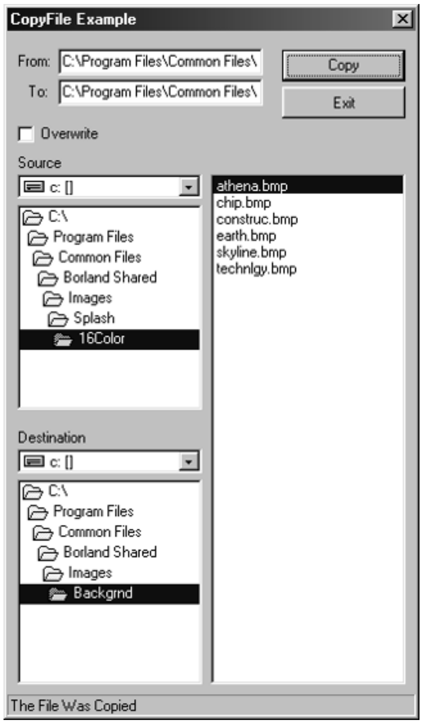
bFailIfExists: Determines how a file is copied if a file exists with the same name as that pointed to by the lpNewFileName parameter. If this parameter is set to TRUE and the new file already exists, the function will fail. If this parameter is set to FALSE, the existing file is overwritten and the function succeeds.

Return Values

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CreateFile, MoveFile



The File Was Copied

Figure 4-2: The file was copied

Example

Listing 4-2: Copying files

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ErrorMessage: Pointer;      // holds a system error string
  ErrorCode: DWORD;           // holds a system error code
begin
  {blank out the status bar}
  StatusBar1.SimpleText := '';

  {attempt to copy the file}
  if not CopyFile(PChar(Edit1.Text+'\'+ExtractFilename(FileListBox1.FileName)),
                  PChar(Edit2.Text+'\'+ExtractFilename(FileListBox1.FileName)),
                  not CheckBox1.Checked) then
  begin
    {if the file was not copied, display the error message}
    ErrorCode := GetLastError;
    FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER or FORMAT_MESSAGE_FROM_SYSTEM,
                  nil, ErrorCode, 0, @ErrorMessage, 0, nil);
    StatusBar1.SimpleText := 'Error Copying File: ' + PChar(ErrorMessage);
    LocalFree(hlocal(ErrorMessage));
  end
  else
    StatusBar1.SimpleText := 'The File Was Copied';
end;
```

CreateDirectory Windows.pas

Syntax

CreateDirectory(	
lpPathName: PChar;	{the new directory path string}
lpSecurityAttributes: PSecurityAttributes	{pointer to directory security attributes}
): BOOL;	{returns TRUE or FALSE}

Description

This function creates a new directory as specified by lpPathName parameter. Under Windows NT and other file systems that support individual file and directory compression, such as NTFS, a new directory inherits the compression attributes of its parent directory.

Parameters

lpPathName: A null-terminated string containing the name of the new directory. This directory name must be less than MAX_PATH characters in size.

lpSecurityAttributes: A pointer to a TSecurityAttributes structure containing information about handle inheritance and file security. This parameter can be set to NIL, indicating that child processes cannot inherit the directory handle. The TSecurityAttributes data structure is defined as:

TSecurityAttributes = record	
nLength: DWORD;	{the size of the TSecurityAttributes structure}
lpSecurityDescriptor: Pointer;	{the security descriptor}
bInheritHandle: BOOL;	{handle inheritance flags}
end;	

The members of this data structure are described under CreateFile.

Note: Under Windows 95/98/Me, the lpSecurityDescriptor member of this structure is ignored.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CreateDirectoryEx, CreateFile, RemoveDirectory

Example

Listing 4-3: Creating a directory

```
procedure TForm1.CreateDirectory1Click(Sender: TObject);
var
  ErrorMessage: Pointer;    // holds a system error message
  ErrorCode: DWORD;         // holds a system error code
```

```
begin
  {determine if a directory path has been specified}
  if DirName.GetTextLen = 0 then
  begin
    StatusBar1.SimpleText := 'Directory name not specified';
    Exit;
  end;

  {if so, then create the new directory under the current directory}
  if not CreateDirectory(PChar(DirectoryListBox1.Directory + '\' +
                         DirName.Text), nil) then
  begin
    {if there was an error creating the directory, display the error message}
    ErrorCode := GetLastError;
    FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER or FORMAT_MESSAGE_FROM_SYSTEM,
                  nil, ErrorCode, 0, @ErrorMessage, 0, nil);
    StatusBar1.SimpleText := 'Error Creating Directory: ' + PChar(ErrorMessage);
    LocalFree(hlocal(ErrorMessage));
  end;

  {update the directory listing to show the new directory}
  DirectoryListBox1.Update;
end;

procedure TForm1.CreateDirectoryFromTemplate1Click(Sender: TObject);
var
  ErrorMessage: Pointer;    // holds a system error message
  ErrorCode: DWORD;         // holds a system error code
begin
  {determine if a directory path has been specified}
  if DirName.GetTextLen = 0 then
  begin
    StatusBar1.SimpleText := 'Directory name not specified';
    Exit;
  end;

  {if so, then create the new directory under the current directory}
  if not CreateDirectoryEx(PChar(Template.Text), PChar(DirectoryListBox1.
                           Directory + '\' + DirName.Text), nil) then
  begin
    {if there was an error creating the directory, display the error message}
    ErrorCode := GetLastError;
    FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER or FORMAT_MESSAGE_FROM_SYSTEM,
                  nil, ErrorCode, 0, @ErrorMessage, 0, nil);
    StatusBar1.SimpleText := 'Error Creating Directory: ' + PChar(ErrorMessage);
    LocalFree(hlocal(ErrorMessage));
  end;

  {reset UI elements}
  Template.Text := '';
  CreateDirectoryFromTemplate1.Enabled := FALSE;
  CreateDirectory1.Enabled := TRUE;
  Template1.Enabled := TRUE;
  ClearTemplate1.Enabled := FALSE;

  {update the directory listing to show the new directory}
  DirectoryListBox1.Update;
end;
```

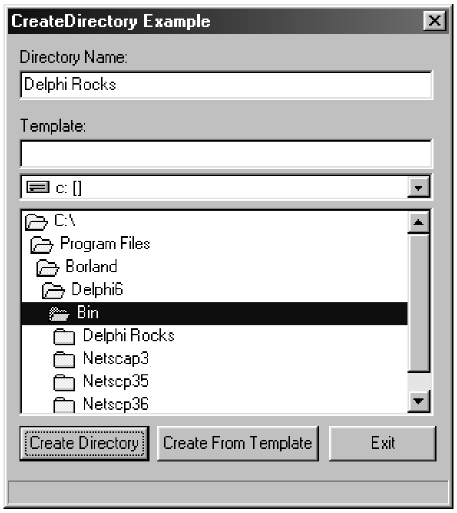


Figure 4-3: A new directory was created

CreateDirectoryEx Windows.pas

Syntax

CreateDirectoryEx(	
lpTemplateDirectory: PChar;	{the directory template string}
lpPathName: PChar;	{the new directory path string}
lpSecurityAttributes: PSecurityAttributes	{pointer to directory security attributes}
): BOOL;	{returns TRUE or FALSE}

Description

This function creates a new directory as specified by the lpPathName parameter that receives the attributes of the template directory specified by the lpTemplateDirectory parameter.

Note: Under Windows NT/2000 and other file systems that support individual file and directory compression, such as NTFS, a new directory inherits the compression attributes of its parent directory.

Parameters

lpTemplateDirectory: A null-terminated string containing the name of an existing directory whose attributes are applied to the new directory being created.

lpPathName: A null-terminated string containing the name of the new directory. This directory name must be less than MAX_PATH characters in size.

lpSecurityAttributes: A pointer to a TSecurityAttributes structure containing information about handle inheritance and file security. This parameter can be set to NIL, indicating that child processes cannot inherit the directory handle. The TSecurityAttributes data structure is defined as:

```
TSecurityAttributes = record
    nLength: DWORD;                   {the size of the TSecurityAttributes structure}
    lpSecurityDescriptor: Pointer;    {the security descriptor}
    bInheritHandle: BOOL;             {handle inheritance flags}
end;
```

The members of this data structure are described under CreateFile.

Note: Under Windows 95/98/Me, the lpSecurityDescriptor member of this structure is ignored.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CreateDirectory, CreateFile, RemoveDirectory

Example

Please see Listing 4-3 under CreateDirectory.

CreateFile Windows.pas

Syntax

```
CreateFile(
lpFileName: PChar;                            {contains the filename to create or open}
dwDesiredAccess: Integer;                     {read/write access flags}
dwShareMode: Integer;                         {file sharing flags}
lpSecurityAttributes: PSecurityAttributes;    {pointer to a TSecurityAttributes structure}
dwCreationDisposition: DWORD;                 {open or creation flags}
dwFlagsAndAttributes: DWORD;                  {file attribute and access flags}
hTemplateFile: THandle;                       {a handle to a template file}
): THandle;                                   {returns a handle to the opened file}
```

Description

This function opens or creates the file specified by the lpFileName parameter. Files can be opened for reading, writing, or both, and they can be created with numerous file attributes and access options. If a file is being created, the function adds the FILE_ATTRIBUTE_ARCHIVE file attribute to those specified by the dwFlagsAnd-Attributes parameter, and the file length is initialized to zero bytes. When the application no longer needs the object, it should close the object's handle by calling the CloseHandle function.

Parameters

lpFileName: A pointer to a null-terminated string containing the name of the file to create or open. This string must not exceed MAX_PATH characters in length.

dwDesiredAccess: Specifies the type of access desired for the file. This parameter may contain one or more values from Table 4-2.

dwShareMode: Specifies how the file is to be shared between applications. If this parameter is set to zero, the file cannot be shared, and any subsequent open operations on the file will fail until the handle is closed. This parameter may contain one or more values from Table 4-3.

lpSecurityAttributes: A pointer to a TSecurityAttributes structure containing information about handle inheritance and file security. This parameter can be set to NIL, indicating that child processes cannot inherit the handle. The TSecurityAttributes data structure is defined as:

```
TSecurityAttributes = record
      nLength: DWORD;                    {the size of the TSecurityAttributes structure}
      lpSecurityDescriptor: Pointer;     {the security descriptor}
      bInheritHandle: BOOL;              {handle inheritance flags}
end;
```

nLength: Specifies the size of the TSecurityAttributes parameter, in bytes. This member should be set to SizeOf(TSecurityAttributes).

lpSecurityDescriptor: A pointer to a security descriptor for the object that controls the sharing of the file. If this member is set to NIL, the file is assigned the default security descriptor for the process. If CreateFile is opening a file, this parameter is ignored.

Note: Under Windows 95/98/Me, this member is always ignored.

bInheritHandle: Indicates if the returned handle is inherited when a new process is created. TRUE indicates that new processes inherit the returned file handle.

dwCreationDisposition: Specifies the function's behavior when a file does or does not exist. This parameter may contain one or more values from Table 4-4.

dwFlagsAndAttributes: Specifies the file attributes and access flags. This parameter may contain one or more values from Table 4-5. If CreateFile is opening a file, this parameter is ignored.

hTemplateFile: Specifies a handle to a file previously opened with the GENERIC_READ flag specified. The file being created gets its file attribute flags from the file specified by this parameter. If CreateFile is opening a file, this parameter is ignored.

Note: Under Windows 95/98/Me, this functionality is not supported and this parameter must be set to zero.

Return Value

If the function succeeds, it returns a handle to the opened or created file. If the function fails, it returns INVALID_HANDLE_VALUE. To get extended error information, call the GetLastError function.

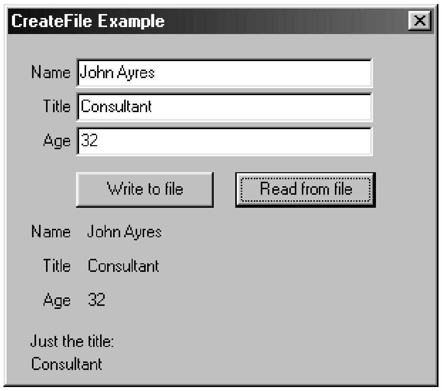
See Also

CloseHandle, CreateDirectory, GetDiskFreeSpaceEx, ReadFile, SetEndOfFile, SetFilePointer, VirtualAlloc*, WriteFile

Example

Listing 4-4: Creating, reading, and writing to a new file

```
{the data structure for our information}
  Information = record
    Name: array[0..255] of char;
    Title: array[0..255] of char;
    Age: Integer;
  end;

var
  Form1: TForm1;

Implementation

const
  {Delphi 6 does not define all available constants}
  FILE_ATTRIBUTE_NOT_CONTENT_INDEXED  = $00002000;
  FILE_ATTRIBUTE_ENCRYPTED            = $00004000;
  FILE_FLAG_OPEN_REPARSE_POINT        = $00200000;
  FILE_FLAG_OPEN_NO_RECALL            = $00100000;

procedure TForm1.Button1Click(Sender: TObject);
var
  FileHandle: THandle;            // a handle to the opened file
  TheInfo: Information;           // holds our information
  NumBytesWritten: DWORD;         // variable to track bytes written
  Security: TSecurityAttributes;  // opened file security attributes
begin
  {copy the supplied information to the data structure}
  StrPCopy(TheInfo.Name, Edit1.Text);
  StrPCopy(TheInfo.Title, Edit2.Text);
  TheInfo.Age := StrToInt(Edit3.Text);

  {create a generic, binary file}
  Security.nLength := SizeOf(TSecurityAttributes);
  Security.bInheritHandle := FALSE;
  FileHandle := CreateFile('TempFile.nfo', GENERIC_WRITE, 0, @Security,
                          CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL or
                          FILE_FLAG_SEQUENTIAL_SCAN, 0);

  {write the data in the data structure directly to the file}
  WriteFile(FileHandle, TheInfo, SizeOf(Information), NumBytesWritten, nil);
  {implicitly set the end of the file.  this could be used to set
   the end of the file to somewhere in the middle}
  SetEndOfFile(FileHandle);

  {force any cached file buffers to write the file to disk}
  FlushFileBuffers(FileHandle);

  {close the file}
  CloseHandle(FileHandle);
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  FileHandle: THandle;            // a handle to the opened file
  TheInfo: Information;           // holds our information
  NumBytesRead: DWORD;            // holds the number of bytes read
  Security: TSecurityAttributes;  // opened file security attributes
  TheTitle: array[0..255] of char; // holds a title string
begin
  {open the existing file for reading}
  Security.nLength := SizeOf(TSecurityAttributes);
  Security.bInheritHandle := FALSE;
  FileHandle := CreateFile('TempFile.nfo', GENERIC_READ, 0, @Security,
                           OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL or
                           FILE_FLAG_SEQUENTIAL_SCAN, 0);
  {indicate an error if the file does not exist}
  if FileHandle=INVALID_HANDLE_VALUE then
  begin
    ShowMessage('No file exists yet.  Press the ''Write to file'' button to '+
                'create a file.');
    Exit;
  end;

  {lock the entire file so no other process may use it}
  LockFile(FileHandle, 0, 0, SizeOf(Information), 0);

  {read in a block of information and store it in our data structure}
  ReadFile(FileHandle, TheInfo, SizeOf(Information), NumBytesRead, nil);

  {display the information}
  Label7.Caption := TheInfo.Name;
  Label8.Caption := TheInfo.Title;
  Label9.Caption := IntToStr(TheInfo.Age);

  {the title is located 256 bytes into the file from the beginning (this is
   how long the Name string is), so reposition the file pointer for reading}
  SetFilePointer(FileHandle, SizeOf(TheInfo.Name), nil, FILE_BEGIN);

  {read one particular string from the file}
  ReadFile(FileHandle, TheTitle, SizeOf(TheTitle), NumBytesRead, nil);

  {display the string}
  Label11.Caption := TheTitle;

  {unlock and close the file}
  UnlockFile(FileHandle, 0, 0, SizeOf(Information), 0);
  CloseHandle(FileHandle);
end;
```



Name John Ayres
Title Consultant
Age 32

Name John Ayres
Title Consultant
Age 32
Just the title:
Consultant

Figure 4-4: The file was created successfully

Table 4-2: CreateFile dwDesiredAccess values

Value	Description
0	Specifies query access to the file.
GENERIC_READ	Specifies read access to the file. Data can be retrieved from the file and the file pointer can be moved.
GENERIC_WRITE	Specifies write access to the file. Data can be written to the file and the file pointer can be moved.

Table 4-3: CreateFile dwShareMode values

Value	Description
FILE_SHARE_DELETE	**Windows NT/2000 and later**: Subsequent open operations on this file will succeed only if they specify the FILE_SHARE_DELETE flag.
FILE_SHARE_READ	Subsequent open operations on this file will succeed only if they specify the FILE_SHARE_READ flag.
FILE_SHARE_WRITE	Subsequent open operations on this file will succeed only if they specify the FILE_SHARE_WRITE flag.

Table 4-4: CreateFile dwCreationDisposition values

Value	Description
CREATE_NEW	Creates a new file. The function will fail if the file already exists.
CREATE_ALWAYS	Always creates a new file, overwriting the file if it already exists.
OPEN_EXISTING	Opens an existing file. The function will fail if the file does not exist.
OPEN_ALWAYS	Always opens the file, creating one if it does not already exist.
TRUNCATE_EXISTING	Opens the specified file and truncates it to a size of zero bytes. The function fails if the file does not exist. The dwDesiredAccess parameter must contain the GENERIC_WRITE flag.

Table 4-5: CreateFile dwFlagsAndAttributes values

Value	Description
FILE_ATTRIBUTE_ARCHIVE	Indicates an archive file or directory and is used by applications to mark files and directories for removal or backup.
FILE_ATTRIBUTE_COMPRESSED	Indicates that the file or directory is compressed.
FILE_ATTRIBUTE_DIRECTORY	Indicates that the specified filename is a directory.
FILE_ATTRIBUTE_ENCRYPTED	**Windows NT/2000/XP and later**: Indicates that the file or directory is encrypted. This flag cannot be used with FILE_ATTRIBUTE_SYSTEM.
FILE_ATTRIBUTE_HIDDEN	Indicates that the specified file or directory is hidden and will not appear in normal directory listings.
FILE_ATTRIBUTE_NORMAL	Indicates that the specified file or directory does not have any other file attributes set.
FILE_ATTRIBUTE_NOT_CONTENT_INDEXED	**Windows NT/2000/XP and later**: Indicates that the file or directory is not to be indexed by content indexing services.
FILE_ATTRIBUTE_OFFLINE	Indicates that the specified file or directory is not immediately available, and it has been physically moved to offline storage.
FILE_ATTRIBUTE_READONLY	Indicates that the specified file or directory is read only. Applications may read from the file or directory, but they may not write to it or delete it.
FILE_ATTRIBUTE_SYSTEM	Indicates that the specified file or directory is used by the system.
FILE_ATTRIBUTE_TEMPORARY	Indicates that the specified file or directory is temporary. The system will not automatically delete temporary files during shutdown.
FILE_FLAG_WRITE_THROUGH	Instructs the operating system to bypass any intermediate cache and write directly to disk.
FILE_FLAG_OVERLAPPED	Performs asynchronous reads and writes on the file. The ReadFile and WriteFile functions may return before the read or write operation has completed. The operating system will not maintain the file pointer when this flag is specified. Note that Windows 95 does not support asynchronous reads or writes to a disk-based file.
FILE_FLAG_NO_BUFFERING	The specified file is opened with no intermediate buffer or cache, which may provide performance increases in some situations. However, the application must conform to specific rules when opening files with this flag: ■ File access must begin at the offsets that are a multiple of the volume's sector size. For example, if the sector size is 512, file access could begin at offset 0, 512, 1024, etc. ■ Bytes can be read into a buffer only in increments equal to the volume's sector size. ■ Buffer addresses for read and write operations must reside in memory at addresses that are a multiple of the volume's sector size.

Value	Description
FILE_FLAG_NO_BUFFERING (cont.)	It is suggested that the developer use the VirtualAlloc function to allocate memory for the buffers, as VirtualAlloc automatically allocates memory at addresses that are a multiple of the volume's sector size. The application can retrieve the size of a volume's sector by calling the GetDiskFreeSpace function.
FILE_FLAG_RANDOM_ACCESS	Indicates that the file will be accessed randomly, allowing the system to optimize file caching for this method of access.
FILE_FLAG_SEQUENTIAL_SCAN	Indicates that the file will be accessed sequentially, allowing the system to optimize file caching for this method of access.
FILE_FLAG_DELETE_ON_CLOSE	Instructs the operating system to immediately delete the file after every open handle for the file is closed. Subsequent open requests for this file will fail unless the FILE_SHARE_DELETE flag is specified.
FILE_FLAG_BACKUP_SEMANTICS	**Windows NT/2000/XP and later**: Indicates that the file is being opened or created for backup or restore purposes.
FILE_FLAG_POSIX_SEMANTICS	Indicates that the file should be accessed using POSIX rules.
FILE_FLAG_OPEN_REPARSE_POINT	**Windows NT/2000/XP and later**: Inhibits the behavior of NTFS reparse points. Cannot be used with the CREATE_ALWAYS flag.
FILE_FLAG_OPEN_NO_RECALL	Indicates that the file data should be retrieved, but the file should continue to reside in remote storage. Used with the Hierarchical Storage Management system.

CreateFileMapping Windows.pas

Syntax

```
CreateFileMapping(
hFile: THandle;                     {a handle to the file being mapped}
lpFileMappingAttributes:
PSecurityAttributes;                {a pointer to a TSecurityAttributes structure}
flProtect: DWORD;                   {mapping object protection flags}
dwMaximumSizeHigh: DWORD;           {high-order double word of maximum size}
dwMaximumSizeLow: DWORD;            {low-order double word of maximum size}
lpName: PChar                       {a pointer to the mapping object name}
): THandle;                         {returns a handle to the file mapping object}
```

Description

This function creates a file mapping object based on the file identified by the hFile parameter. This file mapping object is a direct representation of the file in memory, and any changes to the file in memory affect the file on disk. If the specified size of the memory-mapped file is larger than the file on disk, the disk-based file is increased to the specified size. However, if the size of the file on disk increases beyond the maximum size of the file mapping object, the file mapping object will not contain the extra information in the file. A memory-mapped file can be shared between processes through the use of the OpenFileMapping function. Once a file mapping object is created, the application can obtain access to the file's contents by mapping a view of the file using the MapViewOfFile function. If two processes share the same mapped file object handle, they will see the same data. However, if two processes map the file individually (both of them call the CreateFileMapping function for the same file), changes made to the file by one process will not be seen by the other process. When the application has finished using the mapped file object, close it by calling the UnmapViewOfFile function for every mapped view of the file and CloseHandle for the actual mapped file object handle.

Parameters

hFile: A handle to an open file from which the file mapping object is created. This file must be opened in an access mode compatible with the protection flags specified by the flProtect parameter. This parameter can be set to THandle($FFFFFFFF), which creates a file mapping object of the specified size in the Windows swap file instead of an actual disk-based file. In this case, the dwMaximumSizeHigh and dwMaximumSizeLow parameters must contain values.

lpFileMappingAttributes: A pointer to a TSecurityAttributes structure containing information about handle inheritance and file security. This parameter can be set to NIL, indicating that child processes cannot inherit the handle. The TSecurityAttributes data structure is defined as:

```
TSecurityAttributes = record
      nLength: DWORD;                    {the size of the TSecurityAttributes structure}
      lpSecurityDescriptor: Pointer;     {the security descriptor}
      bInheritHandle: BOOL;              {handle inheritance flags}
end;
```

Please see the CreateFile function for a description of this data structure.

flProtect: Specifies the access protection and attributes of the file mapping object. This parameter may be one value from the following table of protection values (Table 4-6), plus any combination of values from the following table of section attributes (Table 4-7).

dwMaximumSizeHigh: Specifies the high-order double word of the maximum size for the file mapping object.

dwMaximumSizeLow: Specifies the low-order double word of the maximum size for the file mapping object. If this parameter and the dwMaximumSizeHigh parameter are set to zero, the file mapping object will be the same size as the file identified by the hFile parameter.

lpName: A pointer to a null-terminated string containing the name of the file mapping object. This string may contain any character except the backslash (\). If this parameter contains the name of an existing file mapping object, the function requests access to the existing object. This parameter may contain NIL to create an unnamed file mapping object.

Return Value

If the function succeeds, it returns a handle to the file mapping object, either a new one or an existing one if the lpName parameter points to the name of an existing file mapping object. In this case, GetLastError will return ERROR_ALREADY_EXISTS. If the function fails, it returns zero. To get extended error information, call the GetLastError function.

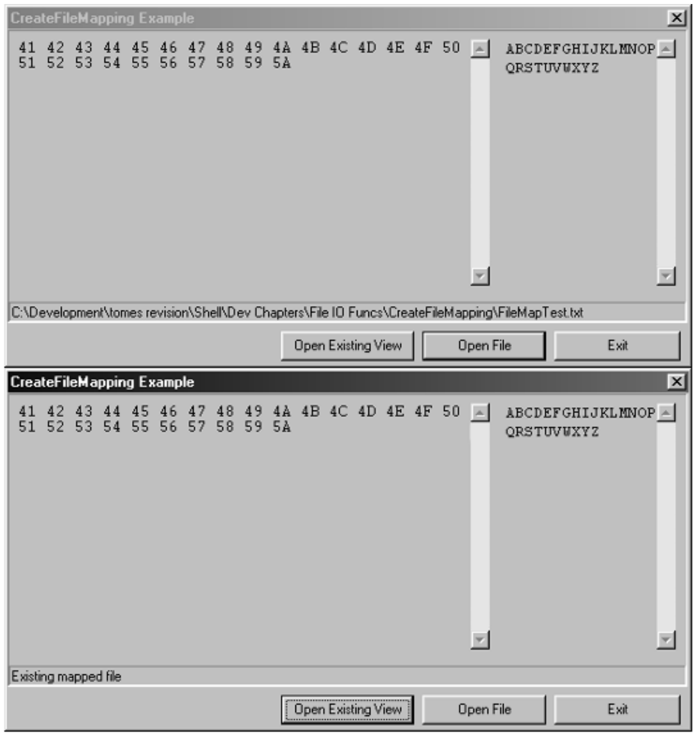
See Also

CloseHandle, FlushViewOfFile, MapViewOfFile, OpenFileMapping, ReadFile, UnMapViewOfFile, VirtualAlloc*, WriteFile

Example

Listing 4-5: Creating a mapped file object

```
var
  Form1: TForm1;
  Data: Pointer;           // holds a pointer to the memory mapped file
  hMapping: THandle;       // holds a handle to the memory mapped file object

implementation

function OpenMappedFile(FileName: string; var FileSize: DWORD): Boolean;
var
  hFile: THandle;       // a handle to the opened file
  HighSize: DWORD;      // the high-order double word of the file size
begin
  {initialize the result of the function in case of errors}
  Result := FALSE;

  {if no filename was specified, exit}
  if Length(FileName) = 0 then Exit;

  {open the file for reading and writing. indicate a
   sequential scan access for better optimization}
  hFile := CreateFile(PChar(FileName), GENERIC_READ or GENERIC_WRITE,
                    FILE_SHARE_READ, nil, OPEN_EXISTING,
                    FILE_FLAG_SEQUENTIAL_SCAN, 0);

  {if the file was not opened, exit;}
  if hFile = INVALID_HANDLE_VALUE then Exit;

  {retrieve the size of the file}
  FileSize := GetFileSize(hFile, @HighSize);

  {create a read/write mapping of the opened file}
  hMapping : =CreateFileMapping(hFile, nil, PAGE_READWRITE, 0, 0,
                              'Delphi File Mapping Example');

  {if the file mapping failed, exit}
  if (hMapping = 0) then
  begin
    CloseHandle(hFile);
    Exit;
  end;

  {close the file handle, as we no longer need it}
  CloseHandle(hFile);
  {map a view of the file}
  Data := MapViewOfFile(hMapping, FILE_MAP_WRITE, 0, 0, 0);

  {if a view of the file was not created, exit}
  if (Data = nil) then Exit;

  {to insure that the file's data can be displayed directly as
   a string, set the very last byte in the file data to a null terminator}
  PChar(Data)[FileSize] := #0;

  {the file was successfully opened and mapped}
  Result := TRUE;
end;

function OpenPreviousMappedFile(var FileSize: Integer): Boolean;
begin
  {initialize the result of the function in case of errors}
  Result := FALSE;

  {open an existing file mapping}
  hMapping := OpenFileMapping(FILE_MAP_WRITE, FALSE, 'Delphi File Mapping Example');

  {if there was an error opening the existing file mapping, exit}
  if hMapping = 0 then Exit;

  {map a view of the file}
  Data := MapViewOfFile(hMapping, FILE_MAP_WRITE, 0, 0, 0);

  {if a view of the file was not created, exit}
  if (Data = nil) then Exit;

  {retrieve the length of the data (which can be represented as a null-terminated string
   due to adding the null terminator to the end when the file was opened}
  FileSize := StrLen(PChar(Data));

  {indicate that the file was successfully opened and mapped}
  Result := TRUE;
end;

procedure DisplayMappedFile(FileName: string; Size: Integer);
var
  Index: Integer;      // general loop counter
  DataPtr: PChar;      // a pointer to the mapped file data
  HexString: PChar;    // a pointer to a concatenated hexadecimal string
begin
  {display the name of the mapped file}
  Form1.StatusBar1.SimpleText := FileName;

  {allocate memory for the hexadecimal representation of the file,
   and initialize it to zeros}
  GetMem(HexString,Size * 3);
  ZeroMemory(HexString, Size * 3);

  {set the pointer to the beginning of the mapped file data}
  DataPtr := Data;
  {begin looping through the data}
  Index := 0;
  while (Index < Size) do
  begin
    {display the value of each byte in the file as a hexadecimal number}
    StrCat(HexString, PChar(IntToHex(Byte(DataPtr[Index]), 2) + ' '));
    Inc(Index);
  end;

  {display the hexadecimal representation of the data and the ASCII
   representation of the data}
  SetWindowText(Form1.Memo1.Handle, HexString);
  SetWindowText(Form1.Memo2.Handle, PChar(Data));

  {free the memory for the hexadecimal string}
  FreeMem(HexString, Size * 3);
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  Size: DWORD;     // holds the size of the memory mapped file
begin
  {open an existing file...}
  if OpenDialog1.Execute then
  begin
    {...map the file...}
    OpenMappedFile(OpenDialog1.FileName, Size);

    {...and display the memory mapped file}
    DisplayMappedFile(OpenDialog1.FileName, Size);
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  Size: Integer;     // holds the size of the memory mapped file
begin
  {open a previously mapped file...}
  if OpenPreviousMappedFile(Size) then
    {...and display it}
    DisplayMappedFile('Existing mapped file', Size);
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  {write any changes to the file}
  FlushViewOfFile(Data, 0);

  {unmap the view of the file}
  if not UnMapViewOfFile(Data) then ShowMessage('Cannot unmap file');

  {close the mapped file handle}
  if not CloseHandle(hMapping) then ShowMessage('Cannot close file');
end;
```


CreateFileMapping Example

```
41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50     ABCDEFGHIJKLMNOP
51 52 53 54 55 56 57 58 59 5A                        QRSTUVWXYZ
```

C:\Development\tomes revision\Shell\Dev Chapters\File IO Funcs\CreateFileMapping\FileMapTest.txt

CreateFileMapping Example

```
41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50     ABCDEFGHIJKLMNOP
51 52 53 54 55 56 57 58 59 5A                        QRSTUVWXYZ
```

Existing mapped file

Figure 4-5: The mapped file

Table 4-6: CreateFileMapping flProtect protection values

Value	Description
PAGE_READONLY	Specifies read-only access to the mapped file memory. Attempting to write to this area will cause an access violation. The file must be opened with the GENERIC_READ flag specified.
PAGE_READWRITE	Specifies read and write access to the mapped file memory. The file must be opened with the GENERIC_READ and GENERIC_WRITE flags specified.
PAGE_WRITECOPY	Specifies that any changes to the file will result in the memory-mapped object containing a copy of the modified data, and the original file will remain unchanged. The file must be opened with the GENERIC_READ and GENERIC_WRITE flags specified.

Table 4-7: CreateFileMapping flProtect section attribute values

Value	Description
SEC_COMMIT	Causes the function to allocate memory for the mapped file from physical storage or the paging file. This is the default behavior.
SEC_IMAGE	**Windows NT/2000/XP and later**: The specified file is an executable image. Mapping information and access protection are retrieved from the file. This flag cannot be combined with any other section attribute flags.

Value	Description
SEC_NOCACHE	The entire memory occupied by the memory-mapped file object is not cached, and file changes are written directly to disk. This flag must be combined with either the SEC_COMMIT or SEC_RESERVE flags.
SEC_RESERVE	Causes the function to reserve memory for the mapped file without allocating physical storage. This reserved memory area cannot be accessed by other memory functions until it is released. The reserved memory area can be committed using the VirtualAlloc function. This flag is only valid when the hFile parameter is set to THandle($FFFFFFFF), indicating a memory-mapped file residing in the Windows swap file.

DeleteFile   Windows.pas

Syntax

```
DeleteFile(
lpFileName: PAnsiChar       {the name of the file to delete}
): BOOL;                    {returns TRUE or FALSE}
```

Description

The DeleteFile function deletes the file indicated by the lpFileName parameter. This function will fail if an application attempts to delete a nonexistent file.

Note: Under Windows NT/2000/XP, the function will fail if the application attempts to delete an opened file or a file that has been memory mapped. Under Windows 95/98/Me, these operations will succeed.

Parameters

lpFileName: A null-terminated string containing the name of the file to delete.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CloseHandle, CreateFile, RemoveDirectory

Example

Please see Listing 4-8 under FindFirstFile.

DosDateTimeToFileTime Windows.pas

Syntax

```
DosDateTimeToFileTime(
wFatDate: WORD;                {a 16-bit DOS date}
wFatTime: WORD;                {a 16-bit DOS time}
var lpFileTime: TFileTime      {a pointer to a TFileTime structure}
): BOOL;                       {returns TRUE or FALSE}
```

Description

This function converts DOS 16-bit date and time values into a 64-bit TFileTime structure.

Parameters

wFatDate: Specifies the 16-bit DOS date value. This is a packed 16-bit value whose bits define the information as described in Table 4-8.

wFatTime: Specifies the 16-bit DOS time value. This is a packed 16-bit value whose bits define the information as described in Table 4-9.

lpFileTime: A pointer to a 64-bit TFileTime structure that receives the Windows-compatible date and time value based on the given DOS date and time values.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

FileTimeToDosDateTime, FileTimeToLocalFileTime, FileTimeToSystemTime, GetFileTime, LocalFileTimeToFileTime, SetFileTime, SystemTimeToFileTime

Example

Please see Listing 4-6 under FileTimeToSystemTime.

Table 4-8: DosDateTimeToFileTime wFatDate values

Bit	Description
0-4	The day of the month (i.e., 1-31).
5-8	The month number (i.e., 1 = January).
9-15	The year offset from 1980 (add this value to 1980 to get the actual year).

Table 4-9: DosDateTimeToFileTime wFatTime values

Bit	Description
0-4	The seconds, divided by two.
5-10	The minute (0-59).
11-15	The hour (0-23, military time).

FileTimeToDosDateTime Windows.pas

Syntax

```
FileTimeToDosDateTime(
const lpFileTime: TFileTime;      {a pointer to a TFileTime structure}
var lpFatDate: WORD;              {a pointer to a buffer receiving the 16-bit DOS date}
var lpFatTime: WORD               {a pointer to a buffer receiving the 16-bit DOS time}
): BOOL;                          {returns TRUE or FALSE}
```

Description

This function converts a 64-bit Windows date and time value into component DOS 16-bit date and time values. FileTimeToDosDateTime can only convert dates in the range of 1/1/1980 to 12/31/2107. The function will fail if the date pointed to by the lpFileTime parameter falls outside of this range.

Parameters

lpFileTime: A pointer to a 64-bit TFileTime structure containing the Windows file time to convert.

lpFatDate: A variable receiving the 16-bit DOS date value. This is a packed 16-bit value whose bits define the information as described in Table 4-10.

lpFatTime: A variable receiving the 16-bit DOS time value. This is a packed 16-bit value whose bits define the information as described in Table 4-11.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

DosDateTimeToFileTime, FileTimeToLocalFileTime, FileTimeToSystemTime, GetFileTime, LocalFileTimeToFileTime, SetFileTime, SystemTimeToFileTime

Example

Please see Listing 4-6 under FileTimeToSystemTime.

Table 4-10: FileTimeToDosDateTime lpFatDate values

Bit	Description
0-4	The day of the month (i.e., 1-31).
5-8	The month number (i.e., 1 = January).
9-15	The year offset from 1980 (add this value to 1980 to get the actual year).

Table 4-11: FileTimeToDosDateTime lpFatTime values

Bit	Description
0-4	The seconds, divided by two.
5-10	The minute (0-59).
11-15	The hour (0-23, military time).

FileTimeToLocalFileTime Windows.pas

Syntax

```
FileTimeToLocalFileTime(
const lpFileTime: TFileTime;       {a pointer to a TFileTime structure}
var lpLocalFileTime: TFileTime     {a pointer to a TFileTime structure}
): BOOL;                           {returns TRUE or FALSE}
```

Description

The FileTimeToLocalFileTime function converts the specified UTC-based time pointed to by the lpFileTime parameter to the local file time.

Parameters

lpFileTime: A pointer to a TFileTime structure holding the 64-bit UTC value to be converted.

lpLocalFileTime: A TFileTime variable that receives the converted local file time.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, use the GetLastError function.

See Also

DosDateTimeToFileTime, FileTimeToDosDateTime, FileTimeToSystemTime, GetFileTime, LocalFileTimeToFileTime, SetFileTime, SystemTimeToFileTime

Example

Please see Listing 4-6 under FileTimeToSystemTime.

FileTimeToSystemTime Windows.pas

Syntax

```
FileTimeToSystemTime(
const lpFileTime: TFileTime;       {a pointer to a TFileTime structure}
var lpSystemTime: TSystemTime      {a pointer to a TSystemTime structure}
): BOOL;                           {returns TRUE or FALSE}
```

Description

This function converts the 64-bit file time pointed to by the lpFileTime parameter into a system time format, which is stored in the TSystemTime structure pointed to by the lpSystemTime parameter.

Parameters

lpFileTime: A pointer to a TFileTime structure holding the 64-bit file time value to convert.

lpSystemTime: A pointer to a TSystemTime structure that receives the converted file time. The TSystemTime data structure is defined as:

```
TSystemTime = record
      wYear: Word;              {the current year}
      wMonth: Word;             {the month number}
      wDayOfWeek: Word;         {the day of the week number}
      wDay: Word;               {the current day of the month}
      wHour: Word;              {the current hour}
      wMinute: Word;            {the current minute}
      wSecond: Word;            {the current second}
      wMilliseconds: Word;      {the current millisecond}
end;
```

wYear: Specifies the current calendar year.

wMonth: Specifies the month number, where 1 = January, 2 = February, etc.

wDayOfWeek: Specifies the day of the week, where 0 = Sunday, 1 = Monday, etc.

wDay: Specifies the day of the month in the range of 1 through 31.

wHour: Specifies the current hour in the range of 0 through 23 (military time).

wMinute: Specifies the current minute in the range of 0 through 59.

wSecond: Specifies the current second in the range of 0 through 59.

wMilliseconds: Specifies the current milliseconds in the range of 0 through 999.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, use the GetLastError function.

See Also

DosDateTimeToFileTime, FileTimeToDosDateTime, FileTimeToLocalFileTime, GetFileTime, LocalFileTimeToFileTime, SetFileTime, SystemTimeToFileTime

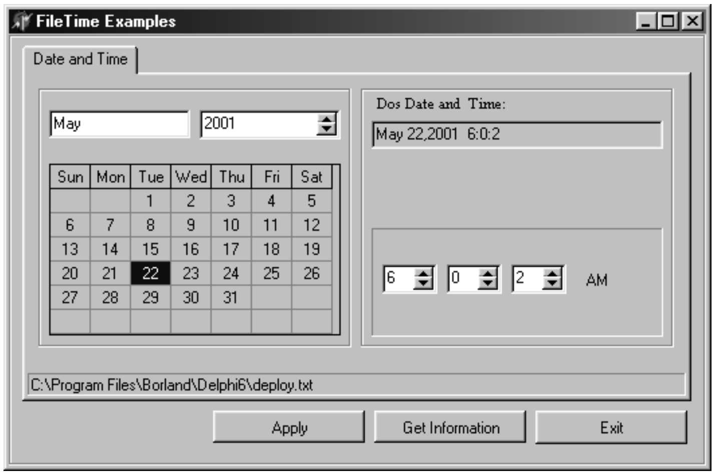
Example

Listing 4-6: Retrieving and setting the file time

```
  {these record structures allow for easy manipulation of DOS date and time values}
  TDosTime = Record
    Hour      : Byte;
    Minutes   : Byte;
    seconds   : Byte;
  end;

  TDosDate = Record
    Year   : Word;
    Month  : Byte;
    Day    : Byte;
  end;

var
  Form1: TForm1;
  hFile: THandle;            // a handle to the opened file
```

```
implementation

{this function provides a convenient way to convert a DOS time into its component parts}
function ConvertDosTimeToSystemTime(FileDosTime: WORD): TDosTime;
var
  DosTime: TDosTime;
begin
  DosTime.Seconds := (FileDosTime and $1F) * 2;
  DosTime.Minutes := (FileDosTime and $7E0) shr 5;
  DosTime.Hour    := (FileDosTime and $F800) shr 11;
  Result          := DosTime;
end;

{this function provides a convenient way to convert a DOS date into its component parts}
function ConvertDosDateToSystemDate(FileDosDate: WORD): TDosDate;
var
  DosDate: TDosDate;
begin
  DosDate.Day   := FileDosDate and $1F;
  DosDate.Month := FileDosDate and $1E0 shr 5;
  DosDate.Year  := (FileDosDate and $FE00) shr 9 + 1980;
  Result        := DosDate;
end;

procedure TForm1.SpeedButton2Click(Sender: TObject);
var
  Security: TSecurityAttributes;       // attributes for the opened file
  FileName: PChar;                     // holds the filename
  WriteTime, LocalTime: TFILETIME;     // holds file times
  DosDate, DosTime: WORD;              // holds the DOS date and time
  infoDosTime: TDosTime;               // holds DOS time information
  infoDosDate: TDosDate;               // holds DOS date information
  SystemTime: TSystemTime;             // holds the last modification time
begin
  {set up the security attributes for the opened file}
  Security.nLength := SizeOf(TSecurityAttributes);
  Security.lpSecurityDescriptor := nil;
  Security.bInheritHandle := FALSE;

  {display the open dialog box}
  if OpenDialog1.Execute then
  begin
    {display the selected filename...}
    FileName := PChar(OpenDialog1.FileName);
    StatusBar1.SimpleText := FileName;

    {...and open it}
    hFile := CreateFile(PChar(FileName),GENERIC_READ or GENERIC_WRITE,
                        FILE_SHARE_READ or FILE_SHARE_WRITE, @Security,
                        OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0);

    {if there was an error, show a message}
    if hFile = INVALID_HANDLE_VALUE then
    begin
      ShowMessage('Error Opening File');
      Exit;
    end;
  end;

  {retrieve the last modification time}
  GetFileTime(hFile, nil, nil, @WriteTime);

  {convert the time to local file time}
  FileTimeToLocalFileTime(WriteTime, LocalTime);

  {finally, convert the time to the system time, so that it
   will match the file time displayed in the Explorer}
  FileTimeToSystemTime(LocalTime, SystemTime);

  {convert the file time into DOS date and time components...}
  FileTimeToDosDateTime(LocalTime, DosDate, DosTime);

  {...and convert it back}
  if not DosDateTimeToFileTime(DosDate, DosTime, LocalTime) then
    ShowMessage ('An error occurred when converting DOS date and time back to'+
                 ' file time.');

  {break out the component parts of the DOS date and time for easy display}
  infoDosTime := ConvertDosTimeToSystemTime(DosTime);
  infoDosDate := ConvertDosDateToSystemDate(DosDate);

  with infoDosTime do
    Edit1.Text := ComboBox1.Items[infoDosDate.Month - 1] + ' ' +
                  IntToStr(infoDosDate.Day) + ',' +
                  IntToStr(infoDosDate.Year) + '  ' +
                  IntToStr(Hour) + ':' +
                  IntToStr(Minutes) + ':' +
                  IntToStr(Seconds);

  {indicate the time of day}
  case SystemTime.WHour of
    12        : Label1.Caption := 'PM';
    13..24    : begin
                  Label1.Caption := 'PM';
                  SystemTime.wHour := SystemTime.wHour - 12;
                end;
    0         : SystemTime.wHour := 12;
  else
    Label1.Caption := 'AM';
  end;

  {display the last modification time of the file}
  SpinEdit1.Value     := SystemTime.wYear;
  SpinEdit2.Value     := SystemTime.wHour;
  SpinEdit3.Value     := SystemTime.wMinute;
  SpinEdit4.Value     := SystemTime.wSecond;
  ComboBox1.ItemIndex := SystemTime.wMonth - 1;
  Calendar1.Month     := SystemTime.wMonth;
  Calendar1.Day       := SystemTime.wDay;
end;
```

```
procedure TForm1.SpeedButton3Click(Sender: TObject);
var
  FileTime, LocalFileTime: TFileTime;  // holds file times
  SystemTime: TSystemTime;             // holds system time information
begin
  {prepare the time information from the values set by the user}
  SystemTime.wHour   := SpinEdit2.Value;

  if (Label1.Caption = 'PM') and (SystemTime.wHour < 12)then
      SystemTime.wHour := SystemTime.wHour + 12;

  SystemTime.wMinute := SpinEdit3.Value;
  SystemTime.wSecond := SpinEdit4.Value;
  SystemTime.wYear   := SpinEdit1.Value;
  SystemTime.wMonth  := ComboBox1.ItemIndex + 1;
  SystemTime.wDay    := Calendar1.Day;

  {convert the system time to a local file time}
  SystemTimeToFileTime(SystemTime, LocalFileTime);

  {convert the local file time to a file time that the file system understands}
  LocalFileTimeToFileTime(LocalFileTime, FileTime);

  {use this time to set the last modification time which shows up in the Explorer}
  SetFileTime(hFile, nil, nil, @FileTime);
end;
```



May 2001

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

May 22,2001 6:0:2

6 0 2 AM

C:\Program Files\Borland\Delphi6\deploy.txt

Figure 4-6: The selected file modification time

FindClose Windows.pas

Syntax

```
FindClose(
hFindFile: THandle      {the search handle}
): BOOL;                {returns TRUE or FALSE}
```

Description

This function closes a search handle as returned by the FindFirstFile and FindNextFile functions.

Parameters

hFile: The file search handle to close.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

FindFirstFile, FindNextFile

Example

Please see Listing 4-8 under FindFirstFile.

FindCloseChangeNotification Windows.pas

Syntax

```
FindCloseChangeNotification(
hChangeHandle: THandle      {a handle to a change notification object}
): BOOL;                    {returns TRUE or FALSE}
```

Description

This function discontinues system monitoring of a file system change notification handle.

Parameters

hChangeHandle: A handle to a file system change notification object as returned by the FindFirstChangeNotification function.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

FindFirstChangeNotification, FindNextChangeNotification

Example

Please see Listing 4-7 under FindFirstChangeNotification.

FindFirstChangeNotification Windows.pas

Syntax

```
FindFirstChangeNotification(
lpPathName: PChar;            {a pointer to the name of the directory to monitor}
bWatchSubtree: BOOL;          {subtree monitor flag}
dwNotifyFilter: DWORD         {change condition flags}
): THandle;                   {returns a handle to a change notification object}
```

Description

This function creates a file system change notification object. It causes the system to monitor the specified directory or subdirectories for specific changes, such as file deletions or name changes. When the conditions specified by the dwNotifyFilter parameter have occurred, the system notifies the returned change notification object. The handle to this object is used with the WaitForSingleObject function, which causes the calling thread to be suspended until the indicated conditions have occured. After the notification, the system can continue monitoring the specified directory by passing the returned handle to the FindNextChangeNotification function. When the notification object is no longer needed, close it by calling the FindCloseChangeNotification function. Ideally, this function would be used in a multithreaded application with threads specifically dedicated to monitoring the change notification object.

Parameters

lpPathName: A pointer to a null-terminated string containing the name of the directory to monitor.

bWatchSubtree: Indicates if the system monitors just the specified directory. If this parameter is set to FALSE, only the specified directory is monitored; a value of TRUE indicates that both the directory and all of its subdirectories are monitored.

dwNotifyFilter: A series of bit flags indicating the conditions under which a change notification will be signaled. This parameter may contain one or more values from Table 4-12.

Return Value

If the function succeeds, it returns a handle to a file system change notification object; otherwise, it returns INVALID_HANDLE_VALUE. To get extended error information, call the GetLastError function.

See Also

FindCloseChangeNotification, FindNextChangeNotification

Example

Listing 4-7: Waiting for a filename change

```
var
  Form1: TForm1;
  NotificationHandle: THandle;    // holds the handle to the notification object

implementation

procedure TForm1.Button2Click(Sender: TObject);
begin
  {establish a notification for filename changes on the selected directory}
  NotificationHandle := FindFirstChangeNotification(PChar(DirectoryListBox1.
                                                 Directory), FALSE,
                                                 FILE_NOTIFY_CHANGE_FILE_NAME);

  {if the notification was set up correctly, modify some UI elements...}
  if (NotificationHandle <> INVALID_HANDLE_VALUE) then
  begin
    Button1.Enabled := TRUE;
    Button2.Enabled := FALSE;
  end
  else
  begin
    {...otherwise indicate that there was an error}
    ShowMessage('There was an error setting the notification');
    Exit;
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  dwResult: DWORD;         // holds the result of waiting on the notification
  Waiting: Boolean;        // loop control variable
begin
  {set up the loop control for a continuous loop}
  Waiting := TRUE;

  {indicate that the application is waiting for the change notification to fire}
  Button1.Enabled := FALSE;
  StatusBar1.SimpleText := 'Now waiting for a filename change';
  Application.ProcessMessages;

  {enter the loop}
  while Waiting do
  begin
    {at this point, the application is suspended until the notification
     object is signaled that a filename change has occurred in the
     selected directory (this includes file deletions)}
    dwResult := WaitForSingleObject(NotificationHandle,INFINITE);
    if (dwResult = WAIT_OBJECT_0) then
    begin
      {indicate that the notification object was signaled}
      ShowMessage('The selected directory signaled a filename change');
      {query the user to see if they wish to continue monitoring this directory}
      if Application.MessageBox('Do you wish to continue monitoring this
                                directory?', 'Continue?', MB_ICONQUESTION or
                                MB_YESNO) = IDYES then
        {if the user wishes to continue monitoring the directory, reset
         the notification object and continue the loop...}
        FindNextChangeNotification(NotificationHandle)
      else
        {...otherwise break out of the loop}
        Waiting := FALSE;
    end;
  end;

  {close the notification object}
  FindCloseChangeNotification(NotificationHandle);

  {reset UI elements}
  Button1.Enabled := FALSE;
  Button2.Enabled := TRUE;
  StatusBar1.SimpleText := '';
  FileListBox1.Update;
end;
```

Table 4-12: FindFirstChangeNotification dwNotifyFilter values

Value	Description
FILE_NOTIFY_CHANGE_ATTRIBUTES	The notification object is signaled when any file or directory attributes change.
FILE_NOTIFY_CHANGE_DIR_NAME	The notification object is signaled when any directory name changes, including deleting or creating a directory.
FILE_NOTIFY_CHANGE_FILE_NAME	The notification object is signaled when any filename change occurs, including renaming, deleting, or creating a file.
FILE_NOTIFY_CHANGE_LAST_WRITE	The notification object is signaled when the last write time of a file or directory is changed. This is detected only when the file is written to disk and may not occur until the file cache is flushed.
FILE_NOTIFY_CHANGE_SECURITY	The notification object is signaled when the security descriptor of any file or directory changes.
FILE_NOTIFY_CHANGE_SIZE	The notification object is signaled when any file in the directory changes size. This is detected only when the file is written to disk and may not occur until the file cache is flushed.

FindFirstFile Windows.pas

Syntax

```
FindFirstFile(
lpFileName: PChar;                  {a pointer to a filename}
var lpFindFileData: TWin32FindData  {a pointer to a TWin32FindData structure}
): THandle;                         {returns a search handle}
```

Description

The FindFirstFile function searches the current directory for the first file that matches the filename specified by the lpFileName parameter. This function will find both files and subdirectories, and the filename being searched for can contain wild cards.

Parameters

lpFileName: A pointer to a null-terminated string containing the path and filename for which to search. This filename may contain wild cards (“*” and “?”).

lpFindFileData: A pointer to a TWin32FindData data structure containing information about the file or subdirectory that was found. The TWin32FindData data structure is defined as:

```
TWin32FindData = record
    dwFileAttributes: DWORD;                     {file attributes}
    ftCreationTime: TFileTime;                   {file creation time}
    ftLastAccessTime: TFileTime;                 {last file access time}
    ftLastWriteTime: TFileTime;                  {last file modification time}
    nFileSizeHigh: DWORD;                        {high double word of file size}
    nFileSizeLow: DWORD;                         {low double word of file size}
    dwReserved0: DWORD;                              {reserved for future use}
    dwReserved1: DWORD;                              {reserved for future use}
    cFileName: array[0..MAX_PATH – 1] of AnsiChar;  {long filename}
    cAlternateFileName: array[0..13] of AnsiChar;    {short filename}
end;
```

dwFileAttributes: Specifies the file attribute flags for the file. See the GetFileAttributes function for a list of possible file attribute flags.

ftCreationTime: Specifies the time that the file was created.

ftLastAccessTime: Specifies the time that the file was last accessed.

ftLastWriteTime: Specifies the time that the file was last modified.

nFileSizeHigh: Specifies the high-order double word of the file size.

nFileSizeLow: Specifies the low-order double word of the file size.

dwReserved0: This member is reserved for future use, and its value is undetermined.

dwReserved1: This member is reserved for future use, and its value is undetermined.

cFileName: A null-terminated string containing the long version of the filename.

cAlternateFileName: A null-terminated string containing the short (8.3) version of the filename.

Return Value

If the function succeeds, it returns a search handle that can be used in subsequent calls to FindNextFile. If the function fails, it returns INVALID_HANDLE_VALUE.

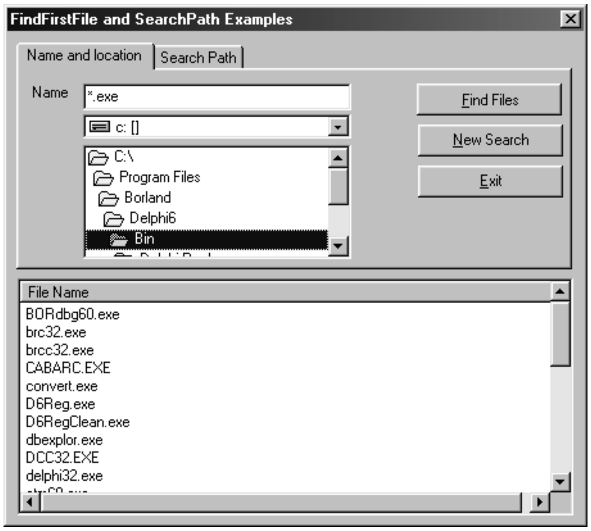
See Also

FindClose, FindNextFile, SearchPath, SetCurrentDirectory

Example

Listing 4-8: Finding files

```
var
  Form1: TForm1;
  ExistingFileName: PChar;   // used in renaming a file

implementation

procedure TForm1.Button1Click(Sender: TObject);
var
  strFileName: string;            // holds the name of the file to find
  FindFileData: TWin32FindData;   // holds file information
  SearchHandle: THandle;          // holds the search handle
begin
  {clear any listed files}
  ListView1.Items.Clear;

  {if there was no file specified, then specify all files}
  if Edit2.GetTextLen = 0 then Edit2.Text := '*.*';

  {construct the filename string}
  strFileName := DirectoryListBox2.Directory + '\' + Edit2.Text;

  {set the directory to the specified directory}
  SetCurrentDirectory(PChar(DirectoryListBox2.Directory));

  {begin the search}
  SearchHandle := FindFirstFile(PChar(strFileName), FindFileData);

  {continue searching for all matching files in the current directory}
  if (SearchHandle <> INVALID_HANDLE_VALUE) then
  repeat
    ListView1.Items.Add.Caption := FindFileData.cFileName;
  Until (FindNextFile(SearchHandle ,FindFileData) = FALSE);

  {all files have been found, so close the search handle}
  Windows.FindClose(SearchHandle);
end;

procedure TForm1.SpeedButton2Click(Sender: TObject);
var
  lpBuffer: PChar;      // receives a path and filename
  lpFilePart: PChar;    // points to the filename
begin
  {clear the listview}
  ListView1.Items.Clear;

  {allocate memory to hold a filename}
  GetMem(lpBuffer, MAX_PATH);
  {if a filename was specified, search for it}
  if Edit1.GetTextLen <> 0 then
  begin
    if (SearchPath(nil, PChar(Edit1.Text), nil, MAX_PATH, lpBuffer,
        lpFilePart) <> 0) then
        {if a file was found, add it to the listview}
        ListView1.Items.Add.Caption := StrPas(lpBuffer)
    else
      MessageBox(0, 'File was not found', 'Error ', MB_OK or MB_ICONWARNING);
  end;

  {free the filename buffer}
  FreeMem(lpBuffer);
end;

procedure TForm1.Delete1Click(Sender: TObject);
begin
  {verify file deletion}
  if (MessageBox (Form1.Handle, 'Are you sure you want to proceed?',
                 'Delete File or Folder', MB_OKCANCEL or
                 MB_ICONQUESTION) = ID_OK) then
  begin
    {delete the file...}
    if (DeleteFile(PChar(ListView1.Selected.Caption)) = FALSE) then
      {...or directory}
      if (RemoveDirectory(PChar(ListView1.Selected.Caption)) = FALSE) then
      begin
        {indicate an error}
        MessageBox(Form1.Handle, 'Error Deleting File or Folder',
                   'Delete File or Folder ', MB_ICONERROR or MB_OK);
        Exit;
      end;

    {delete the value from the list view}
    ListView1.Items.Delete(ListView1.Items.IndexOf(ListView1.Selected));
  end;
end;

procedure TForm1.ListView1Edited(Sender:TObject; Item:TListItem; var S:String);
var
  OldName,NewName: PChar;    // holds the old and new filenames
begin
  {allocate memory for the filename strings}
  GetMem(OldName, MAX_PATH);
  GetMem(NewName, MAX_PATH);

  {retrieve the existing filename}
  ExistingFileName := PChar(ListView1.Selected.Caption);

  {copy the new and old filenames, including path, to the string buffers}
  StrPCopy(NewName, DirectoryListBox2.Directory + '\' + S);
  StrPCopy(OldName, DirectoryListBox2.Directory + '\' + ExistingFileName);

  {rename the file}
  if not MoveFile(OldName, NewName) then
   ShowMessage('Error Renaming file');
  {free the string buffers}
  FreeMem(OldName);
  FreeMem(NewName);
end;
```



BORdbg60.exe
brc32.exe
brcc32.exe
CABARC.EXE
convert.exe
D6Reg.exe
D6RegClean.exe
dbexplor.exe
DCC32.EXE
delphi32.exe

Figure 4-7: Some files were found

FindNextChangeNotification Windows.pas

Syntax

```
FindNextChangeNotification(
hChangeHandle: THandle      {a handle to a change notification object}
): BOOL;                    {returns TRUE or FALSE}
```

Description

This function instructs the system to monitor the specified file system change notification object for another change in its original notification conditions. Calling the FindFirstChangeNotification function creates the notification object. After FindNext-ChangeNotification has reset the change notification object, it can be used with the WaitForSingleObject function to suspend the calling thread until the specified change conditions have occurred.

Parameters

hChangeHandle: A handle to a file system change notification object as returned by a call to the FindFirstChangeNotification function.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended information, call the GetLastError function.

See Also

FindCloseChangeNotification, FindFirstChangeNotification

Example

Please see Listing 4-7 under FindFirstChangeNotification.

FindNextFile Windows.pas

Syntax

```
FindNextFile(
hFindFile: THandle;                    {a file search handle}
var lpFindFileData: TWin32FindData     {a pointer to a TWin32FindData structure}
): BOOL;                               {returns TRUE or FALSE}
```

Description

The FindNextFile function continues to search for a file based on the filename specified by a previous call to the FindFirstFile function.

Parameters

hFindFile: A search handle as returned by a previous call to the FindFirstFile function.

lpFindFileData: A pointer to a TWin32FindData data structure containing information about the file or subdirectory that was found. The TWin32FindData data structure is defined as:

```
TWin32FindData = record
      dwFileAttributes: DWORD;                         {file attributes}
      ftCreationTime: TFileTime;                       {file creation time}
      ftLastAccessTime: TFileTime;                     {last file access time}
      ftLastWriteTime: TFileTime;                      {last file modification time}
      nFileSizeHigh: DWORD;                            {high double word of file size}
      nFileSizeLow: DWORD;                             {low double word of file size}
      dwReserved0: DWORD;                              {reserved for future use}
      dwReserved1: DWORD;                              {reserved for future use}
      cFileName: array[0..MAX_PATH – 1] of AnsiChar;   {long filename}
      cAlternateFileName: array[0..13] of AnsiChar;    {short filename}
end;
```

Please see the FindFirstFile function for a description of this data structure.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

FindClose, FindFirstFile, SearchPath

Example

Please see Listing 4-8 under FindFirstFile.

FlushFileBuffers Windows.pas

Syntax

```
FlushFileBuffers(
hFile:THandle          {a handle to an opened file}
): BOOL;               {returns TRUE or FALSE}
```

Description

This function clears any file buffers for the file associated with the handle in the hFile parameter, causing any buffered data to be immediately written to disk.

Parameters

hFile: A handle to an open file that is to be written to disk. This file must have been opened with the GENERIC_WRITE flag specified.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CreateFile, ReadFile, WriteFile

Example

Please see Listing 4-4 under CreateFile.

FlushViewOfFile Windows.pas

Syntax

```
FlushViewOfFile(
const lpBaseAddress: Pointer;          {base address of mapped file data}
dwNumberOfBytesToFlush: DWORD          {the number of bytes to flush}
): BOOL;                               {returns TRUE or FALSE}
```

Description

This function forces the specified range of bytes within a memory-mapped file to be immediately written to the disk-based representation of the file.

Parameters

lpBaseAddress: A pointer to the base address within the memory-mapped file object data of the range of data to write to disk.

dwNumberOfBytesToFlush: Specifies the number of bytes to write to disk. If this parameter is set to zero, the entire memory-mapped file object is written to disk.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CreateFileMapping, MapViewOfFile, OpenFileMapping, UnmapViewOfFile

Example

Please see Listing 4-5 under CreateFileMapping.

GetCurrentDirectory Windows.pas

Syntax

```
GetCurrentDirectory(
nBufferLength: DWORD;     {the size of lpBuffer in characters}
lpBuffer: PAnsiChar       {a pointer to a buffer receiving the directory name}
): DWORD;                 {returns the number of characters copied to the buffer}
```

Description

This function returns the path of the current directory for the calling process. This directory is stored in the buffer pointed to by the lpBuffer parameter.

Parameters

nBufferLength: Specifies the size of the buffer pointed to by the lpBuffer parameter, in bytes, and must include the null terminator.

lpBuffer: A pointer to a buffer that receives the absolute path for the current directory of the calling process. If this parameter is set to NIL, the return value indicates the required size of the buffer to hold the directory path, including the null terminator.

Return Value

If the function succeeds, it returns the number of characters copied to the lpBuffer buffer, not including the null terminator. If the function fails, it returns zero. To get extended error information, call the GetLastError function.

See Also

CreateDirectory, GetSystemDirectory, GetWindowsDirectory, RemoveDirectory, SetCurrentDirectory

Example

Please see Listing 4-11 under SetFileAttributes.

GetFileAttributes Windows.pas

Syntax

```
GetFileAttributes(
lpFileName: PChar       {the filename whose attributes are retrieved}
): DWORD;               {returns file attribute flags}
```

Description

This function returns the file attributes for the file or directory specified by the lpFileName parameter.

Parameters

lpFileName: A null-terminated string containing the name of the file or directory from which to retrieve file attributes. This string must not be longer than MAX_PATH characters.

Return Value

If the function succeeds, the return value contains one or more of the values from the following table, indicating the current file attributes for the specified file or directory. If the function fails, it returns $FFFFFFFF. To get extended error information, call the GetLastError function.

See Also

FindFirstFile, FindNextFile, SetFileAttributes

Example

Please see Listing 4-11 under SetFileAttributes.

Table 4-13: GetFileAttributes return values

Value	Description
FILE_ATTRIBUTE_ARCHIVE	Indicates an archive file or directory and is used by applications to mark files and directories for removal or backup.
FILE_ATTRIBUTE_COMPRESSED	Indicates that the specified file or directory is compressed.
FILE_ATTRIBUTE_DIRECTORY	Indicates that the specified filename is a directory.
FILE_ATTRIBUTE_ENCRYPTED	**Windows NT/2000/XP and later**: Indicates that the file or directory is encrypted. This flag cannot be used with FILE_ATTRIBUTE_SYSTEM.
FILE_ATTRIBUTE_HIDDEN	Indicates that the specified file or directory is hidden and will not appear in normal directory listings.
FILE_ATTRIBUTE_NORMAL	Indicates that the specified file or directory does not have any other file attributes set.
FILE_ATTRIBUTE_NOT_CONTENT_INDEXED	**Windows NT/2000/XP and later**: Indicates that the file or directory is not to be indexed by content indexing services.
FILE_ATTRIBUTE_OFFLINE	Indicates that the specified file or directory is not immediately available and has been physically moved to offline storage.

Value	Description
FILE_ATTRIBUTE_READONLY	Indicates that the specified file or directory is read only. Applications may read from the file or directory but may not write to it or delete it.
FILE_ATTRIBUTE_REPARSE_POINT	**Windows NT/2000/XP and later**: Indicates that the file has an associated reparse point.
FILE_ATTRIBUTE_SPARSE_FILE	Indicates that this is a sparse file.
FILE_ATTRIBUTE_SYSTEM	Indicates that the specified file or directory is used by the system.
FILE_ATTRIBUTE_TEMPORARY	Indicates that the specified file or directory is temporary. The system will not automatically delete temporary files during shutdown.

GetFileInformationByHandle Windows.pas

Syntax

```
GetFileInformationByHandle(
hFile: THandle;                                {a handle to a file}
var lpFileInformation: TByHandleFileInformation  {a pointer to file information}
): BOOL;                                       {returns TRUE or FALSE}
```

Description

This function retrieves file information for the file associated with the handle identified by the hFile parameter. This handle cannot be the handle to a pipe. GetFileInformationByHandle is affected by the type of file system that the indicated file resides on, and it may return partial information if certain information elements are not supported by the file system.

Parameters

hFile: The handle to the file for which information is retrieved.

lpFileInformation: A pointer to a TByHandleFileInformation data structure receiving information about the indicated file. The TByHandleFileInformation structure is defined as:

```
TByHandleFileInformation = record
    dwFileAttributes: DWORD;         {file attribute flags}
    ftCreationTime: TFileTime;       {file creation time}
    ftLastAccessTime: TFileTime;     {last file access time}
    ftLastWriteTime: TFileTime;      {last file modification time}
    dwVolumeSerialNumber: DWORD;     {volume serial number}
    nFileSizeHigh: DWORD;            {high-order double word of file size}
    nFileSizeLow: DWORD;             {low-order double word of file size}
    nNumberOfLinks: DWORD;           {number of links to the file}
    nFileIndexHigh: DWORD;       {high-order double word of unique identifier}
    nFileIndexLow: DWORD;        {low-order double word of unique identifier}
end;
```

dwFileAttributes: Specifies the file attribute flags for the file. See the GetFileAttributes function for a list of possible file attribute flags.

ftCreationTime: Specifies the time that the file was created.

ftLastAccessTime: Specifies the time that the file was last accessed.

ftLastWriteTime: Specifies the time that the file was last modified.

dwVolumeSerialNumber: Specifies the serial number of the volume that contains the file.

nFileSizeHigh: Specifies the high-order double word of the file size.

nFileSizeLow: Specifies the low-order double word of the file size.

nNumberOfLinks: Specifies the number of links to the file. The FAT system always sets this member to one, but other file systems, such as NTFS, can set this member to a greater value.

nFileIndexHigh: Specifies the high-order double word of the file's unique identifier.

nFileIndexLow: Specifies the low-order double word of the file's unique identifier.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CreateFile, GetFileAttributes, GetFileSize, GetFileTime

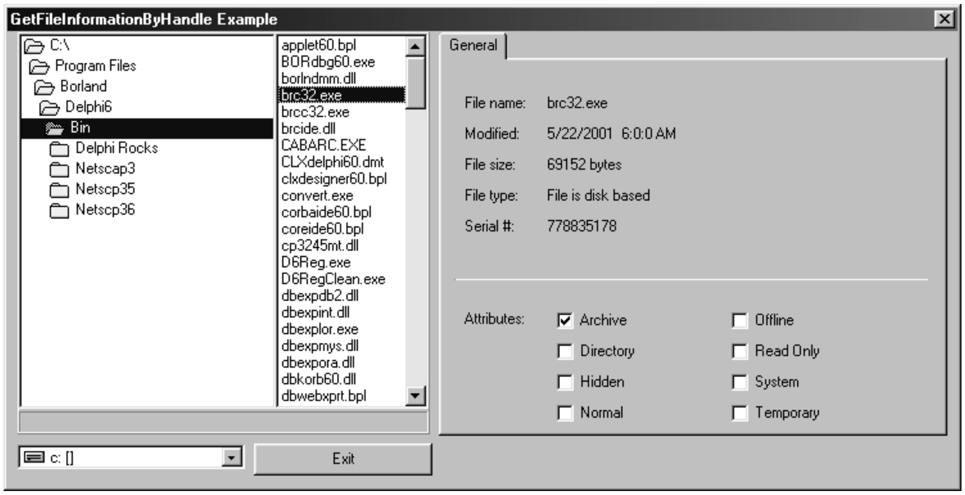
Example

Listing 4-9: Retrieving file information from a handle

```
procedure TForm1.FileListBox1Change(Sender: TObject);
var
  Security: TSecurityAttributes;        // security attributes for the file
  hFile: Integer;                       // holds the file handle
  FileInfo: TByHandleFileInformation;   // holds the file information
  Intermediate: TFileTime;              // holds a file time
  SystemTime: TSystemTime;              // holds the converted file time
  FileType: DWORD;                      // holds the file type
  AMPM: string;                         // morning/evening indicator
begin
  {clear the status bar}
  StatusBar1.SimpleText := '';

  {initialize the security information}
  Security.nLength := SizeOf(TSecurityAttributes);
  Security.bInheritHandle := FALSE;

  {open the selected file for reading}
  hFile := CreateFile(PChar(FileListBox1.FileName), GENERIC_READ, 0, @Security,
                  OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
  if (hFile <> INVALID_HANDLE_VALUE) then
```

```
begin
  {retrieve the file information}
  GetFileInformationByHandle(hFile, FileInfo);

  {display the selected file's attributes}
  checkBox1.Checked := BOOLEAN(FileInfo.dwFileAttributes and FILE_ATTRIBUTE_ARCHIVE);
  CheckBox2.Checked := BOOLEAN(FileInfo.dwFileAttributes and FILE_ATTRIBUTE_DIRECTORY);
  CheckBox3.Checked := BOOLEAN(FileInfo.dwFileAttributes and FILE_ATTRIBUTE_HIDDEN);
  CheckBox4.Checked := BOOLEAN(FileInfo.dwFileAttributes and FILE_ATTRIBUTE_OFFLINE);
  CheckBox5.Checked := BOOLEAN(FileInfo.dwFileAttributes and FILE_ATTRIBUTE_READONLY);
  CheckBox6.Checked := BOOLEAN(FileInfo.dwFileAttributes and FILE_ATTRIBUTE_SYSTEM);
  CheckBox7.Checked := BOOLEAN(FileInfo.dwFileAttributes and FILE_ATTRIBUTE_NORMAL);
  CheckBox8.Checked := BOOLEAN(FileInfo.dwFileAttributes and FILE_ATTRIBUTE_TEMPORARY);

  {display the filename}
  Label1.Caption := ExtractFileName(FileListBox1.FileName);

 {we must first convert the file time into the local file time, and then convert this
  into the system time to get the correct modification time}
 FileTimeToLocalFileTime(FileInfo.ftLastWriteTime, Intermediate);
 FileTimeToSystemTime(Intermediate, SystemTime);

 {indicate morning or evening, and modify the time so we are
  not displaying military standard}
 if SystemTime.wHour > 11 then AMPM := ' PM' else AMPM := ' AM';
 if SystemTime.wHour > 12 then SystemTime.wHour := SystemTime.wHour-12;

 {display the time}
 Label2.Caption := IntToStr(SystemTime.wMonth) + '/' +
                   IntToStr(SystemTime.wDay) +
                   '/' + IntToStr(SystemTime.wYear) + '  ' +
                   IntToStr(SystemTime.wHour) + ':' +
                   IntToStr(SystemTime.wMinute) + ':' +
                   IntToStr(SystemTime.wSecond) + AMPM;

  {display the volume serial number}
  Label8.Caption := IntToStr(FileInfo.dwVolumeSerialNumber);

  {display the file size}
  Label4.Caption := IntToStr(GetFileSize(hFile, nil)) + ' bytes';

  {display the file type}
  FileType := GetFileType(hFile);
  case (FileType) of
    FILE_TYPE_UNKNOWN: Label6.Caption := 'File is of unknown type';
    FILE_TYPE_DISK   : Label6.Caption := 'File is disk based';
    FILE_TYPE_CHAR   : Label6.Caption := 'File is a character file';
    FILE_TYPE_PIPE   : Label6.Caption := 'File is a named or anonymous pipe';
  end;

  {we are through examining the file, so close the handle}
  CloseHandle(hFile);
end

else
  {if the file could not be opened, indicate that it is in use}
```

```
    StatusBar1.SimpleText := 'File is in use';
end;
```


GetFileInformationByHandle Example

```
File name:   brc32.exe
Modified:    5/22/2001  6:0:0 AM
File size:   69152 bytes
File type:   File is disk based
Serial #:    778835178

Attributes:  [✓ Archive]      Offline
             Directory        Read Only
             Hidden           System
             Normal           Temporary
```

Figure 4-8: The file information

GetFileSize Windows.pas

Syntax

GetFileSize(	
hFile: THandle;	{the handle of a file
lpFileSizeHigh: Pointer	{a pointer to the high-order double word of the file size}
): DWORD;	{returns the low-order double word of the file size}

Description

This function returns the size, in bytes, of the file associated with the handle specified by the hFile parameter. This file handle must identify a disk-based file.

Parameters

hFile: A handle to the file from which the size is being retrieved. This file must be opened with the GENERIC_READ and GENERIC_WRITE flags.

lpFileSizeHigh: A pointer to a variable receiving the high-order double word of the file size, if the file is large. If the size of the file being queried will not exceed the capacity of the double word value returned by the function, this parameter can be set to NIL.

Return Value

If the function succeeds, it returns the low-order double word of the file size, and the high-order double word is stored in the variable pointed to by the lpFileSizeHigh parameter. If the function fails, it returns $FFFFFFFF. To get extended error information, call the GetLastError function.

See Also

GetFileInformationByHandle, GetFileType

Example

Please see Listing 4-9 under GetFileInformationByHandle.

GetFileTime Windows.pas

Syntax

```
GetFileTime(
hFile: THandle;                 {a handle to an opened file}
lpCreationTime: PFileTime;      {pointer to buffer receiving the file creation time}
lpLastAccessTime: PFileTime;    {pointer to buffer receiving the last file access time}
lpLastWriteTime: PFileTime      {pointer to buffer receiving the last file write time}
): BOOL;                        {returns TRUE or FALSE}
```

Description

This function retrieves the file creation time, last file access time, and last file write time of the opened file associated with the handle given in the hFile parameter.

Parameters

hFile: A handle to the opened file whose file times are to be retrieved. This file must have been opened with the GENERIC_READ access flag specified.

lpCreationTime: A pointer to a TFileTime data structure to receive the 64-bit file creation time. This parameter may be set to NIL if this time value is not required.

lpLastAccessTime: A pointer to a TFileTime data structure to receive the 64-bit last file access time. This parameter may be set to NIL if this time value is not required.

lpLastWriteTime: A pointer to a TFileTime data structure to receive the 64-bit last file modification time. This parameter may be set to NIL if this time value is not required. This time value is the time value displayed in the Explorer.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

FileTimeToLocalFileTime, FileTimeToSystemTime, SetFileTime

Example

Please see Listing 4-6 under FileTimeToSystemTime.

GetFileType Windows.pas

Syntax

```
GetFileType(
hFile: THandle          {the handle of a file}
): DWORD;               {returns a flag indicating the file type}
```

Description

This function retrieves the type of the file represented by the specified handle.

Parameters

hFile: The handle to the file whose type is being retrieved.

Return Value

If the function succeeds, it returns one value from the following table; otherwise, it returns zero.

See Also

GetFileSize, GetFileTime

Example

Please see Listing 4-9 under GetFileInformationByHandle.

Table 4-14: GetFileType return values

Value	Description
FILE_TYPE_UNKNOWN	The file type is not known.
FILE_TYPE_DISK	The file is a disk-based file.
FILE_TYPE_CHAR	The file is a character stream, such as a console or LPT device.
FILE_TYPE_PIPE	The file is a named or anonymous pipe.

GetFileVersionInfo   Windows.pas

Syntax

```
GetFileVersionInfo(
lptstrFilename: PChar;       {a pointer to a filename}
dwHandle: DWORD;             {this parameter is ignored}
dwLen: DWORD;                {the size of the lpData buffer}
lpData: Pointer              {a pointer to a buffer receiving the version resource}
): BOOL;                     {returns TRUE or FALSE}
```

Description

This function retrieves the file version information resource from the specified file. This function will only succeed on Win32 file images; 16-bit file images are not supported.

Parameters

lptstrFilename: A pointer to a null-terminated string containing the path and filename of the file for which the version information resource is retrieved.

dwHandle: This parameter is completely ignored and may contain any value.

dwLen: Specifies the size of the buffer pointed to by the lpData parameter, in bytes. This parameter should be set to the value returned by the GetFileVersionInfoSize function.

lpData: A pointer to a buffer that receives the file version information resource from the specified file. The pointer to this buffer is used in subsequent calls to VerQueryValue to retrieve individual file version information values.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call GetLastError function.

See Also

GetFileVersionInfoSize, VerQueryValue

Example

Please see Listing 4-12 under VerQueryValue.

GetFileVersionInfoSize Windows.pas

Syntax

```
GetFileVersionInfoSize(
lptstrFilename: PChar;        {a pointer to a filename}
var lpdwHandle: DWORD         {a variable that is set to zero}
): DWORD;                     {returns the size of the version information resource}
```

Description

This function retrieves the size of the specified file's version information resource, in bytes, which is used in a subsequent call to GetFileVersionInfo. This function will only succeed on Win32 file images; 16-bit file images are not supported.

Parameters

lptstrFilename: A pointer to a null-terminated string containing the path and filename of the file for which the size of the version information resource is retrieved.

lpdwHandle: A variable that the function sets to zero.

Return Value

If the function succeeds, it returns the size of the specified file's version information resource, in bytes; otherwise, it returns zero. To get extended error information, call GetLastError function.

See Also

GetFileVersionInfo, VerQueryValue

Example

Please see Listing 4-12 under VerQueryValue.

GetFullPathName Windows.pas

Syntax

```
GetFullPathName(
lpFileName: PAnsiChar;        {the filename}
nBufferLength: DWORD;         {the size of lpBuffer, in characters}
lpBuffer: PAnsiChar;          {a pointer to a buffer receiving the path}
var lpFilePart: PAnsiChar     {a pointer to the filename part inside lpBuffer}
): DWORD;                     {returns the number of characters copied to the buffer}
```

Description

This function returns the full path and filename, including the drive, for the filename identified by the lpFileName parameter. The resulting filename and path is not checked for validity or that it points to an existing file. The returned filename will be in the long filename format.

Parameters

lpFileName: A null-terminated string containing the filename for which to retrieve a full path.

nBufferLength: Specifies the size of the buffer pointed to by the lpBuffer parameter, in characters, and must include the null terminator.

lpBuffer: A pointer to a buffer that receives the full path and filename. If this parameter is set to NIL, the return value indicates the required size of the lpBuffer to hold the full path and filename.

lpFilePart: A pointer to a variable that receives a pointer into the lpBuffer at the beginning of the filename in the full path.

Return Value

If the function succeeds, it returns the number of characters copied to the buffer pointed to by the lpBuffer parameter, including the null terminator. If the function fails, it returns zero. To get extended error information, call the GetLastError function.

See Also

GetShortPathName, GetTempPath, SearchPath

Example

Please see Listing 4-11 under SetFileAttributes.

GetShortPathName Windows.pas

Syntax

```
GetShortPathName(
lpszLongPath: PChar;          {the long path name}
lpszShortPath: PChar;         {a pointer to a buffer that receives the short path name}
cchBuffer: DWORD              {the size of the lpszShortPath buffer, in characters}
): DWORD;                     {returns the number of characters copied to the buffer}
```

Description

This function extracts the short path version of the path specified by the lpszLongPath parameter (i.e., the resulting path is in the 8.3 filename form and contains the "~" character to display long directory names). If the volume that the specified long path name resides on does not support the 8.3 filename format, this function will return ERROR_INVALID_PARAMETER if the specified path is longer than 67 characters.

Parameters

lpszLongPath: A null-terminated string containing the long path from which to extract the short path name. This does not necessarily have to be a fully qualified path.

lpszShortPath: A pointer to a buffer that receives the full path and filename. If this parameter is set to NIL, the return value indicates the required size of lpszShortPath buffer to hold the short path. This buffer can be the same buffer pointed to by the lpszLongPath parameter.

cchBuffer: Specifies the size of the buffer pointed to by the lpszShortPath parameter and must include the null terminator.

Return Value

If the function succeeds, it returns the number of characters copied to the buffer pointed to by the lpszShortPath parameter, not including the null terminator. If the function fails, it returns zero. To get extended error information, call the GetLastError function.

See Also

FindFirstFile, GetFullPathName, GetTempPath, SearchPath

Example

Please see Listing 4-11 under SetFileAttributes.

GetTempFileName Windows.pas

Syntax

```
GetTempFileName(
lpPathName: PChar;         {a pointer to a path}
lpPrefixString: PChar;     {a pointer to the filename prefix string}
uUnique: UINT;             {a unique number used in the filename}
lpTempFileName: PChar      {a pointer to a buffer receiving the temporary filename}
): UINT;                   {returns the unique number used in the filename}
```

Description

This function creates a temporary filename based on the given path, prefix string, and unique number. The filename created always has a .TMP extension. Temporary files created with this function are not automatically deleted when Windows shuts down.

Parameters

lpPathName: A pointer to a null-terminated string containing the path where the temporary file is stored. Typically, this value is the path returned from the GetTempPath function.

lpPrefixString: A pointer to a null-terminated string containing the prefix characters to be used in the filename. The first three letters in the temporary filename are set to the first three letters in the string pointed to by this parameter.

uUnique: An unsigned integer that is converted into a hexadecimal string that follows the prefix characters in the temporary filename. If this parameter is non-zero, the hexadecimal string formed from this parameter is appended to the prefix string obtained from the lpPrefixString parameter to create the temporary filename. However, the file is not created and the function does not test the filename to see if it is unique. If this parameter is set to zero, the function uses a hexadecimal string derived from the current system time. The filename is assembled, and if it is unique, it is created in the target directory. If it is not unique, the hexadecimal number is incremented by one and the filename is tested again. This process continues until a unique filename is found.

lpTempFileName: A pointer to a null-terminated string buffer that receives the created temporary filename string.

Return Value

If the function succeeds, it returns the unique numeric value used in the temporary filename; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

CreateFile, GetTempPath

Example

Listing 4-10: Creating a unique filename

```
var
  Form1: TForm1;
  PathName: array[0..MAX_PATH] of char;  // holds the temporary file path

implementation

procedure TForm1.FormCreate(Sender: TObject);
begin
  {retrieve the path for temporary files}
  GetTempPath(MAX_PATH, @PathName);

  {change the listbox directory to this directory, and display it}
  FileListBox1.Directory := string(PathName);
  Label2.Caption := string(PathName);
end;

procedure TForm1.Button1Click(Sender: TObject);
var
```

```
  NewTempName: array[0..MAX_PATH] of char;  // holds a temporary filename
begin
  {create a temporary filename}
  GetTempFileName(PathName, 'WOW', 0, @NewTempName);

  {display the filename, and update the file listbox}
  Label4.Caption := ExtractFileName(string(NewTempName));
  FileListBox1.Update;
end;
```

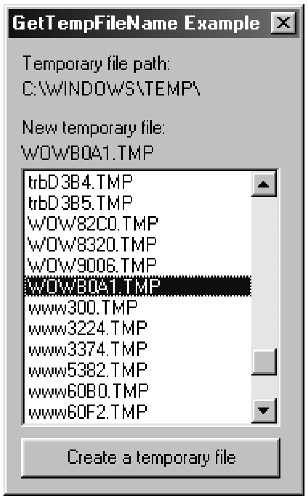


Figure 4-9: The temporary filename was created

GetTempPath Windows.pas

Syntax

GetTempPath(	
nBufferLength: DWORD;	{the size of the lpBuffer buffer}
lpBuffer: PChar	{a pointer to a buffer receiving the temporary file path}
): DWORD;	{returns the number of characters copied to the buffer}

Description

This function retrieves the directory designated for storing temporary files. The directory is retrieved from the TMP environment variable, the TEMP environment variable if TMP is not defined, or the current directory if both the TMP and the TEMP environment variables are not defined.

Parameters

nBufferLength: Specifies the size of the buffer pointed to by the lpBuffer parameter. If this parameter is set to zero, the function returns the size required to store the temporary file path.

lpBuffer: A pointer to a null-terminated string buffer that receives the temporary file path.

Return Value

If this function succeeds, it returns the number of characters copied to the lpBuffer parameter, not including the null terminator character. If the function fails, it returns zero. To get extended error information, call the GetLastError function.

See Also

GetTempFileName

Example

Please see Listing 4-10 under GetTempFileName.

LocalFileTimeToFileTime Windows.pas

Syntax

```
LocalFileTimeToFileTime(
const lpLocalFileTime: TFileTime;      {a pointer to a TFileTime structure}
var lpFileTime: TFileTime              {a pointer to a TFileTime structure}
): BOOL;                               {returns TRUE or FALSE}
```

Description

The LocalFileTimeToFileTime function converts the specified local file time pointed to by the lpLocalFileTime parameter to a UTC-based time value.

Parameters

lpLocalFileTime: A pointer to a TFileTime structure that contains the local file time to be converted.

lpFileTime: A TFileTime variable that receives the converted UTC value.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, use the GetLastError function.

See Also

DosDateTimeToFileTime, FileTimeToDosDateTime, FileTimeToLocalFileTime, FileTimeToSystemTime, GetFileTime, SetFileTime, SystemTimeToFileTime

Example

Please see Listing 4-6 under FileTimeToSystemTime.

LockFile Windows.pas

Syntax

```
LockFile(
hFile: THandle;                    {a handle to an open file}
dwFileOffsetLow: DWORD;            {low-order double word of lock region offset}
dwFileOffsetHigh: DWORD;           {high-order double word of lock region offset}
nNumberOfBytesToLockLow: DWORD;         {low-order double word of lock region
                                         length}
nNumberOfBytesToLockHigh: DWORD         {high-order double word of lock region
                                         length}
): BOOL;                                {returns TRUE or FALSE}
```

Description

This function reserves a region of an open file for exclusive access by the calling process. While the file is locked, no other process will have read or write access to the locked region. Although locked regions may not overlap, it does not cause an error to lock a region that goes beyond the end of the file. A locked region can be unlocked by calling the UnlockFile function. All locked regions on a file should be removed before the file is closed or the application is terminated. This function only succeeds on a FAT-based file system if Share.exe is running.

Parameters

hFile: A handle to the open file which is to be locked. This file must have been created with either the GENERIC_READ or GENERIC_WRITE flags specified.

dwFileOffsetLow: Specifies the low-order word of the offset from the beginning of the file where the locked region should begin.

dwFileOffsetHigh: Specifies the high-order word of the offset from the beginning of the file where the locked region should begin.

nNumberOfBytesToLockLow: Specifies the low-order word of the length, in bytes, of the region to lock.

nNumberOfBytesToLockHigh: Specifies the high-order word of the length, in bytes, of the region to lock.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call GetLastError.

See Also

CreateFile, UnlockFile

Example

Please see Listing 4-4 under CreateFile.

MapViewOfFile Windows.pas

Syntax

```
MapViewOfFile(
hFileMappingObject: THandle;        {a handle to a file mapping object}
dwDesiredAccess: DWORD;             {file view access flags}
dwFileOffsetHigh: DWORD;            {high-order double word of file offset}
dwFileOffsetLow: DWORD;             {low-order double word of file offset}
dwNumberOfBytesToMap: DWORD         {the number of bytes to map}
): Pointer;                         {returns a pointer to the mapped data}
```

Description

This function makes the indicated range of bytes in the memory-mapped file specified by the hFileMappingObject parameter visible and accessible to the application. It returns a pointer to the beginning of the mapped memory, giving the application direct access to the file's data.

Parameters

hFileMappingObject: A handle to an open file mapping object as returned by the CreateFileMapping and OpenFileMapping functions.

dwDesiredAccess: Specifies the desired access to the memory occupied by the mapped file view. This parameter can be one value from the following table.

dwFileOffsetHigh: Specifies the high-order double word of the offset from the beginning of the file from which to start the view mapping.

dwFileOffsetLow: Specifies the low-order double word of the offset from the beginning of the file from which to start the view mapping. The combined 64-bit offset from the beginning of the file must be a multiple of the system's memory allocation granularity. Call the GetSystemInfo function to retrieve memory allocation granularity.

dwNumberOfBytesToMap: Specifies the number of bytes within the file to map. If this parameter is set to zero, the entire file is mapped into a view.

Return Value

If the function succeeds, it returns a pointer to the beginning of the mapped file's view. If the function fails, it returns NIL. To get extended error information, call the GetLastError function.

See Also

CreateFileMapping, GetSystemInfo, OpenFileMapping, UnMapViewOfFile

Example

Please see Listing 4-5 under CreateFileMapping.

Table 4-15: MapViewOfFile dwDesiredAccess values

Value	Description
FILE_MAP_WRITE	Specifies read/write access to the viewed memory range. The file mapping object must have been created with the PAGE_READ-WRITE flag specified.
FILE_MAP_READ	Specifies read only access to the viewed memory range. The file mapping object must have been created with the PAGE_READ-WRITE or PAGE_READONLY flags specified.
FILE_MAP_COPY	Specifies copy on write access to the viewed memory range. Under Windows 95/98/Me, the file mapping object must have been created with the PAGE_WRITECOPY flag specified. When the memory range for a mapped file is modified, the modifications are not written to the original disk file. If this memory-mapped file is shared between processes by using the OpenFileMapping function, any changes to the memory-mapped data will be seen by sharing processes under Windows 95/98/Me but will not be seen by other processes under Windows NT/2000 and later.

MoveFile Windows.pas

Syntax

```
MoveFile(
lpExistingFileName: PAnsiChar;    {the name and path of the existing file}
lpNewFileName: PAnsiChar          {the name and path of the new file}
): BOOL;                          {returns TRUE or FALSE}
```

Description

This function renames the file or directory identified by the lpExistingFileName parameter to the new name identified by the lpNewFileName parameter. If a directory is moved (i.e., renamed), so are its child directories. However, this function will fail if the application attempts to move the directory across volumes.

Parameters

lpExistingFileName: A null-terminated string containing the name and path of the file or directory being renamed.

lpNewFileName: A null-terminated string containing the new name and path for the file or directory. The new file or directory name must not currently exist in the destination.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CopyFile

Example

Please see Listing 4-8 under FindFirstFile.

OpenFileMapping Windows.pas

Syntax

```
OpenFileMapping(
dwDesiredAccess: DWORD;        {memory-mapped file access flags}
bInheritHandle: BOOL;          {handle inheritance flag}
lpName: PChar                  {a pointer to the name of the file mapping object}
): THandle;                    {returns a handle to a file mapping object}
```

Description

This function opens a named file mapping object that currently exists. This can be a file mapping object created by the current process or by another process.

Parameters

dwDesiredAccess: Specifies the desired access to the memory occupied by the mapped file view. This parameter can be one value from the following table.

bInheritHandle: Indicates if the returned handle is inherited when a new process is created. A value of TRUE indicates that new processes inherit the returned file handle.

lpName: A pointer to a null-terminated string containing the name of a file mapping object previously created by the CreateFileMapping function, either within the current process or another process. If a file mapping object by this name is opened and its memory access attributes do not conflict with those specified by the dwDesiredAccess parameter, the function succeeds.

Return Value

If the function succeeds, it returns a handle to the specified file mapping object; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

CreateFileMapping, MapViewOfFile, UnMapViewOfFile

Example

Please see Listing 4-5 under CreateFileMapping.

Table 4-16: OpenFileMapping dwDesiredAccess values

Value	Description
FILE_MAP_WRITE	Specifies read/write access to the viewed memory range. The file mapping object must have been created with the PAGE_READWRITE flag specified.
FILE_MAP_READ	Specifies read-only access to the viewed memory range. The file mapping object must have been created with the PAGE_READWRITE or PAGE_READONLY flags specified.

Value	Description
FILE_MAP_COPY	Specifies copy on write access to the viewed memory range. Under Windows 95/98/Me, the file mapping object must have been created with the PAGE_WRITECOPY flag specified. When the memory range for a mapped file is modified, the modifications are not written to the original disk file. If this memory-mapped file is shared between processes by using the OpenFileMapping function, any changes to the memory-mapped data will be seen by sharing processes under Windows 95/98/Me but will not be seen by other processes under Windows NT/2000 and later.

ReadFile Windows.pas

Syntax

```
ReadFile(
hFile: THandle;                     {a handle to an open file}
var Buffer;                         {pointer to the buffer receiving the data}
nNumberOfBytesToRead: DWORD;        {specifies number of bytes to read from file}
var lpNumberOfBytesRead: DWORD;     {receives the number of bytes actually read}
lpOverlapped: POverLapped           {a pointer to a TOverLapped structure}
): BOOL;                            {returns TRUE or FALSE}
```

Description

This function retrieves the number of bytes specified by the nNumberOfBytesToRead parameter from the file associated with the handle specified in the hFile parameter. These bytes are stored in the buffer pointed to by the Buffer parameter. The origin of the read operation within the file is dependent upon how the file was opened and the value of the lpOverlapped parameter. Typically, the lpOverlapped parameter contains NIL, and the read operation begins at the current file pointer. After the read operation has completed, the file pointer is incremented by the number of bytes read, unless the file was opened with the FILE_FLAG_OVERLAPPED flag specified. In this case, the file pointer is not incremented, and the application must move the file pointer explicitly. The read operation will fail if it attempts to read any part of a file that has been locked with the LockFile function. The application must not access the buffer pointed to by the Buffer parameter until the read operation has completed.

Note: Windows 95/98/Me does not support asynchronous reads on disk-based files.

Parameters

hFile: A handle to the file being read. This file must have been opened with the GENERIC_READ flag specified.

lpBuffer: A pointer to a buffer receiving the information read from the file.

nNumberOfBytesToRead: Specifies the number of bytes to read from the file.

lpNumberOfBytesRead: A pointer to a double word receiving the number of bytes actually read from the file. This variable is initialized to zero before the function starts the read. This parameter must contain a pointer if the lpOverlapped parameter is set to NIL.

lpOverlapped: A pointer to a TOverlapped data structure. If the file identified by the hFile parameter was opened with the FILE_FLAG_OVERLAPPED flag specified, this parameter must contain a pointer. If the file was opened with the FILE_FLAG_OVERLAPPED flag specified, the read operation begins at the offset specified within the structure, and ReadFile may return before the read operation has completed. In this case, ReadFile will return FALSE, and GetLastError will return ERROR_IO_PENDING. The event specified in the TOverlapped structure will be signaled upon completing the read operation. If the file was not opened with the FILE_FLAG_OVERLAPPED flag specified and this parameter is not NIL, the read operation begins at the offset specified within the structure, and ReadFile does not return until the read operation is completed. If the file was not opened with the FILE_FLAG_OVERLAPPED flag specified and this parameter is NIL, the read operation begins at the current file pointer and does not return until the read operation is complete.

Note: Under Windows 95/98/Me, this parameter must be set to NIL.

The TOverlapped data structure is defined as:

```
TOverlapped = record
      Internal: DWORD;           {reserved for internal use}
      InternalHigh: DWORD;       {reserved for internal use}
      Offset: DWORD;             {specifies the file position from which to start}
      OffsetHigh: DWORD;         {the high double word of the starting offset}
      hEvent: THandle;           {a handle to an event object}
end;
```

Internal: This member is reserved for internal operating system use.

InternalHigh: This member is reserved for internal operating system use.

Offset: Specifies the low-order double word of the byte offset from the beginning of the file from which to start the operation.

OffsetHigh: Specifies the high-order double word of the byte offset from the beginning of the file from which to start the operation.

hEvent: A handle to an event object that is set to the signaled state when the operation has completed.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CreateFile, LockFile, UnlockFile, WriteFile

Example

Please see Listing 4-4 under CreateFile.

RemoveDirectory Windows.pas

Syntax

```
RemoveDirectory(
lpPathName: PAnsiChar        {the name of the directory to delete}
): BOOL;                     {returns TRUE or FALSE}
```

Description

This function deletes the directory specified by the lpPathName parameter. This directory must not contain any files, and the calling process must have delete access to the directory.

Parameters

lpPathName: A null-terminated string containing the path name of the directory to be deleted.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CreateDirectory, CreateDirectoryEx

Example

Please see Listing 4-8 under FindFirstFile.

SearchPath Windows.pas

Syntax

```
SearchPath(
lpPath: PChar;               {a pointer to a search path}
lpFileName: PChar;           {a pointer to a filename}
lpExtension: PChar;          {a pointer to a file extension}
nBufferLength: DWORD;        {the size of the lpBuffer buffer}
lpBuffer: PChar;             {a pointer to a buffer}
var lpFilePart: PChar        {a pointer to the filename}
): DWORD;                    {returns the number of characters copied to the buffer}
```

Description

The SearchPath function searches the path pointed to by the lpPath parameter for the filename pointed to by the lpFileName parameter.

Parameters

lpPath: A pointer to a null-terminated string containing the path in which to search for the specified filename. If this parameter is set to NIL, SearchPath will search the following directories in order:

1. The directory containing the application.
2. The current directory.
3. The Windows system directory as returned by the GetSystemDirectory function.
4. The Windows directory as returned by the GetWindowsDirectory function.
5. The directories listed in the PATH environment variable.

lpFileName: A pointer to a null-terminated string containing the file for which to search.

lpExtension: A pointer to a null-terminated string containing the file extension, including the period. If the file extension is not needed, or the filename pointed to by the lpFileName parameter contains an extension, this parameter can be set to NIL.

nBufferLength: Specifies the size of the buffer pointed to by the lpBuffer parameter, in characters. If this parameter is set to zero, the function returns the required size of the buffer to store the full path and filename.

lpBuffer: A pointer to a buffer which receives the path and filename of the file that was found.

lpFilePart: Receives a pointer into the lpBuffer buffer where the filename part of the returned path and filename begins, immediately following the final backslash of the path.

Return Value

If the function succeeds, it returns the number of characters copied to the buffer pointed to by the lpBuffer parameter. If the function fails, it returns zero. To get extended error information, call the GetLastError function.

See Also

FindFirstFile, FindNextFile, GetSystemDirectory, GetWindowsDirectory, SetCurrentDirectory.

Example

Please see Listing 4-8 under FindFirstFile.

SetCurrentDirectory Windows.pas

Syntax

```
SetCurrentDirectory(
lpPathName: PAnsiChar      {the name of the new current directory }
): BOOL;                   {returns TRUE or FALSE}
```

Description

This function changes the current directory of the calling process to the new directory identified by the lpPathName parameter.

Parameters

lpPathName: A null-terminated string containing the path to the new directory. This path can be either a fully qualified path or a relative path.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetCurrentDirectory

Example

Please see Listing 4-11 under SetFileAttributes.

SetEndOfFile   Windows.pas

Syntax

```
SetEndOfFile(
hFile: THandle          {a handle to an open file}
): BOOL;                {returns TRUE or FALSE}
```

Description

This function sets the end of file to the current file pointer position, either extending or truncating the file. If the file is extended, the contents of the file between the old end of file position and the new one are undetermined. If the CreateFileMapping function has been used to create a file mapping object for the file associated with the handle in the hFile parameter, the application must call UnmapViewOfFile and CloseHandle to close the file mapping object before the SetEndOfFile function can be used.

Parameters

hFile: A handle to a file whose end of file position is to be moved. This file must have been opened with the GENERIC_WRITE flag specified.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CloseHandle, CreateFile, CreateFileMapping, SetFilePointer, UnmapViewOfFile

Example

Please see Listing 4-4 under CreateFile.

SetFileAttributes Windows.pas

Syntax

```
SetFileAttributes(
lpFileName: PChar;                {the filename}
dwFileAttributes: DWORD           {file attribute flags}
): BOOL;                          {returns TRUE or FALSE}
```

Description

This function sets the file attributes for the file or directory specified by the lpFileName parameter.

Parameters

lpFileName: A null-terminated string containing the name of the file or directory from which to retrieve file attributes. This string must not be longer than MAX_PATH characters.

dwFileAttributes: Specifies the attributes being set for the file. This parameter can contain one or more values from Table 4-17.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetFileAttributes

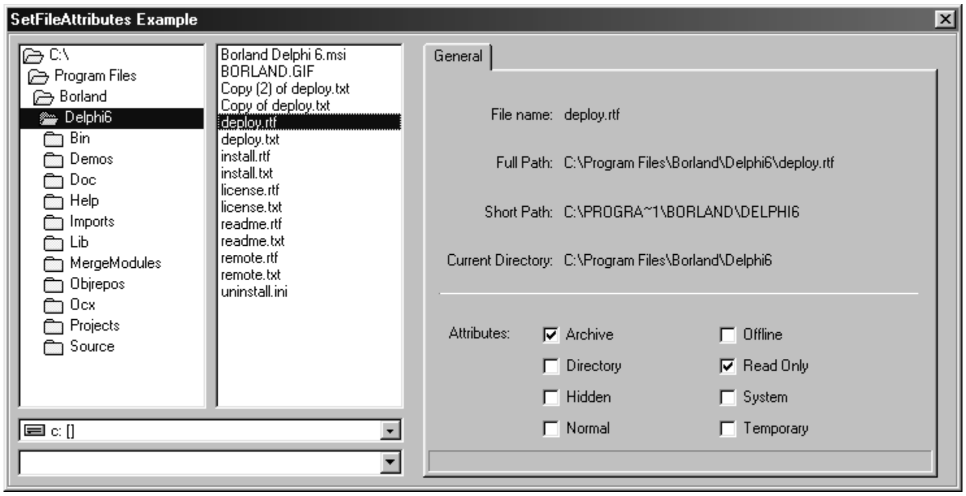
Example

Listing 4-11: Getting and setting file attributes

```
const
  {Delphi 6 does not define all available attributes}
  FILE_ATTRIBUTE_SPARSE_FILE          = $00000200;
  FILE_ATTRIBUTE_REPARSE_POINT        = $00000400;
  FILE_ATTRIBUTE_NOT_CONTENT_INDEXED  = $00002000;
  FILE_ATTRIBUTE_ENCRYPTED            = $00004000;

procedure TForm1.FileListBox1Change(Sender: TObject);
var
   PathBuffer: array[0..255] of char;   // holds path names
   FilePart: PChar;                     // a pointer to the filename
begin
  {if the file list box has an item selected, retrieve its information}
  if FileListBox1.ItemIndex > -1 then
  begin
    {unhook the checkbox OnClick methods, as accessing their Checked
     property fires the method}
    CheckBox1.OnClick := nil;
    CheckBox2.OnClick := nil;
    CheckBox3.OnClick := nil;
    CheckBox4.OnClick := nil;
    CheckBox5.OnClick := nil;
    CheckBox6.OnClick := nil;
    CheckBox7.OnClick := nil;
    CheckBox8.OnClick := nil;

    {retrieve and display the various file attributes for the selected file}
    CheckBox1.Checked := Boolean(GetFileAttributes(PChar(FileListBox1.FileName))
                                 and FILE_ATTRIBUTE_ARCHIVE);
    CheckBox2.Checked := Boolean(GetFileAttributes(PChar(FileListBox1.FileName))
                                 and FILE_ATTRIBUTE_DIRECTORY);
    CheckBox3.Checked := Boolean(GetFileAttributes(PChar(FileListBox1.FileName))
                                 and FILE_ATTRIBUTE_HIDDEN);
    CheckBox4.Checked := Boolean(GetFileAttributes(PChar(FileListBox1.FileName))
                                 and FILE_ATTRIBUTE_OFFLINE);
    CheckBox5.Checked := Boolean(GetFileAttributes(PChar(FileListBox1.FileName))
                                 and FILE_ATTRIBUTE_READONLY);
    CheckBox6.Checked := Boolean(GetFileAttributes(PChar(FileListBox1.FileName))
                                 and FILE_ATTRIBUTE_SYSTEM);
    CheckBox7.Checked := Boolean(GetFileAttributes(PChar(FileListBox1.FileName))
                                 and FILE_ATTRIBUTE_NORMAL);
    CheckBox8.Checked := Boolean(GetFileAttributes(PChar(FileListBox1.FileName))
                                 and FILE_ATTRIBUTE_TEMPORARY);

    {display the file's name}
    Label1.Caption := ExtractFileName(FileListBox1.FileName);

    {display the full, qualified path for the selected file}
    GetFullPathName(PChar(Label1.Caption), 255, PathBuffer, FilePart);
    Label10.Caption := string(PathBuffer);

    {display the short path form of the qualified path}
    GetShortPathName(PChar(DirectoryListBox1.Directory), PathBuffer, 255);
    Label11.Caption := string(PathBuffer);

    {display the current directory}
    GetCurrentDirectory(255, PathBuffer);
    Label12.Caption := string(PathBuffer);

    {rehook the checkbox OnClick methods}
    CheckBox1.OnClick := CheckBox1Click;
    CheckBox2.OnClick := CheckBox1Click;
    CheckBox3.OnClick := CheckBox1Click;
    CheckBox4.OnClick := CheckBox1Click;
    CheckBox5.OnClick := CheckBox1Click;
    CheckBox6.OnClick := CheckBox1Click;
    CheckBox7.OnClick := CheckBox1Click;
    CheckBox8.OnClick := CheckBox1Click;
  end;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
var
  FileAttributes: DWORD;      // holds collective file attributes
  ErrorMessage: Pointer;      // holds a system error string
  ErrorCode: DWORD;           // holds a system error code
begin
```

```
  {unhook the checkbox OnClick methods, as accessing their Checked
   property fires the method}
  CheckBox1.OnClick := nil;
  CheckBox2.OnClick := nil;
  CheckBox3.OnClick := nil;
  CheckBox4.OnClick := nil;
  CheckBox5.OnClick := nil;
  CheckBox6.OnClick := nil;
  CheckBox7.OnClick := nil;
  CheckBox8.OnClick := nil;

  {prepare to sum file attributes}
  FileAttributes := 0;

  {add all of the file attributes selected}
  if CheckBox1.Checked then
    FileAttributes := FileAttributes or FILE_ATTRIBUTE_ARCHIVE;
  if CheckBox2.Checked then
    FileAttributes := FileAttributes or FILE_ATTRIBUTE_DIRECTORY;
  if CheckBox3.Checked then
    FileAttributes := FileAttributes or FILE_ATTRIBUTE_HIDDEN;
  if CheckBox4.Checked then
    FileAttributes := FileAttributes or FILE_ATTRIBUTE_OFFLINE;
  if CheckBox5.Checked then
    FileAttributes := FileAttributes or FILE_ATTRIBUTE_READONLY;
  if CheckBox6.Checked then
    FileAttributes := FileAttributes or FILE_ATTRIBUTE_SYSTEM;
  if CheckBox7.Checked then
    FileAttributes := FileAttributes or FILE_ATTRIBUTE_NORMAL;
  if CheckBox8.Checked then
    FileAttributes := FileAttributes or FILE_ATTRIBUTE_TEMPORARY;

  {set the file attributes of the selected file}
  if not SetFileAttributes(PChar(FileListBox1.FileName), FileAttributes) then
  begin
    {if there was an error, display the error message}
    ErrorCode := GetLastError;
    FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER or FORMAT_MESSAGE_FROM_SYSTEM,
                  nil, ErrorCode, 0, @ErrorMessage, 0, nil);
    StatusBar1.SimpleText := 'Error Copying File: ' + PChar(ErrorMessage);
    LocalFree(hlocal(ErrorMessage));
  end;

  {rehook the checkbox OnClick methods}
  CheckBox1.OnClick := CheckBox1Click;
  CheckBox2.OnClick := CheckBox1Click;
  CheckBox3.OnClick := CheckBox1Click;
  CheckBox4.OnClick := CheckBox1Click;
  CheckBox5.OnClick := CheckBox1Click;
  CheckBox6.OnClick := CheckBox1Click;
  CheckBox7.OnClick := CheckBox1Click;
  CheckBox8.OnClick := CheckBox1Click;
end;

procedure TForm1.ComboBox1Change(Sender: TObject);
begin
```

```
    {set the current directory to the selected directory}
    SetCurrentDirectory(PChar(ComboBox1.Items[ComboBox1.ItemIndex]));

    {update the directory list box accordingly}
    DirectoryListBox1.Directory := ComboBox1.Items[ComboBox1.ItemIndex];
    DirectoryListBox1.Update;
  end;
```



File name: deploy.rtf

Full Path: C:\Program Files\Borland\Delphi6\deploy.rtf

Short Path: C:\PROGRA~1\BORLAND\DELPHI6

Current Directory: C:\Program Files\Borland\Delphi6

Attributes: [✓] Archive, [ ] Directory, [ ] Hidden, [ ] Normal, [ ] Offline, [✓] Read Only, [ ] System, [ ] Temporary

Figure 4-10: Viewing file attributes

Table 4-17: SetFileAttributes dwFileAttributes values

Value	Description
FILE_ATTRIBUTE_ARCHIVE	Indicates an archive file or directory and is used by applications to mark files and directories for removal or backup.
FILE_ATTRIBUTE_COMPRESSED	Indicates that the specified file or directory is compressed.
FILE_ATTRIBUTE_DIRECTORY	Indicates that the specified filename is a directory.
FILE_ATTRIBUTE_ENCRYPTED	**Windows NT/2000/XP and later**: Indicates that the file or directory is encrypted. This flag cannot be used with FILE_ATTRIBUTE_SYSTEM.
FILE_ATTRIBUTE_HIDDEN	Indicates that the specified file or directory is hidden and will not appear in normal directory listings.
FILE_ATTRIBUTE_NORMAL	Indicates that the specified file or directory does not have any other file attributes set.
FILE_ATTRIBUTE_NOT_CONTENT_INDEXED	**Windows NT/2000/XP and later**: Indicates that the file or directory is not to be indexed by content indexing services.
FILE_ATTRIBUTE_OFFLINE	Indicates that the specified file or directory is not immediately available and has been physically moved to offline storage.
FILE_ATTRIBUTE_READONLY	Indicates that the specified file or directory is read only. Applications may read from the file or directory, but they may not write to it or delete it.
FILE_ATTRIBUTE_REPARSE_POINT	**Windows NT/2000/XP and later**: Indicates that the file has an associated reparse point.

Value	Description
FILE_ATTRIBUTE_SPARSE_FILE	Indicates that this is a sparse file.
FILE_ATTRIBUTE_SYSTEM	Indicates that the specified file or directory is used by the system.
FILE_ATTRIBUTE_TEMPORARY	Indicates that the specified file or directory is temporary. The system will not automatically delete temporary files during shutdown.

SetFilePointer Windows.pas

Syntax

```
SetFilePointer(
hFile: THandle;                   {a handle to an open file}
lDistanceToMove: Longint;         {the distance to move in bytes}
lpDistanceToMoveHigh: Pointer;    {points to high-order double word of distance to
                                    move}
dwMoveMethod: DWORD               {movement origin flags}
): DWORD;                         {returns the low-order double word of file pointer}
```

Description

This function repositions the current file pointer within the file identified by the hFile parameter. The new position is based off of the origin of movement specified by the dwMoveMethod parameter and the 64-bit offset formed by the lDistanceToMove and lpDistanceToMoveHigh parameters.

If the file identified by the hFile parameter was opened with the FILE_FLAG_NO_BUFFERING flag specified, the file pointer can only be moved in increments of the volume's sector size. Call the GetDiskFreeSpace function to retrieve a disk volume's sector size.

Parameters

hFile: A handle to the open file whose file pointer is to be repositioned. The file must have been opened with either the GENERIC_READ or GENERIC_WRITE flags specified.

lDistanceToMove: Specifies the low-order double word of the distance, in bytes, to move the file pointer. A positive value moves the file pointer forward in the file, and a negative value moves it backward.

lpDistanceToMoveHigh: A pointer to the high-order double word of the distance, in bytes, to move the file pointer. This parameter can be set to NIL, in which case the file pointer can only be moved within a range of 2^{32}–2 bytes. If this parameter is not NIL, the file pointer can be moved within a range of 2^{64}–2 bytes, and the value pointed at by this parameter receives the new high-order double word of the file pointer when the function returns.

dwMoveMethod: Specifies the starting point of the file pointer for the movement. This parameter can be one value from Table 4-18.

Return Value

If the function succeeds, it returns the low-order double word of the new file pointer position. If the function fails, it returns $FFFFFFFF. To get extended error information, call the GetLastError function. If the lpDistanceToMoveHigh parameter is not NIL and the function failed, GetLastError will return NO_ERROR.

See Also

CreateFile, GetDiskFreeSpace, ReadFile, SetEndOfFile, WriteFile

Example

Please see Listing 4-4 under CreateFile.

Table 4-18: SetFilePointer dwMoveMethod values

Value	Description
FILE_BEGIN	The starting point for the movement begins at the beginning of the file.
FILE_CURRENT	The starting point for the movement begins at the current file pointer position.
FILE_END	The starting point for the movement begins at the end of the file.

SetFileTime Windows.pas

Syntax

```
SetFileTime(
hFile: THandle;                  {a handle to an opened file}
lpCreationTime: PFileTime;       {a pointer to buffer containing file creation time}
lpLastAccessTime: PFileTime;     {a pointer to buffer containing last file access time}
lpLastWriteTime: PFileTime       {a pointer to buffer containing last file write time}
): BOOL;                         {returns TRUE or FALSE}
```

Description

This function sets the file creation time, last file access time, and last file write time of the opened file associated with the handle given in the hFile parameter.

Parameters

hFile: A handle to the opened file whose file times are to be modified. This file must have been opened with the GENERIC_WRITE access flag specified.

lpCreationTime: A pointer to a TFileTime data structure containing the 64-bit time value with which to set the file's creation time. This parameter may be set to NIL if this time value does not need to be modified.

lpLastAccessTime: A pointer to a TFileTime data structure containing the 64-bit time value with which to set the file's last access time. This parameter may be set to NIL if this time value does not need to be modified.

lpLastWriteTime: A pointer to a TFileTime data structure containing the 64-bit time value with which to set the file's last modification time. This parameter may be set to

NIL if this time value does not need to be modified. This time value is the time value displayed in the Explorer.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

FileTimeToLocalFileTime, FileTimeToSystemTime, GetFileTime

Example

Please see Listing 4-6 under FileTimeToSystemTime.

SystemTimeToFileTime Windows.pas

Syntax

```
SystemTimeToFileTime(
const lpSystemTime: TSystemTime;     {a pointer to a TSystemTime structure}
var lpFileTime: TFileTime            {a pointer to a TFileTime structure}
): BOOL;                             {returns TRUE or FALSE}
```

Description

This function converts the values stored in the TSystemTime structure pointed to by the lpSystemTime parameter into a 64-bit file time.

Parameters

lpSystemTime: A pointer to a TSystemTime structure containing the system time information to be converted. The TSystemTime data structure is defined as:

```
TSystemTime = record
     wYear: Word;              {the current year}
     wMonth: Word;             {the month number}
     wDayOfWeek: Word;         {the day of the week number}
     wDay: Word;               {the current day of the month}
     wHour: Word;              {the current hour}
     wMinute: Word;            {the current minute}
     wSecond: Word;            {the current second}
     wMilliseconds: Word;      {the current millisecond}
end;
```

Please see the FileTimeToSystemTime function for a description of this data structure.

lpFileTime: A pointer to a TFileTime structure receiving the 64-bit converted file time.

Return Values

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, use the GetLastError function.

See Also

DosDateTimeToFileTime, FileTimeToDosDateTime, FileTimeToLocalFileTime, FileTimeToSystemTime, GetFileTime, LocalFileTimeToFileTime, SetFileTime

Example

Please see Listing 4-6 under FileTimeToSystemTime.

UnlockFile ***Windows.pas***

Syntax

```
UnlockFile(
hFile: THandle;                     {a handle to an open file}
dwFileOffsetLow: DWORD;             {low-order double word of lock region offset}
dwFileOffsetHigh: DWORD;            {high-order double word of lock region offset}
nNumberOfBytesToUnlockLow: DWORD;   {low-order double word of lock region
                                     length}
nNumberOfBytesToUnlockHigh: DWORD   {high-order double word of lock region
                                     length}
): BOOL;                            {returns TRUE or FALSE}
```

Description

This function unlocks a previously locked region in a file, providing access to the region to other processes. The unlocked region must match the locked region exactly as determined by the previous call to LockFile. All locked regions on a file should be removed before the file is closed or the application is terminated.

Parameters

hFile: A handle to the open file which is to be unlocked. This file must have been created with either the GENERIC_READ or GENERIC_WRITE flags specified.

dwFileOffsetLow: Specifies the low-order word of the offset from the beginning of the file where the locked region begins.

dwFileOffsetHigh: Specifies the high-order word of the offset from the beginning of the file where the locked region begins.

nNumberOfBytesToUnlockLow: Specifies the low-order word of the length, in bytes, of the region to unlock.

nNumberOfBytesToUnlockHigh: Specifies the high-order word of the length, in bytes, of the region to unlock.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call GetLastError.

See Also

CreateFile, LockFile

Example

Please see Listing 4-4 under CreateFile.

UnMapViewOfFile   Windows.pas

Syntax

```
UnMapViewOfFile(
lpBaseAddress: Pointer       {a pointer to the base address of the mapped view}
): BOOL;                     {returns TRUE or FALSE}
```

Description

This function removes a view of a file mapping object from the process's address space. A file that has been mapped to memory using the CreateFileMapping function is not closed until all views of the file have been closed by using UnMapViewOfFile.

Parameters

lpBaseAddress: A pointer to the base address of the mapped view of the file mapping object. This pointer must be the exact address location originally returned by the previous call to the MapViewOfFile function.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call GetLastError.

See Also

CreateFileMapping, MapViewOfFile, OpenFileMapping

Example

Please see Listing 4-5 under CreateFileMapping.

VerQueryValue   Windows.pas

Syntax

```
VerQueryValue(
pBlock: Pointer;             {a pointer to the version resource}
lpSubBlock: PChar;           {a pointer to a version value string}
var lplpBuffer: Pointer;     {a pointer to a buffer receiving a pointer to the value}
var puLen: UINT              {a pointer to a buffer receiving the value length}
): BOOL;                     {returns TRUE or FALSE}
```

Description

This function retrieves a pointer to the file version information type specified by the lpSubBlock parameter from the file version resource identified by the pBlock parameter. The pointer to this information is stored in the buffer pointed to by the lplpBuffer parameter. This function will only succeed on Win32 file images; 16-bit file images are not supported. Use GetFileVersionInfo to retrieve the file version resource for the pBlock parameter.

Parameters

pBlock: A pointer to a buffer containing the file version resource as returned by GetFileVersionInfo.

lpSubBlock: A pointer to a null-terminated string containing the type of file version information to retrieve. This string can contain one value from Table 4-19. The file version information retrieved can be either a data structure containing specific information or the actual name of a version information type. In order to use names, the application must first use the \\VarFileInfo\\Translation value to retrieve the translation code. This translation code is used in all subsequent version name values. To use the version code to specify a version name, the version code must be inserted in the version name as a string of hexadecimal numbers consisting of the low word of the conversion code concatenated with the high word of the conversion code. See the following listing for an example of using version name values.

lplpBuffer: A pointer to a buffer which receives a pointer to the requested file version information. The pointer received will point to a null-terminated string.

puLen: A pointer to a buffer which receives the length of the requested file version information in characters.

Return Value

If the function succeeds and the file version information resource contains the requested information type, it returns TRUE. If the function fails, or there is no file version information for the requested information type, it returns FALSE.

See Also

GetFileVersionInfo, GetFileVersionInfoSize

Example

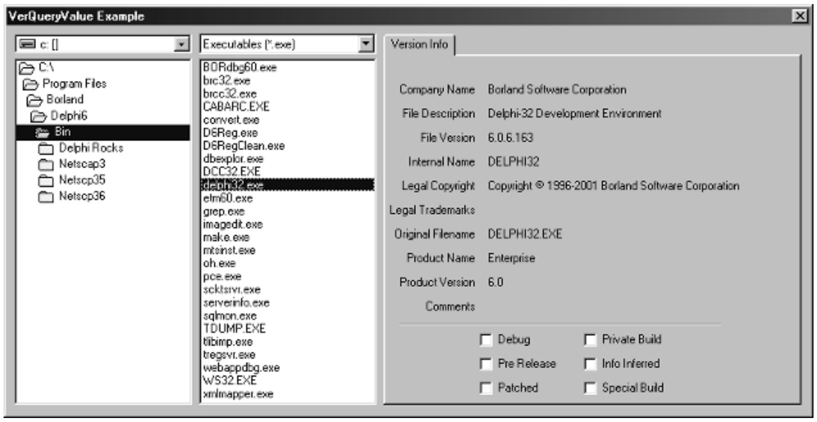
Listing 4-12: Retrieving file version information

```
procedure TForm1.FileListBox1Click(Sender: TObject);
var
  VerInfoSize: DWORD;         // holds the size of the version info resource
  GetInfoSizeJunk: DWORD;     // a junk variable, its value is ignored
  VersionInfo: Pointer;       // points to the version info resource
  Translation: Pointer;       // holds version info translation table
  InfoPointer: Pointer;       // a pointer to version information
  VersionInfoSize: UINT;      // holds the size of version information
  VersionValue: string;       // holds the version info request string
begin
  {retrieve the size of the version information resource, if one exists}
  VerInfoSize := 0;
  VerInfoSize := GetFileVersionInfoSize(PChar(FileListBox1.FileName), GetInfoSizeJunk);

  {if there was a version information resource available...}
  if VerInfoSize>0 then
```

```
begin
  {hide the 'not available' indicator}
  Label1.Visible := FALSE;

  {retrieve enough memory to hold the version resource}
  GetMem(VersionInfo, VerInfoSize);

  {retrieve the version resource for the selected file}
  GetFileVersionInfo(PChar(FileListBox1.FileName), 0, VerInfoSize, VersionInfo);

  {retrieve a pointer to the translation table}
  VerQueryValue(VersionInfo, '\\VarFileInfo\\Translation',
                Translation, VersionInfoSize

  {initialize the version value request string}
  VersionValue :='\\StringFileInfo\\'+
                IntToHex(LoWord(LongInt(Translation^)),4) +
                IntToHex(HiWord(LongInt(Translation^)),4) +
                '\\';

  {retrieve and display the company name}
  VerQueryValue(VersionInfo, PChar(VersionValue + 'CompanyName'),
                InfoPointer, VersionInfoSize);
  Label17.Caption := string(PChar(InfoPointer));

  {retrieve and display the file description}
  VerQueryValue(VersionInfo, PChar(VersionValue + 'FileDescription'),
                InfoPointer, VersionInfoSize);
  Label16.Caption := string(PChar(InfoPointer));

  {retrieve and display the file version}
  VerQueryValue(VersionInfo, PChar(VersionValue + 'FileVersion'), InfoPointer,
                VersionInfoSize);
  Label15.Caption := string(PChar(InfoPointer));

  {retrieve and display the internal filename}
  VerQueryValue(VersionInfo, PChar(VersionValue + 'InternalName'),InfoPointer,
                VersionInfoSize);
  Label14.Caption := string(PChar(InfoPointer));

  {retrieve and display the legal copyright}
  VerQueryValue(VersionInfo, PChar(VersionValue + 'LegalCopyright'),
                InfoPointer, VersionInfoSize);
  Label13.Caption := string(PChar(InfoPointer));

  {retrieve and display the legal trademarks}
  if VerQueryValue(VersionInfo, PChar(VersionValue + 'LegalTrademarks'),
                   InfoPointer, VersionInfoSize) then
    Label19.Caption := string(PChar(InfoPointer))
  else
    Label19.Caption := '';

  {retrieve and display the original filename}
  VerQueryValue(VersionInfo, PChar(VersionValue + 'OriginalFilename'),
                InfoPointer, VersionInfoSize);
  Label12.Caption := string(PChar(InfoPointer));
```

```
    {retrieve and display the product name}
    VerQueryValue(VersionInfo, PChar(VersionValue + 'ProductName'), InfoPointer,
                  VersionInfoSize);
    Label11.Caption := string(PChar(InfoPointer));

    {retrieve and display the product version}
    VerQueryValue(VersionInfo, PChar(VersionValue + 'ProductVersion'),
                  InfoPointer, VersionInfoSize);
    Label10.Caption := string(PChar(InfoPointer));

    {retrieve and display the comments. some version info resources may
     not have this information.}
    if VerQueryValue(VersionInfo, PChar(VersionValue + 'Comments'), InfoPointer,
                     VersionInfoSize) then
      Label21.Caption := string(PChar(InfoPointer))
    else
      Label21.Caption := '';

    {retrieve and display file build flags}
    if VerQueryValue(VersionInfo, '\', InfoPointer, VersionInfoSize) then
    begin
      CheckBox1.Checked := BOOL(TVSFixedFileInfo(InfoPointer^).dwFileFlags and
                           VS_FF_DEBUG);
      CheckBox2.Checked := BOOL(TVSFixedFileInfo(InfoPointer^).dwFileFlags and
                           VS_FF_PRERELEASE);
      CheckBox3.Checked := BOOL(TVSFixedFileInfo(InfoPointer^).dwFileFlags and
                           VS_FF_PATCHED);
      CheckBox4.Checked := BOOL(TVSFixedFileInfo(InfoPointer^).dwFileFlags and
                           VS_FF_PRIVATEBUILD);
      CheckBox5.Checked := BOOL(TVSFixedFileInfo(InfoPointer^).dwFileFlags and
                           VS_FF_INFOINFERRED);
      CheckBox6.Checked := BOOL(TVSFixedFileInfo(InfoPointer^).dwFileFlags and
                           VS_FF_SPECIALBUILD);
    end
    else
    begin
      CheckBox1.Checked := FALSE;
      CheckBox2.Checked := FALSE;
      CheckBox3.Checked := FALSE;
      CheckBox4.Checked := FALSE;
      CheckBox5.Checked := FALSE;
      CheckBox6.Checked := FALSE;
    end;

    {free the version resource memory}
    FreeMem(VersionInfo, VerInfoSize);
  end
  else
  begin
    {otherwise, indicate that no version information is available}
    Label1.Visible := TRUE;

    {delete any previous version information}
    Label17.Caption := '';
    Label16.Caption := '';
    Label15.Caption := '';
```

```
        Label14.Caption := '';
        Label13.Caption := '';
        Label12.Caption := '';
        Label11.Caption := '';
        Label10.Caption := '';
        Label19.Caption := '';
        Label21.Caption := '';

        CheckBox1.Checked := FALSE;
        CheckBox2.Checked := FALSE;
        CheckBox3.Checked := FALSE;
        CheckBox4.Checked := FALSE;
        CheckBox5.Checked := FALSE;
        CheckBox6.Checked := FALSE;
      end;
    end;
```



Company Name	Borland Software Corporation
File Description	Delphi-32 Development Environment
File Version	6.0.6.163
Internal Name	DELPHI32
Legal Copyright	Copyright © 1996-2001 Borland Software Corporation
Legal Trademarks	
Original Filename	DELPHI32.EXE
Product Name	Enterprise
Product Version	6.0
Comments	

☐ Debug ☐ Private Build
☐ Pre Release ☐ Info Inferred
☐ Patched ☐ Special Build

Figure 4-11: The file version information

Table 4-19: VerQueryValue lpSubBlock values

Value	Description
\	Stores a pointer to a TVSFixedFileInfo data structure in the buffer pointed to by the lplpBuffer parameter. The TVSFixedFileInfo structure contains specific file version information.
\\VarFileInfo\\Translation	Retrieves a pointer to a translation value. This translation value is needed for the following values.
\\StringFileInfo\\<translation value>\\CompanyName	Retrieves a pointer to a string containing the name of the company that created the file.
\\StringFileInfo\\<translation value>\\FileDescription	Retrieves a pointer to a string containing a description of the file.
\\StringFileInfo\\<translation value>\\FileVersion	Retrieves a pointer to a string containing the file version number.
\\StringFileInfo\\<translation value>\\InternalName	Retrieves a pointer to a string containing the internal name of the file.
\\StringFileInfo\\<translation value>\\LegalCopyright	Retrieves a pointer to a string containing the legal copyright of the company that created the file, if any.

Value	Description
\\StringFileInfo\\<translation value> \\LegalTrademarks	Retrieves a pointer to a string containing the legal trademarks of the company that created the file, if any.
\\StringFileInfo\\<translation value> \\OriginalFilename	Retrieves a pointer to a string containing the original name of the file.
\\StringFileInfo\\<translation value> \\ProductName	Retrieves a pointer to a string containing the name of the product to which the file belongs.
\\StringFileInfo\\<translation value> \\ProductVersion	Retrieves a pointer to a string containing the version of the product to which the file belongs.
\\StringFileInfo\\<translation value> \\Comments	Retrieves a pointer to a string containing any comments about the file.

The TVSFixedFileInfo data structure is defined as:

```
TVSFixedFileInfo = packed record
      dwSignature: DWORD;             {the data structure signature}
      dwStrucVersion: DWORD;          {the data structure version}
      dwFileVersionMS: DWORD;         {most significant 32 bits of the file version}
      dwFileVersionLS: DWORD;         {least significant 32 bits of the file version}
      dwProductVersionMS: DWORD;      {most significant 32 bits of product version}
      dwProductVersionLS: DWORD;      {least significant 32 bits of product version}
      dwFileFlagsMask: DWORD;         {bitmask representing valid version
                                       attributes}
      dwFileFlags: DWORD;             {version attribute flags}
      dwFileOS: DWORD;                {file operating system type}
      dwFileType: DWORD;              {file type flags}
      dwFileSubtype: DWORD;           {file subtype flags}
      dwFileDateMS: DWORD;            {most significant 32 bits of the file date}
      dwFileDateLS: DWORD;            {least significant 32 bits of the file date}
end;
```

dwSignature: Always contains the value $FEEF04BD.

dwStrucVersion: Specifies the version number of this structure where the high-order word indicates the major version number and the low-order word indicates the minor version number.

dwFileVersionMS: Specifies the most significant 32 bits of the file's version number. This value can be combined with the value of the dwFileVersionLS member to obtain the full 64-bit file version number.

dwFileVersionLS: Specifies the least significant 32 bits of the file's version number. This value can be combined with the value of the dwFileVersionMS member to obtain the full 64-bit file version number.

dwProductVersionMS: Specifies the most significant 32 bits of the version number of the product to which this file belongs. This value can be combined with the value of the dwProductVersionLS member to obtain the full 64-bit product version number.

dwProductVersionLS: Specifies the least significant 32 bits of the version number of the product to which this file belongs. This value can be combined with the value of the dwProductVersionMS member to obtain the full 64-bit product version number.

dwFileFlagsMask: A bitmask indicating which bits of the dwFileFlags member are valid.

dwFileFlags: A series of bit flags indicating various attributes of the file. This member can contain one or more flags from Table 4-20.

dwFileOS: Specifies the operating system for which this file was designed to operate on. This member can be one value from Table 4-21.

dwFileType: Indicates the file type. This member can be one value from Table 4-22.

dwFileSubtype: Indicates the function of the file. This value is dependent on the value of the dwFileType member and can be one value from Table 4-23. For any values of dwFileType not listed in the table, dwFileSubtype will contain zero. If the dwFileType parameter contains VFT_VXD, dwFileSubtype will contain the virtual device identifier.

dwFileDateMS: Specifies the most significant 32 bits of the file's date and time stamp.

dwFileDateLS: Specifies the least significant 32 bits of the file's date and time stamp.

Table 4-20: VerQueryValue TVSFixedFileInfo.dwFileFlags values

Value	Description
VS_FF_DEBUG	The file contains debug information and was compiled with debug features enabled.
VS_FF_PRERELEASE	The file is a development version and is not meant for public distribution.
VS_FF_PATCHED	The file has been modified and is not identical to the original shipping version.
VS_FF_PRIVATEBUILD	The file was not built using standard release procedures and is intended for internal use only.
VS_FF_INFOINFERRED	The file's version information was created dynamically and some of the information in this structure may be incomplete or incorrect.
VS_FF_SPECIALBUILD	The file was built using standard release procedures, but it is a variation of the normal shipping version of the file.

Table 4-21: VerQueryValue TVSFixedFileInfo.dwFileOS values

Value	Description
VOS_UNKNOWN	The file was designed for an unknown operating system.
VOS_NT	The file was designed for use under Windows NT.
VOS_WINDOWS32	The file was designed for use under the Win32 API.
VOS_DOS_WINDOWS32	The file was designed for use under the Win32 API running on MS-DOS.

Value	Description
VOS_NT_WINDOWS32	The file was designed for use under the Win32 API running on Windows NT.

Table 4-22: VerQueryValue TVSFixedFileInfo.dwFileType values

Value	Description
VFT_UNKNOWN	The file type is unknown.
VFT_APP	The file is an application.
VFT_DLL	The file is a dynamic link library.
VFT_DRV	The file contains a device driver.
VFT_FONT	The file contains a font.
VFT_VXD	The file contains a virtual device driver.
VFT_STATIC_LIB	The file contains a static link library.

Table 4-23: VerQueryValue TVSFixedFileInfo.dwFileSubtype values

Value of dwFileType	Value	Description
VFT_DRV	VFT2_UNKNOWN	The driver type is unknown.
VFT_DRV	VFT2_DRV_PRINTER	The file contains a printer driver.
VFT_DRV	VFT2_DRV_KEYBOARD	The file contains a keyboard driver.
VFT_DRV	VFT2_DRV_LANGUAGE	The file contains a language driver.
VFT_DRV	VFT2_DRV_DISPLAY	The file contains a display driver.
VFT_DRV	VFT2_DRV_MOUSE	The file contains a mouse driver.
VFT_DRV	VFT2_DRV_NETWORK	The file contains a network driver.
VFT_DRV	VFT2_DRV_SYSTEM	The file contains a system driver.
VFT_DRV	VFT2_DRV_INSTALLABLE	The file contains an installable driver.
VFT_DRV	VFT2_DRV_SOUND	The file contains a sound driver.
VFT_FONT	VFT2_UNKNOWN	The font type is unknown.
VFT_FONT	VFT2_FONT_RASTER	The file contains a raster font.
VFT_FONT	VFT2_FONT_VECTOR	The file contains a vector font.
VFT_FONT	VFT2_FONT_TRUETYPE	The file contains a TrueType font.

WriteFile *Windows.pas*

Syntax

```
WriteFile(
hFile: THandle;                       {a handle to an open file}
const Buffer;                         {the buffer containing the data to be written}
nNumberOfBytesToWrite: DWORD;         {the number of bytes to write to the file}
var lpNumberOfBytesWritten: DWORD;{receives number of bytes actually written}
lpOverlapped: POverlapped             {a pointer to a TOverlapped structure}
): BOOL;                              {returns TRUE or FALSE}
```

Description

This function writes the number of bytes specified by the nNumberOfBytesToWrite parameter to the file associated with the handle specified in the hFile parameter. These bytes come from the buffer pointed to by the Buffer parameter. The origin of the write operation within the file is dependent upon how the file was opened and the value of the lpOverlapped parameter. Typically, the lpOverlapped parameter contains NIL, and the write operation begins at the current file pointer. After the write operation has completed, the file pointer is incremented by the number of bytes written, unless the file was opened with the FILE_FLAG_OVERLAPPED flag specified. In this case, the file pointer is not incremented, and the application must move the file pointer explicitly. The write operation will fail if it attempts to write to any part of a file that has been locked with the LockFile function. The application must not access the buffer pointed to by the Buffer parameter until the write operation has completed.

Note: Windows 95/98/Me does not support asynchronous writes on disk-based files.

Parameters

hFile: A handle to the file being read. This file must have been opened with the GENERIC_WRITE flag specified.

Buffer: A pointer to a buffer containing the information to be written to the specified file.

nNumberOfBytesToWrite: Specifies the number of bytes to be written to the file.

lpNumberOfBytesWritten: A pointer to a double word receiving the number of bytes actually written to the file. This variable is initialized to zero before the function starts the write. This parameter must contain a pointer if the lpOverlapped parameter is set to NIL.

lpOverlapped: A pointer to a TOverlapped data structure. If the file identified by the hFile parameter was opened with the FILE_FLAG_OVERLAPPED flag specified, this parameter must contain a pointer. If the file was opened with the FILE_FLAG_OVERLAPPED flag specified, the write operation begins at the offset specified within the structure, and WriteFile may return before the write operation has completed. In this case, WriteFile will return FALSE, and GetLastError will return ERROR_IO_PENDING. The event specified in the TOverlapped structure will be signaled upon completing the read operation. If the file was not opened with the FILE_FLAG_OVERLAPPED flag specified and this parameter is not NIL, the write operation begins at the offset specified within the structure, and WriteFile does not return until the write operation is completed. If the file was not opened with the FILE_FLAG_OVERLAPPED flag specified and this parameter is NIL, the write operation begins at the current file pointer and does not return until the write operation is complete.

Note: Under Windows 95/98/Me, this parameter must be set to NIL.

The TOverlapped data structure is defined as:

```
TOverlapped = record
      Internal: DWORD;            {reserved for internal use}
      InternalHigh: DWORD;        {reserved for internal use}
      Offset: DWORD;              {specifies the file position from which to start}
      OffsetHigh: DWORD;          {the high double word of the starting offset}
      hEvent: THandle;            {a handle to an event object}
end;
```

Please see the ReadFile function for a description of this data structure.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CreateFile, LockFile, ReadFile, SetEndOfFile, UnlockFile

Example

Please see Listing 4-4 under CreateFile.

Chapter 5

Input Functions

Windows has the responsibility of providing input services to an application from a variety of input devices including the keyboard, mouse, and joystick. The input resources are shared devices. Windows performs reasonably well in directing the inputs to the correct application in a multitasking environment. At the same time, an application can have some control in its own management of monitoring all system inputs from an input device, such as the mouse.

The Keyboard

Windows can provide keyboard input functionality in the context of international character sets. The keyboard state may be queried to see which key combination the user might have pressed. The keyboard state, along with the virtual key codes, gives the programmer capabilities to deploy an internationally ready application. The underlying Windows operating system takes care of much of the low-level language translation work.

This is accomplished by using what is called an input locale identifier. Formerly known as a keyboard layout, an input locale identifier includes information about both the input language as well as the physical layout of the keyboard. However, it has a much broader concept than the original keyboard layout, as an input locale identifier encompasses alternative forms of input, such as a speech-to-text converter or an Input Method Editor (IME). The programmer can have an application load, unload, and select an active input locale identifier. There are API functions for getting keyboard characters translated in the context of the active input locale identifier into virtual keys. The active input locale identifier can be changed dynamically, so applications that make use of such functionality should handle the WM_INPUTLANGCHANGE message to detect this change.

The keyboard can be emulated with the keybd_event API function. This function generates the same Windows messages that the system itself would generate from actual keypresses. Keyboard messages are normally sent to the window that has focus.

The Mouse

The mouse is another shared device that is monitored and managed by Windows. The mouse activity is normally reported to the window that is directly under the mouse cursor. However, an application can assign or "capture" the mouse activity and cause the mouse messages to go to a capture window. This behavior continues until the capture is released.

There are also API functions for restricting the motion of the mouse to a rectangular area. The ClipCursor function provides this capability. An application that assumes such a global control of a valuable system resource should take care that it releases the device when appropriate.

Mouse activity can be simulated just like the keyboard can. See the program example for the mouse_event function. The mouse motion, location, and button activity can all be synthesized using the mouse_event function. It may be easier to control the mouse programmatically using mouse_event rather than sending mouse messages to a target window.

The input functions that provide input simulation described in this chapter can be used for creating training or demo programs. By being able to simulate keystrokes and mouse activity, an application can demonstrate to the user how an application works. The value of seeing the mouse move on the screen under program control can have a big impact on training effectiveness. These functions can also be used to provide hints or other user interface services as the application needs dictate.

Delphi vs. the Windows API

Quite simply, few, if any, of the functions discussed in this chapter have any sort of equivalent in Delphi. For applications that deal with specialized input or must deal with input in a specialized way, these functions provide a wide assortment of necessary functionality. Joystick input, synthesis of keyboard and mouse events, keyboard key translation, and functions that deal with the cursor, mouse buttons, and caret are all addressed by these Windows API functions.

Input Functions

The following input functions are covered in this chapter.

Table 5-1: Input functions

Function	Description
ActivateKeyboardLayout	Activates a specified keyboard layout.
ClipCursor	Confines the mouse cursor to a rectangular region.
DragDetect	Captures the mouse and tracks its movement.
GetAsyncKeyState	Determines if a specific key is up or down.
GetCapture	Retrieves the handle of the window with the mouse capture.

Function	Description
GetCaretBlinkTime	Retrieves the caret blink rate.
GetCaretPos	Retrieves the caret position.
GetClipCursor	Retrieves the mouse cursor confinement coordinates.
GetCursorPos	Retrieves the mouse cursor position relative to the screen.
GetDoubleClickTime	Retrieves the double-click interval.
GetInputState	Determines if there are mouse or keyboard messages in the message queue.
GetKeyboardLayout	Retrieves a keyboard layout.
GetKeyboardLayoutList	Retrieves a list of keyboard layouts for the current locale.
GetKeyboardLayoutName	Retrieves a keyboard layout name.
GetKeyboardState	Retrieves the up or down state of all 256 virtual key codes.
GetKeyboardType	Retrieves information about the keyboard.
GetKeyNameText	Retrieves a string representing the name of the key.
GetKeyState	Retrieves the up or down state of an individual virtual key.
keybd_event	Simulates keyboard activity.
joyGetDevCaps	Gets the capabilities of an installed joystick.
joyGetNumDevs	Retrieves the number of joysticks installed.
joyGetPos	Retrieves the position of the joystick.
joyGetPosEx	Retrieves additional information concerning the joystick position.
joyGetThreshold	Retrieves the joystick threshold value.
joyReleaseCapture	Releases joystick message capturing.
joySetCapture	Captures joystick messages.
joySetThreshold	Sets the joystick threshold value.
LoadKeyboardLayout	Loads a keyboard layout.
MapVirtualKey	Translates a virtual key code.
MapVirtualKeyEx	Translates a virtual key code according to the keyboard layout.
mouse_event	Simulates mouse activity.
OemKeyScan	Converts an OEM ASCII codes.
ReleaseCapture	Releases mouse capture.
SetCapture	Captures mouse messages.
SetCaretBlinkTime	Sets the caret blink rate.
SetCaretPos	Sets the caret position.
SetCursorPos	Sets the mouse cursor position.
SetDoubleClickTime	Sets the double-click interval.
SetKeyboardState	Sets the state of all 256 virtual key codes.
SwapMouseButton	Swaps the logical mouse buttons.
UnloadKeyboardLayout	Unloads a loaded keyboard layout.
VkKeyScan	Translates a character into a virtual key code.
VkKeyScanEx	Translates a character into a virtual key code according to the keyboard layout.

ActivateKeyboardLayout Windows.pas

Syntax

```
ActivateKeyboardLayout(
klh: HKL;               {input local identifier}
Flags: UINT             {activation flag}
): HKL;                 {returns previous handle}
```

Description

The ActivateKeyboardLayout function activates the keyboard layout for the specified local identifier. In Windows 95/98, it affects the current thread. In Windows NT and later, it affects all threads.

Parameters

klh: The input local identifier (this was formerly known as the keyboard layout handle and defines both a local and a keyboard layout for the locale). Under Windows 95/98/Me, this handle can be obtained by calling LoadKeyboardLayout or GetKeyboardLayoutList. Under Windows NT/2000 and later, the handle can only be obtained by a call to LoadKeyboardLayout. This parameter can also be set to HKL_NEXT or HKL_PREV, which refer to the next or previous entries in the keyboard layout list.

Flags: Specifies keyboard layout options and can be one value from the following table.

Return Value

If the function succeeds, the return value is the previous input locale identifier. If the function fails, it returns zero, indicating no matching keyboard layout was found. To get extended error information, call the GetLastError function.

See Also

GetKeyboardLayoutList, LoadKeyboardLayout, UnloadKeyboardLayout

Example

Please see Listing 5-6 under LoadKeyboardLayout.

Table 5-2: ActivateKeyboardLayout Flags values

Value	Description
0	Do not reorder.
KLF_REORDER	Reorder the keyboard handle list object by placing the specified layout at the head of the list. Without this flag, the list is rotated without any change in the keyboard layout order.
KLF_RESET	**Windows 2000 and later**: If this is included and KLF_SHIFTLOCK is not, the caps lock state is toggled by pressing the Caps Lock key. If KLF_SHIFTLOCK is included, the caps lock state is toggled by pressing either Shift key. This setting is persistent and is written to the user's profile in the registry.

Value	Description
KLF_SETFORPROCESS	**Windows 2000 and later**: Activates the indicated local identifier (and physical keyboard layout) for the entire process. The current thread's active window receives a WM_INPUTLANGCHANGE message.
KLF_SHIFTLOCK	**Windows 2000 and later**: Used with KLF_RESET, listed above.
KLF_UNLOADPREVIOUS	Obsolete; use UnloadKeyboardLayout instead.

ClipCursor Windows.pas

Syntax

```
ClipCursor(
lpRect: PRect              {specifies the rectangular clipping region}
): BOOL;                   {returns TRUE or FALSE}
```

Description

The ClipCursor function limits the cursor movements to the rectangular region specified by the lpRect parameter. This affects all cursor movement in all applications until the original rectangular coordinates are restored. To save the original coordinates, call the GetClipCursor function. Once the cursor has been confined with ClipCursor, any call to the SetCursorPos function will be based on the specified clipping region coordinates.

Parameters

lpRect: Points to a TRect structure which specifies the coordinates for the clipping region. If this parameter is zero, the cursor is free to move anywhere.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetClipCursor

Example

Please see Listing 5-10 under SwapMouseButton.

DragDetect Windows.pas

Syntax

```
DragDetect(
p1: HWND;                  {handle of window receiving mouse input}
p2: TPoint                 {initial mouse position}
): BOOL;                   {returns TRUE or FALSE}
```

Description

The DragDetect function captures mouse messages and receives the cursor coordinates until the left mouse button is released, the Esc key is pressed, or the cursor goes outside of the drag rectangle around the point specified by parameter p2. The specifications for the drag rectangle may be obtained with the GetSystemMetrics API function.

Parameters

p1: Handle of the window receiving the mouse input.

p2: Position of the mouse in coordinates relative to the screen.

Return Value

If the function succeeds, and if the mouse moves inside of the drag rectangle while holding the left mouse button down, the function returns TRUE; otherwise, it returns FALSE.

See Also

GetSystemMetrics*

Example

Listing 5-1: Using DragDetect in a graphics paint application

```
procedure TForm1.Panel1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  XY_coordinates, UV_coordinates: TPoint;   // holds coordinates
begin
  {coordinates for the rectangle}
  XY_coordinates.X:= Panel1.Width;
  XY_coordinates.Y:= Panel1.Height;

  {detect mouse drags on the panel}
  while(DragDetect(Panel1.Handle,XY_coordinates)) do
  begin
    {retrieve the cursor position}
    GetCursorPos(UV_coordinates);
    Windows.ScreenToClient(Panel1.Handle, UV_coordinates);

    {display the mouse coordinates}
    Edit1.Text:= IntToStr(UV_coordinates.X);
    Edit2.Text:= IntToStr(UV_coordinates.Y);

    {draw a pixel at the mouse coordinates}
    SetPixel(GetDc(Panel1.Handle), UV_coordinates.x, UV_coordinates.Y,
             ColorGrid1.ForegroundColor);
  end
end;

procedure TForm1.ColorGrid1Change(Sender: TObject);
begin
  {erase the background}
  Panel1.Color:= ColorGrid1.BackgroundColor;
```

```
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  {redraw the panel}
  Panel1.Refresh;
end;
```

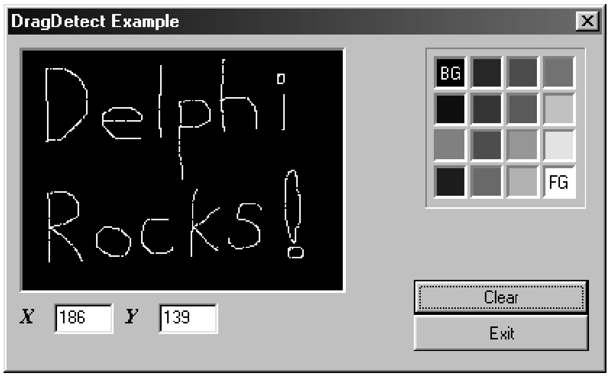


Figure 5-1: The detected drag coordinates

GetAsyncKeyState   Windows.pas

Syntax

```
GetAsyncKeyState(
vKey: Integer              {a virtual key code}
): SHORT;                  {keypress state code}
```

Description

This function determines if the key indicated by the virtual key code specified in the vKey parameter is up or down at the time of the function call. It also determines if the key was pressed after a previous call to GetAsyncKeyState. This function will also work with mouse buttons, but it reports the state of the physical mouse buttons regardless of logical mouse button mapping.

Parameters

vKey: Specifies a virtual key code.

Return Value

If the function succeeds, and the most significant bit is set in the return value, the specified key is down at the time of the function call. If the least significant bit is set, the key was pressed after a previous call to the GetAsyncKeyState function. If the function fails, or a window in another thread has focus, it returns zero.

See Also

GetKeyboardState, GetKeyState, GetSystemMetrics*, MapVirtualKey, SetKeyboardState

Example

Please see Listing 5-7 under MapVirtualKey.

GetCapture ***Windows.pas***

Syntax

GetCapture: HWND; {returns a handle to the window that has the capture}

Description

The GetCapture function determines which window has the mouse capture. Only one window may have mouse capture assigned to it, and that window will receive the mouse input regardless of where the mouse cursor is on the screen.

Return Value

If the function succeeds, and a window in the current thread has the mouse capture, it returns the handle to the window with the mouse capture; otherwise, it returns zero.

See Also

ReleaseCapture, SetCapture

Example

Please see Listing 5-10 under SwapMouseButton.

GetCaretBlinkTime ***Windows.pas***

Syntax

GetCaretBlinkTime: UINT; {returns the blink time interval in milliseconds}

Description

The GetCaretBlinkTime function returns the blink time of the caret in milliseconds. The blink time is the time interval between the first appearance and the second appearance of the caret.

Return Value

If the function succeeds, it returns the caret blink time in milliseconds; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

SetCaretBlinkTime

Example

Please see Listing 5-9 under SetCaretBlinkTime.

GetCaretPos Windows.pas

Syntax

```
GetCaretPos(
var lpPoint: TPoint      {points to caret coordinates}
): BOOL;                 {returns TRUE or FALSE}
```

Description

The GetCaretPos function retrieves the current position of the caret, in client coordinates.

Parameters

lpPoint: Points to a TPoint structure that receives the coordinates of the caret. The coordinates are always given relative to the client area.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call GetLastError function.

See Also

SetCaretPos, SetCursorPos

Example

Please see Listing 5-9 under SetCaretBlinkTime.

GetClipCursor Windows.pas

Syntax

```
GetClipCursor(
var lpRect: TRect        {coordinates for the clipping region}
): BOOL;                 {returns TRUE or FALSE}
```

Description

The GetClipCursor function retrieves the coordinates of the current clipping region, defined as the rectangle where the mouse cursor is confined.

Parameters

lpRect: Points to a TRect structure that receives the coordinates for the clipping region. The TRect structure must be allocated by the caller.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call GetLastError function.

See Also

ClipCursor, GetCursorPos

Example

Please see Listing 5-10 under SwapMouseButton.

GetCursorPos Windows.pas

Syntax

```
GetCursorPos(
var lpPoint: TPoint      {receives coordinates of cursor}
): BOOL;                 {returns TRUE or FALSE}
```

Description

The GetCursorPos function retrieves the mouse cursor position relative to the screen.

Parameters

lpPoint: Points to a TPoint structure, which receives the current mouse cursor's position in screen coordinates. This structure must be allocated by the caller.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

ClipCursor, SetCursorPos, SetCaretPos

Example

Please see Listing 5-10 under SwapMouseButton.

GetDoubleClickTime Windows.pas

Syntax

```
GetDoubleClickTime: UINT;        {returns time interval elapsed
                                  between two mouse clicks}
```

Description

The GetDoubleClickTime function gets the time interval in milliseconds that can elapse between the first and second mouse clicks. If the mouse moves or the time interval between clicks is greater than this time interval, the system will not treat the event as a double mouse click. To change the double-click time, use the SetDoubleClickTime function.

Return Value

If the function succeeds, it returns the double-click time in milliseconds. This function does not indicate an error upon failure.

See Also

SetDoubleClickTime

Example

Please see Listing 5-10 under SwapMouseButton.

GetInputState Windows.pas

Syntax

```
GetInputState: BOOL;          {returns TRUE or FALSE}
```

Description

GetInputState examines the message queue for mouse, button, keyboard, or timer event messages. It returns a Boolean value reflecting the existence of these message types in the queue.

Return Value

This function returns TRUE if there are input messages in the queue, or FALSE if not. The function does not indicate an error upon failure.

See Also

GetQueueStatus*

Example

Please see Listing 5-2 under GetKeyboardType.

GetKeyboardLayout Windows.pas

Syntax

```
GetKeyboardLayout(
dwLayout: DWORD               {thread being queried}
): HKL;                       {returns an input locale identifier}
```

Description

GetKeyboardLayout retrieves the active input locale identifier for the specified thread. To get the layout for the current thread, set the dwLayout parameter to zero.

Parameters

dwLayout: Specifies the handle for the thread that is being queried. This must be a valid handle for a thread.

Return Value

If the function succeeds, it returns the input locale identifier for the specified thread. The low-order word is the language identifier for the thread, and the high-order word is the device handle for the keyboard layout. If the function fails, it returns zero.

See Also

GetKeyboardLayoutList, LoadKeyboardLayout, UnloadKeyboardLayout

Example

Please see Listing 5-11 under VkKeyScanEx.

GetKeyboardLayoutList Windows.pas

Syntax

```
GetKeyboardLayoutList(
nBuff: Integer;          {number of keyboard layout handles}
var List                 {receives array of input locale identifiers}
): UINT;                 {returns the number of identifiers}
```

Description

The GetKeyboardLayoutList function retrieves the list input locale identifiers for the current system locale. It can be used to retrieve the actual list or the number of entries in the list.

Parameters

nBuff: Specifies the number of handles that the buffer can hold. If this parameter is set to zero, the function returns the number of entries in the list.

List: Points to an array that receives the input locale identifiers. If the nBuff parameter is zero, this parameter is ignored.

Return Value

If nBuff is not zero and the function succeeds, it returns the number of identifiers placed in the buffer pointed to by the List parameter. If nBuff is zero, GetKeyboard-LayoutList returns the number of input locale identifiers available. If the function fails, it returns zero.

See Also

LoadKeyboardLayout, GetKeyboardLayout, UnloadKeyboardLayout

Example

Please see Listing 5-6 under LoadKeyboardLayout.

GetKeyboardLayoutName Windows.pas

Syntax

```
GetKeyboardLayoutName(
pwszKLID: PChar          {output buffer for input locale name}
): BOOL;                 {returns TRUE or FALSE}
```

Description

The GetKeyboardLayoutName function retrieves the name of the active input locale identifier in the form of a string. The buffer pointed to by the pwszKLID parameter will receive a null-terminated string representation of a hexadecimal value composed of a primary language identifier and a sub-language identifier. Under Windows 95/98/Me, this function retrieves the active input locale identifier only for the calling thread. Under Windows NT/2000 and later, it retrieves the input locale identifier for the system.

Parameters

pwszKLID: A pointer to a null-terminated string which receives the name of the keyboard layout identifier.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetKeyboardLayoutList, LoadKeyboardLayout, UnloadKeyboardLayout

Example

Please see Listing 5-6 under LoadKeyboardLayout.

GetKeyboardState Windows.pas

Syntax

```
GetKeyboardState(
var KeyState: TKeyboardState       {array to receive virtual key codes}
): BOOL;                           {returns TRUE or FALSE}
```

Description

The GetKeyboardState function retrieves the status of all 256 virtual keys into an array of 256 bytes. Use the virtual key codes as an index into this array to retrieve individual virtual key states (i.e., KeyState[VK_SHIFT]). The values in the array change as keyboard messages are removed from the queue, not when they are posted. To get the status of a single key, use the GetKeyState or GetAsyncKeyState functions.

Parameters

KeyState: Points to a TKeyboardState structure, which is an array of 256 bytes. This array receives the information about key states for all 256 virtual keys. If the high-order bit of an array value is 1, that key is pressed. If the low-order bit is 1, the key is toggled on, such as the Caps, Shift, or Alt keys. TKeyboardState is defined as follows:

```
TKeyboardState = array[0..255] of Byte;       {virtual key code states}
```

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetAsyncKeyState, GetKeyNameText, MapVirtualKey, SetKeyboardState

Example

Please see Listing 5-7 under MapVirtualKey.

GetKeyboardType Windows.pas

Syntax

GetKeyboardType(	
nTypeFlag: Integer	{type of information}
): Integer;	{returns the specified information}

Description

The GetKeyboardType function retrieves information about the keyboard depending on what type of data is requested. The type, subtype, and number of function keys may be obtained according to the state of the nTypeFlag parameter.

Parameters

nTypeFlag: Specifies the type of information to retrieve, such as keyboard type, subtype, or number of function keys. This parameter can be set to one value from Table 5-3. If the keyboard subtype is requested, the return value will be OEM specific with a meaning that is described in the subtype table (Table 5-4). If the number of function keys is requested, the return value is not the number of function keys but a code that is translated using the function key table (Table 5-5).

Return Value

If the function succeeds, it returns the requested information about the keyboard; otherwise, it returns zero.

See Also

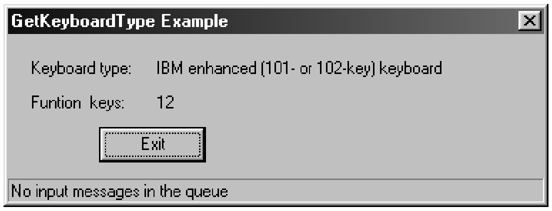
keybd_event

Example

Listing 5-2: Retrieving information about the keyboard

```
const
  KBType: array[0..6] of string = ('IBM® PC/XT® ( ) or compatible (83-key) keyboard',
                                   'Olivetti® "ICO" (102-key) keyboard',
                                   'IBM PC/AT® (84-key) or similar keyboard',
                                   'IBM enhanced (101- or 102-key) keyboard',
                                   'Nokia® 1050 and similar keyboards',
                                   'Nokia 9140 and similar keyboards',
                                   'Japanese keyboard');

procedure TForm1.FormActivate(Sender: TObject);
begin
  {display the keyboard type}
  Label3.Caption := KBType[GetKeyboardType(0) - 1];

  {display the number of function keys}
  Label4.Caption := IntToStr(GetKeyboardType(2));
```

```
end;
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  {check the state of the input queue}
  if (GetInputState = TRUE) then
    StatusBar1.SimpleText := 'Input messages in the queue'
  else
    StatusBar1.SimpleText := 'No input messages in the queue';
end;
```


GetKeyboardType Example

Keyboard type: IBM enhanced (101- or 102-key) keyboard

Funtion keys: 12

No input messages in the queue

Figure 5-2: The current keyboard type

Table 5-3: GetKeyboardType nTypeFlag values

Value	Description
0	Requesting keyboard type.
1	Requesting keyboard subtype.
2	Requesting number of function keys.

Table 5-4: GetKeyboardType subtype return values

Value	Description
1	IBM PC/XT or compatible keyboard (83 keys).
2	Olivetti “ICO” (102-key) keyboard.
3	IBM PC/AT (84-key) keyboard.
4	IBM enhanced (101- or 102-key) keyboard.
5	Nokia 1050 and similar keyboard.
6	Nokia 9140 and similar keyboard.
7	Japanese keyboard.

Table 5-5: GetKeyboardType function key count return values

Value	Description
1	10 function keys.
2	12 or 18 function keys.
3	10 function keys.
4	12 function keys.
5	10 function keys.
6	24 function keys.
7	Number of function keys is dependent on hardware and specified by the OEM.

GetKeyNameText Windows.pas

Syntax

```
GetKeyNameText(
lParam: Longint;        {lParam from the input message)
lpString: PChar;        {pointer to output buffer}
nSize: Integer          {maximum size of the buffer}
): Integer              {returns the size of data in the buffer}
```

Description

The GetKeyNameText function retrieves the name of the specified key and stores it in the buffer pointed to by the lpString parameter. The format of the key name depends on the current input locale identifier. On some keyboards, key names are longer than one character. The keyboard driver maintains the name list for the keys. Under Windows 95/98/Me, the key name is translated according to the currently loaded keyboard layout for the current thread. Under Windows NT/2000 and later, the key name is translated according to the currently loaded keyboard layout for the system.

Parameters

lParam: Specifies the lParam parameter of a keyboard message. This parameter contains information on the keystroke whose key name is to be retrieved and is interpreted as described in the following table.

lpString: Points to the output buffer that will receive the name of the key.

nSize: Specifies size of the output buffer pointed to by the lpString parameter.

Return Value

If the function succeeds, it returns the size of the string copied to the output buffer in characters, not including the null terminator. This function does not indicate an error upon failure.

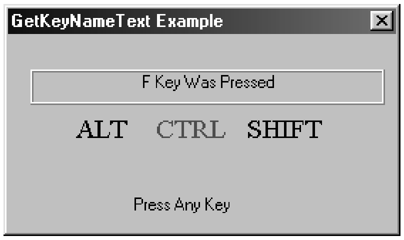
See Also

GetKeyState

Example

Listing 5-3: Retrieving keystroke names

```
procedure TForm1.WndProc(var Msg: TMessage);
var
  lpString: PChar;   // holds the key name
begin
  {if the message was a keystroke message...}
  if Msg.Msg = WM_KEYDOWN then
  begin
    {retrieve the name of the key pressed}
    lpString := StrAlloc(100);
    GetKeyNameText(Msg.LParam, lpString, 100);
    StaticText1.Caption := lpString  + ' Key Was Pressed';
    StrDispose(lpString);
    {indicate if the Shift key was pressed}
    if HiByte(GetKeyState(VK_SHIFT)) <> 0 then
      StaticText4.Font.Color := clRed
    else
      StaticText4.Font.Color := clBlack;

    {indicate if the Ctrl key was pressed}
    if HiByte(GetKeyState(VK_CONTROL)) <> 0 then
      StaticText3.Font.Color := clRed
    else
      StaticText3.Font.Color := clBlack;

    {indicate if the Alt key was pressed}
    if HiByte(GetKeyState(VK_MENU)) <> 0 then
      StaticText2.Font.Color := clRed
    else
      StaticText2.Font.Color := clBlack;
  end;

  {pass all messages to the window procedure}
  inherited WndProc(Msg);
end;

procedure TForm1.WMGetDlgCode(var Message: TWMGetDlgCode);
begin
  inherited;

  {this forces the system to send all keys to the form}
  Message.Result := Message.Result or DLGC_WANTALLKEYS or DLGC_WANTARROWS
    or DLGC_WANTTAB;
end;
```


GetKeyNameText Example

F Key Was Pressed

ALT CTRL SHIFT

Press Any Key

Figure 5-3: The keystroke name

Table 5-6: GetKeyNameText lParam values

Bits	Description
16-23	Specifies the key scan code.
24	Distinguishes extended key behavior on enhanced keyboard.
25	If this bit is set, the function does not differentiate between left and right Shift and Ctrl keys.

GetKeyState Windows.pas

Syntax

```
GetKeyState(
nVirtKey: Integer        {virtual key code}
): SHORT;                {returns the state}
```

Description

The GetKeyState function retrieves the status of the key specified by the nVirtKey parameter. The key state can be up, down, or toggled (i.e., Caps, Shift, or Ctrl).

Parameters

nVirtKey: Specifies the virtual key code for which to retrieve the status. The virtual key codes for keys from "A" to "Z" and "0" to "9" are the same as the ASCII value of the keys. See the following table.

Return Value

If the function succeeds, it returns the state of the specified key. If the high-order bit is set, the key is pressed. If the low-order bit is set, the key is toggled to an on state. This function does not indicate an error upon failure.

See Also

GetAsyncKeyState, GetKeyboardState, MapVirtualKey, SetKeyboardState

Example

Please see Listing 5-3 under GetKeyNameText.

Table 5-7: GetKeyState nVirtKey values

Value	Description
VK_F1–VK_F12	Function keys F1-F12
VK_NUMPAD0–VK_NUMPAD9	Numeric keypad 0-9 with NumLock on
VK_CANCEL	Ctrl-Break
VK_RETURN	Enter
VK_BACK	Backspace
VK_TAB	Tab
VK_CLEAR	Numeric keypad 5 with NumLock off
VK_SHIFT	Shift
VK_CONTROL	Ctrl
VK_MENU	Alt
VK_PAUSE	Pause
VK_ESCAPE	Esc
VK_SPACE	Spacebar
VK_PRIOR	Page Up and PgUp
VK_NEXT	Page Down and PgDn

Value	Description
VK_END	End
VK_HOME	Home
VK_LEFT	Left arrow
VK_UP	Up arrow
VK_RIGHT	Right arrow
VK_DOWN	Down arrow
VK_SNAPSHOT	Print Screen
VK_INSERT	Insert and Ins
VK_DELETE	Delete and Del
VK_MULTIPLY	Numeric keypad *
VK_ADD	Numeric keypad +
VK_SUBTRACT	Numeric keypad –
VK_DECIMAL	Numeric keypad .
VK_DIVIDE	Numeric keypad /
VK_CAPITAL	Caps Lock
VK_NUMLOCK	Num Lock
VK_SCROLL	Scroll Lock

keybd_event ***Windows.pas***

Syntax

```
keybd_event(
bVk: Byte;                {virtual key code}
bScan: Byte;              {scan code}
dwFlags: DWORD;           {option flags}
dwExtraInfo: DWORD        {additional information about the key}
);                        {this procedure does not return a value}
```

Description

The keybd_event function simulates a keystroke. The system generates a WM_KEYUP or WM_KEYDOWN message as if the key were pressed on the keyboard.

Parameters

bVk: The virtual key code in the range of 1-254. See GetKeyState for virtual keycode identifiers.

bScan: The hardware scan code for the key.

dwFlags: Flags identifying keystroke operations. This parameter can contain one or more values from the following table.

dwExtraInfo: Specifies an additional 32-bit value associated with the keystroke.

See Also

GetAsyncKeyState, GetKeyState, MapVirtualKey, SetKeyboardState

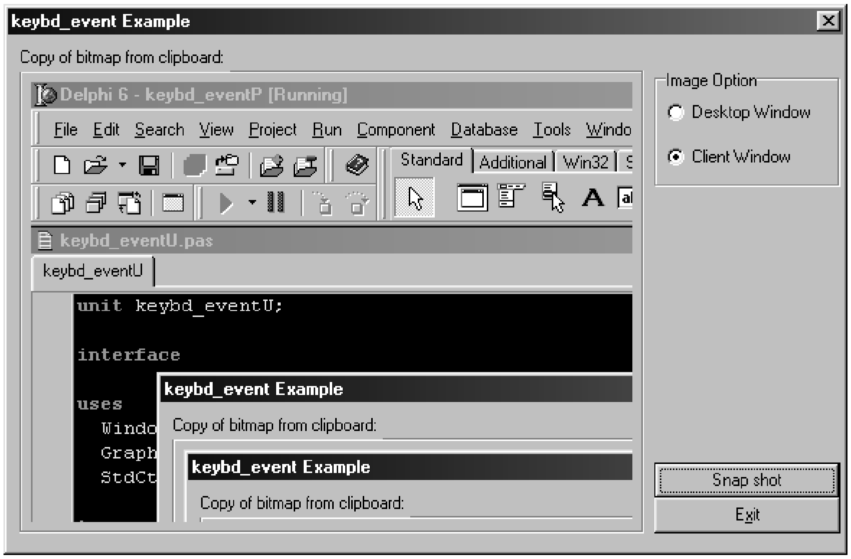
Example

Listing 5-4: Simulating the PRNTSCRN key using keybd_event

```
procedure TForm1.ButtonSnapShotClick(Sender: TObject);
var
  Bitmap: TBitmap;     // holds a bitmap
begin
  {see which radio button is checked}
  If ImageOptions.ItemIndex = 0
  then keybd_event(VK_SNAPSHOT,1,0,0)    {desktop window snapshot}
  else keybd_event(VK_SNAPSHOT,0,0,0);   {client window snapshot}

  {check to see if there is a picture}
  if Clipboard.HasFormat(CF_BITMAP) then
  begin
    {Create a bitmap to hold the contents of the clipboard}
    Bitmap := TBitmap.Create;

    {trap for clipboard bitmap errors}
    try
      {get the bitmap off the clipboard using Assign}
      Bitmap.Assign(Clipboard);

      {copy the bitmap to the Image}
      Image1.Canvas.Draw(0, 0, Bitmap);
    finally
      {the bitmap is no longer needed, so free it}
      Bitmap.Free;
    end;
  end;
end;
```

Table 5-8: keybd_event dwFlags values

Value	Description
KEYEVENTF_EXTENDEDKEY	If this flag is specified, then the scan code is prefixed with the byte value $E0 (224).
KEYEVENTF_KEYUP	If specified, the key is being released. If not, the key is being pressed.


keybd_event Example
Copy of bitmap from clipboard:
Delphi 6 - keybd_eventP [Running]
File Edit Search View Project Run Component Database Tools Windo
Standard Additional Win32
keybd_eventU.pas
keybd_eventU
keybd_event Example
Copy of bitmap from clipboard:
keybd_event Example
Copy of bitmap from clipboard:
Image Option
Desktop Window
Client Window
Snap shot
Exit

```
unit keybd_eventU;

interface

uses
  Windo
  Graph
  StdCt
```

Image Option: Desktop Window [✓ Client Window]

Figure 5-4: The simulated PRNTSCRN results

joyGetDevCaps *Mmsystem.pas*

Syntax

```
joyGetDevCaps(
uJoyID: UINT;                {joystick identifier}
lpCaps: PJoyCaps;            {points to TJoyCaps structure}
uSize: UINT                  {size of the TJoyCaps structure}
): MMRESULT;                 {returns an error condition}
```

Description

joyGetDevCaps retrieves the joystick capabilities into a TJoyCaps structure provided by the caller.

Parameters

uJoyID: A joystick identifier which can be JOYSTICKID1 or JOYSTICKID2.

lpCaps: Points to a TJoyCaps structure which receives the capabilities of the specified joystick. The TJoyCaps data structure is defined as:

```
TJoyCaps = record
    wMid: Word;                                   {manufacturer ID}
    wPid: Word;                                   {product ID}
    szPname: array[0..MAXPNAMELEN-1] of AnsiChar;    {product name}
    wXmin: UINT;                                  {minimum x position value}
    wXmax: UINT;                                  {maximum x position value}
    wYmin: UINT;                                  {minimum y position value}
    wYmax: UINT;                                  {maximum y position value}
    wZmin: UINT;                                  {maximum z position value}
```

```
        wZmax: UINT;                    {maximum z position value}
        wNumButtons: UINT;              {number of buttons}
        wPeriodMin: UINT;               {minimum message period when captured}
        wPeriodMax: UINT;               {maximum message period when captured}
        wRmin: UINT;                    {minimum r position value}
        wRmax: UINT;                    {maximum r position value}
        wUmin: UINT;                    {minimum u (fifth axis) position value}
        wUmax: UINT;                    {maximum u (fifth axis) position value}
        wVmin: UINT;                    {minimum v (sixth axis) position value}
        wVmax: UINT;                    {maximum v (sixth axis) position value}
        wCaps: UINT;                    {joystick capabilities}
        wMaxAxes: UINT;                 {maximum number of axes supported}
        wNumAxes: UINT;                 {number of axes in use}
        wMaxButtons: UINT;              {maximum number of buttons supported}
        szRegKey: array[0..MAXPNAMELEN – 1] of AnsiChar;  {registry key}
        szOEMVxD: array[0..MAX_JOYSTICKOEMVXDNAME – 1] of AnsiChar;
        {OEM VxD}
end;
```

wMid: Manufacturer's identifier.
wPid: Product identifier.
szPname: Name of the joystick as a null-terminated string.
wXmin: Minimum value of the joystick's x-coordinate.
wXmax: Maximum value of the joystick's x-coordinate.
wYmin: Minimum value of the joystick's y-coordinate.
wYmax: Maximum value of the joystick's y-coordinate.
wZmin: Minimum value of the joystick's z-coordinate.
wZmax: Maximum value of the joystick's z-coordinate.
wNumButtons: Number of buttons on the joystick.
wPeriodMin: Smallest polling frequency supported with joySetCapture on.
wPeriodMax: Largest polling frequency supported with joySetCapture on.
wRmin: Minimum rudder value (fourth axis).
wRmax: Maximum rudder value (fourth axis).
wUmin: Minimum value of fifth axis.
wUmax: Maximum value of fifth axis.
wVmin: Minimum value of sixth axis.
wVmax: Maximum value of sixth axis.
wCaps: Joystick capabilities as shown in Table 5-9.
wMaxAxes: Maximum number of axes supported.
wNumAxes: Number of axes in current use.
wMaxButtons: Number of buttons supported.
szRegKey: Joystick registry key as a null-terminated string.
szOEMVxD: Name of the OEM driver as a null-terminated string.

uSize: Specifies the size, in bytes, of the TJoyCaps structure.

Return Value

The function will return a success or failure result code as shown in Table 5-10.

See Also

joyGetPos, joyGetPosEx

Example

Please see Listing 5-5 under joySetCapture.

Table 5-9: joyGetDevCaps lpCaps.wCaps values

Value	Description
JOYCAPS_HASZ	Joystick has z-coordinate information.
JOYCAPS_HASR	Joystick has fouth axis information.
JOYCAPS_HASU	Joystick has fifth axis information.
JOYCAPS_HASV	Joystick has sixth axis information.
JOYCAPS_HASPOV	Joystick has point-of-view information.
JOYCAPS_POV4DIR	Joystick point of view supports discrete values for centered, forward, backward, left, and right.
JOYCAPS_POVCTS	Joystick point of view supports continuous degree bearings.

Table 5-10: joyGetDevCaps return values

Value	Description
JOYERR_NOERROR	The function succeeded.
MMSYSERR_NODRIVER	Joystick driver is not present.
MMSYSERR_INVALPARAM	Invalid parameter is passed.

joyGetNumDevs Mmsystem.pas

Syntax

```
joyGetNumDevs: UINT;      {returns the number of joysticks supported by the driver}
```

Description

The joyGetNumDevs retrieves the number of joysticks supported by the current joystick driver. Use joyGetPos to determine if a joystick is attached to the system.

Return Value

If the function succeeds, it returns the number of joysticks supported by the current joystick driver. If the function fails, or there is no joystick driver present, it returns zero.

See Also

joyGetDevCaps

Example

Please see Listing 5-5 under joySetCapture.

joyGetPos Mmsystem.pas

Syntax

```
joyGetPos(
uJoyID: UINT;              {joystick identifier}
lpInfo: PJoyInfo           {points to TJoyInfo structure}
): MMRESULT;               {returns an error condition}
```

Description

The joyGetPos function retrieves information about joystick position and button status for the joystick identified by the uJoyID parameter. Position and button status are stored in a TJoyInfo structure. This function can be used to determine if the joystick is currently attached to the system by checking the return value.

Parameters

uJoyID: The joystick identifier of the joystick whose position is to be checked. This parameter can be set to JOYSTICKID1 or JOYSTICKID2.

lpInfo: A pointer to a TJoyInfo structure that receives the joystick position information. The TJoyInfo data structure is defined as:

```
TJoyInfo = record
      wXpos: UINT;         {x position}
      wYpos: UINT;         {y position}
      wZpos: UINT;         {z position}
      wButtons: UINT;      {button states}
end;
```

wXpos: The current X position of the joystick.
wYpos: The current Y position of the joystick.
wZpos: The current Z position of the joystick.
wButtons: Status of the buttons as shown in Table 5-11.

Return Value

The function will return a success or failure result code as shown in Table 5-12.

See Also

joyGetPosEx

Example

Please see Listing 5-5 under joySetCapture.

Table 5-11: joyGetPos lpInfo.wButtons values

Value	Description
JOY_BUTTON1	First joystick button is pressed.
JOY_BUTTON2	Second joystick button is pressed.
JOY_BUTTON3	Third joystick button is pressed.
JOY_BUTTON4	Fourth joystick button is pressed.

Table 5-12: joyGetPos return values

Value	Description
JOYERR_NOERROR	The function succeeded.
MMSYSERR_NODRIVER	Joystick driver not found.
MMSYSERR_INVALPARAM	Invalid parameter.
JOYERR_UNPLUGGED	Joystick is unplugged.

joyGetPosEx Mmsystem.pas

Syntax

```
joyGetPosEx(
uJoyID: UINT;               {joystick identifier}
lpInfo: PJoyInfoEx          {points to TJoyInfoEx structure}
): MMRESULT;                {returns an error condition}
```

Description

The joyGetPosEx function retrieves information about joystick position and button status for the joystick identified by the uJoyID parameter. Position and button status are stored in a TJoyInfoEx structure. This function provides more information about the joystick position than the joyGetPos function.

Parameters

uJoyID: The joystick identifier of the joystick whose position is to be checked. This parameter can be set to JOYSTICKID1 or JOYSTICKID2.

lpInfo: A pointer to a TJoyInfoEx structure that receives the joystick position information. The TJoyInfoEx data structure is defined as:

```
TJoyInfoEx = record
      dwSize: DWORD;                {size of structure}
      dwFlags: DWORD;               {flags indicating what to return}
      wXpos: UINT;                  {x (first axis) position}
      wYpos: UINT;                  {y (second axis) position}
      wZpos: UINT;                  {z (third axis) position}
      dwRpos: DWORD;                {fourth axis position}
      dwUpos: DWORD;                {fifth axis position}
      dwVpos: DWORD;                {sixth axis position}
      wButtons: UINT;               {button states}
      dwButtonNumber: DWORD;        {current button number pressed}
```

```
        dwPOV: DWORD;                    {point-of-view state}
        dwReserved1: DWORD;              {reserved for system communication with
                                          joystick driver}
        dwReserved2: DWORD;              {reserved for future use}
    end;
```

dwSize: The size of this structure in bytes. This member should be set to SizeOf(TJoyInfoEx).

dwFlags: Option specifying which data is requested as shown in Table 5-13.

wXpos: Current first axis coordinate.

wYpos: Current second axis coordinate.

wZpos: Current third axis coordinate.

dwRpos: Current fourth axis coordinate.

dwUpos: Current fifth axis coordinate.

dwVpos: Current sixth axis coordinate.

wButtons: The current state of all 32 buttons supported by the system. Each button has an identifier (JOY_BUTTON1 through JOY_BUTTON32) that is simply an identifier for the bit positions in the 32-bit wButtons value. If the specified bit is set, the button is pressed.

dwButtonNumber: The current button number that is pressed.

dwPOV: The current position of the point-of-view control, in hundredths of degrees. This value can range from 0 to 35,900. When the JOY_RETURNPOV flag is set in the dwFlags entry, the value of dwPOV will be one of the values from the dwPOV table (Table 5-14). An application that supports only the point-of-view values shown in the table must have the JOY_RETURNPOV flag set. If the application can accept the variable degree information, it should set the JOY_RETURNPOVCTS flag, which also supports the JOY_POV constants in the dwPOV table.

dwReserved1: Reserved for future use.

dwReserved2: Reserved for future use.

Return Value

The function will return a success or failure result code as shown in Table 5-15.

See Also

joyGetDevCaps, joyGetPos

Example

Please see Listing 5-5 under joySetCapture.

Table 5-13: joyGetPosEx lpInfo.dwFlags values

Value	Description
JOY_RETURNALL	Equivalent to setting all of the JOY_RETURN bits, except for the JOY_RETURNRAWDATA bit.
JOY_RETURNBUTTONS	dwButtons contains information about each joystick button.
JOY_RETURNCENTERED	Centers the joystick neutral position to the central value of each axis.
JOY_RETURNPOV	The dwPOV member contains information about the point-of-view control, expressed in whole degrees.
JOY_RETURNPOVCTS	The dwPOV member contains valid information about the point-of-view control expressed in continuous, one-hundredth degree units.
JOY_RETURNR	The dwRpos member contains valid rudder pedal data for the fourth axis.
JOY_RETURNRAWDATA	Data stored in this structure contains uncalibrated joystick readings.
JOY_RETURNU	The dwUpos member contains valid data for a fifth axis.
JOY_RETURNV	The dwVpos member contains valid data for a sixth axis.
JOY_RETURNX	The dwXpos member contains valid data for the x-coordinate (first axis) of the joystick.
JOY_RETURNY	The dwYpos member contains valid data for the y-coordinate (second axis) of the joystick.
JOY_RETURNZ	The dwZpos member contains valid data for the z-coordinate (third axis) of the joystick.
JOY_USEDEADZONE	Expands the range for the neutral position of the joystick as a dead zone. Coordinate information is the same for all positions in the dead zone.
The following flags are intended for use in applications requiring custom calibration.	
JOY_CAL_READALWAYS	Reads the joystick port even if the driver does not detect a connected device.
JOY_CAL_READXYONLY	Reads the X and Y raw position values, placing them in the wXpos and wYpos members, respectively.
JOY_CAL_READ3	Reads the X, Y, and Z raw position values, placing them in the wXpos, wYpos, and wZpos members, respectively.
JOY_CAL_READ4	Reads the X, Y, Z, and rudder raw position values, placing them in the wXpos, wYpos, wZpos, and dwRPos members, respectively.
JOY_CAL_READXONLY	Reads the X raw position value, placing it in the wXpos member.
JOY_CAL_READYONLY	Reads the Y raw position value, placing it in the wYpos member.
JOY_CAL_READ5	Reads the X, Y, Z, rudder, and fifth axis raw position values, placing them in the wXpos, wYpos, wZpos, dwRPos, and dwUpos members, respectively.

Value	Description
JOY_CAL_READ6	Reads the X, Y, Z, rudder, fifth, and sixth axis raw position values, placing them in the wXpos, wYpos, wZpos, dwRPos, dwUpos, and dwVpos members, respectively.
JOY_CAL_READZONLY	Reads the Z raw position value, placing it in the wZpos member.
JOY_CAL_READRONLY	Reads the rudder raw position value, placing it in the dwRpos member.
JOY_CAL_READUONLY	Reads the fifth axis raw position value, placing it in the dwUpos member.
JOY_CAL_READVONLY	Reads the sixth raw position value, placing it in the dwVpos member.

Table 5-14: joyGetPosEx lpInfo.dwPOV values

Value	Description
JOY_POVBACKWARD	Point-of-view hat is pressed backward. The value 18,000 represents 180.00 degrees (to the rear).
JOY_POVCENTERED	Point-of-view hat is in the neutral position. The value –1 means there is no angle to report.
JOY_POVFORWARD	Point-of-view hat is pressed forward. The value 0 represents 0.00 degrees (straight ahead).
JOY_POVLEFT	Point-of-view hat is being pressed to the left. The value 27,000 represents 270.00 degrees (90.00 degrees to the left).
JOY_POVRIGHT	Point-of-view hat is pressed to the right. The value 9,000 represents 90.00 degrees (to the right).

Table 5-15: joyGetPosEx return values

Value	Description
JOYERR_NOERROR	The function succeeded.
MMSYSERR_NODRIVER	Joystick driver is not present.
MMSYSERR_INVALPARAM	Invalid parameter is passed.
MMSYSERR_BADDEVICEID	Joystick identifier is invalid.
JOYERR_UNPLUGGED	Joystick is unplugged.

joyGetThreshold Mmsystem.pas

Syntax

```
joyGetThreshold(
uJoyID: UINT;                {joystick identifier}
lpuThreshold: PUINT          {points to joystick threshold value}
): MMRESULT;                 {returns an error code}
```

Description

The joyGetThreshold function retrieves the joystick movement threshold. The threshold is the distance that the joystick must be moved before the driver sends a WM_JOYMOVE message.

Parameters

uJoyID: The joystick identifier of the joystick whose threshold is to be retrieved. This parameter can be set to JOYSTICKID1 or JOYSTICKID2.

lpuThreshold: A pointer to an integer receiving the joystick threshold value.

Return Value

The function will return a success or failure result code as shown in the table below.

See Also

joySetThreshold

Example

Please see Listing 5-5 under joySetCapture.

Table 5-16: joyGetThreshold return values

Value	Description
JOYERR_NOERROR	The function succeeded.
MMSYSERR_NODRIVER	Joystick driver is not present.
MMSYSERR_INVALPARAM	Invalid parameter is passed.

joyReleaseCapture   Mmsystem.pas

Syntax

```
joyReleaseCapture(
uJoyID: UINT                {joystick identifier}
): MMRESULT;                {returns an error code}
```

Description

The joyReleaseCapture function releases the captured joystick.

Parameters

uJoyID: The joystick identifier of the joystick to be released. This parameter can be set to JOYSTICKID1 or JOYSTICKID2.

Return Value

The function will return a success or failure result code as shown in the table below.

See Also

joySetCapture

Example

Please see Listing 5-5 under joySetCapture.

Table 5-17: joyReleaseCapture return values

Value	Description
JOYERR_NOERROR	The function succeeded.
MMSYSERR_NODRIVER	The joystick driver was not found.
JOYERR_PARMS	The specified joystick uJoyID is invalid.

joySetCapture Mmsystem.pas

Syntax

```
joySetCapture(
Handle: HWND;           {identifies the window to be captured}
uJoyID: UINT;           {joystick identifier}
uPeriod: UINT;          {polling frequency}
bChanged: BOOL          {change flag for message frequency}
): MMRESULT;            {returns an error code}
```

Description

The joySetCapture function captures the messages generated by the joystick driver. Joystick messages will be sent to the window specified by the Handle parameter. The function fails if the joystick is already captured. joyReleaseCapture may be used to release the capture before calling joySetCapture. The joystick is automatically released if the capture window is destroyed.

Parameters

Handle: Identifies the window that receives the joystick messages.

uJoyID: The joystick identifier of the joystick to be captured. This parameter can be set to JOYSTICKID1 or JOYSTICKID2.

uPeriod: The polling frequency in milliseconds.

bChanged: Indicates when messages are to be sent to the capture window. If this parameter is set to TRUE, messages are to be sent to the capture window only when the unreported motion exceeds the threshold value. If this parameter is set to FALSE, the messages are to be sent to the capture window when the polling interval has passed.

Return Value

The function will return a success or failure result code as shown in Table 5-18.

See Also

joyReleaseCapture

Example

Listing 5-5: Joystick motion in Delphi

```
var
  Form1: TForm1;
  Threshold: Integer;         // holds the joystick threshold value

implementation

procedure TForm1.WndProc(var Msg: TMessage);
var
   Cpoint: TPoint;           // holds the joystick position coordinates
begin
  {if the joystick has moved...}
  if Msg.Msg = MM_JOY1MOVE then
  begin
    {retrieve the coordinates relative to the panel}
    Cpoint.X := Msg.LParamLo;
    Cpoint.Y := Msg.LParamHI;
    JoyToClient(Cpoint);

    {modify the Smiley picture based on the position of the joystick}
    if ((Cpoint.x >= 50) and (Cpoint.x <= 55)) and
       ((Cpoint.y >= 40) and (Cpoint.y <= 45)) then
      Image2.Picture.Bitmap.Canvas.CopyRect(Rect(0,0,105,85),
        Image4.Picture.Bitmap.Canvas, Rect(0,0,105,85))
    else
      Image2.Picture.Bitmap.Canvas.CopyRect(Rect(0,0,105,85),
        Image3.Picture.Bitmap.Canvas, Rect(0,0,105,85));

    {draw the crosshair}
    Image2.Picture.Bitmap.Canvas.Pen.Color := clRed;
    Image2.Picture.Bitmap.Canvas.Pen.Width := 2;
    Image2.Picture.Bitmap.Canvas.MoveTo(Cpoint.X - 8 ,Cpoint.Y);
    Image2.Picture.Bitmap.Canvas.LineTo(Cpoint.X + 8 ,Cpoint.Y);
    Image2.Picture.Bitmap.Canvas.MoveTo(Cpoint.X ,Cpoint.Y - 8);
    Image2.Picture.Bitmap.Canvas.LineTo(Cpoint.X ,Cpoint.Y + 8);
    Image2.Picture.Bitmap.Canvas.Ellipse(Cpoint.X - 4 ,Cpoint.Y -4,
      Cpoint.X + 4 ,Cpoint.Y +4);
  end;

  {if a joystick button was pressed...}
  if Msg.Msg = MM_JOY1BUTTONDOWN then
  begin
    {color in a shape depending on which button was pressed}
    if Boolean(Msg.WParam and JOY_BUTTON1) then
      Shape1.Brush.Color := clRed;
    if Boolean(Msg.WParam and JOY_BUTTON2) then
      Shape2.Brush.Color := clRed;
  end;

  {if a joystick button was released...}
  if Msg.Msg = MM_JOY1BUTTONUP then
  begin
    {refill the shape with its original color}
    if not Boolean(Msg.WParam and JOY_BUTTON1) then
      Shape1.Brush.Color := clMaroon;
    if not Boolean(Msg.WParam and JOY_BUTTON2) then
      Shape2.Brush.Color := clMaroon;
  end;

  {send the messages on to the default message handler}
  inherited WndProc(Msg);
end;

procedure TForm1.joyInit;
var
  lpjoyInfoEx: TJOYINFOEX;    // holds extended joystick information
  lpjoyInfo: TJOYINFO;        // holds joystick information
  NumOfDevs: Integer;         // holds the number of joystick devices
  Dev1: Integer;              // holds joystick position return values
begin
  {get joystick threshold}
  joyGetThreshold(JOYSTICKID1, @Threshold);

  {get number of joystick}
  NumofDevs := joyGetNumDevs;

  {if there are no joystick devices present, indicate an error}
  if  NumOfDevs = 0  then
  begin
    MessageBox(Form1.Handle, 'Joystick driver not present', 'Error',
               MB_OK or MB_ICONWARNING);
    Exit;
  end;

  {determine if there is a joystick present}
  Dev1 := joyGetPosEx(JOYSTICKID1, @lpjoyInfoEx);
  if Dev1 = MMSYSERR_BADDEVICEID then
    MessageBox(Form1.Handle,'Joystick 1 is not present', 'Error ', MB_OK);

  {determine if the joystick is unplugged}
  Dev1 := joyGetPos(JOYSTICKID1, @lpjoyInfo);
  if Dev1 = JOYERR_UNPLUGGED then
    MessageBox(Form1.Handle,'Joystick is unplugged', 'Error ', MB_OK);

  {set the joystick threshold}
  joySetThreshold(JOYSTICKID1, 125);
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
  {capture joystick messages}
  if (joySetCapture(Form1.Handle, JOYSTICKID1, 0, TRUE) <> JOYERR_NOERROR ) then
  begin
    {indicate that there was a problem capturing the joystick}
    MessageBox(Form1.Handle,'Joystick is not captured', 'Error',
               MB_OK or MB_ICONWARNING);
    Close;
  end;
end;
```

```
{convert joystick coordinates to client coordinates}
procedure TForm1.joyToClient(var pptJoyPos: TPoint);
var
  JCaps: TJoyCaps;    // holds joystick device capabilities
  CRect: TRect;       // holds window coordinates
begin
  {get joystick capabilities}
  if (joyGetDevCaps(JOYSTICKID1, @JCaps, SizeOf(TJOYCAPS))<>JOYERR_NOERROR) then
    Exit;

  {set the joystick position relative to the panel}
  Windows.GetClientRect(Panel1.Handle, CRect);
  pptJoyPos.X := TRUNC((Panel1.Width - 1) * (pptJoyPos.X - JCaps.wXmin) /
    (JCaps.wXmax - JCaps.wXmin));
  pptJoyPos.Y := TRUNC((Panel1.Height - 1) * (pptJoyPos.Y - JCaps.wYmin) /
    (JCaps.wYmax - JCaps.wYmin));
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  {initialize joystick}
  Application.ProcessMessages;
  JoyInit;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  {release joystick capture}
  joyReleaseCapture(JOYSTICKID1);
end;

procedure TForm1.BitBtn1Click2(Sender: TObject);
begin
  Close;
end;
```

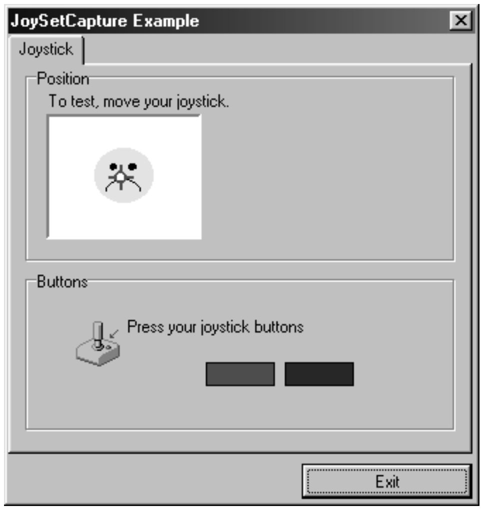


Figure 5-5: Smiley experiences Delphi's joystick capabilities

Table 5-18: joySetCapture return values

Value	Description
JOYERR_NOERROR	The function succeeded.
JOYERR_NOCANDO	The capture cannot take place because of a system conflict, such as a timer not being available for polling.
JOYERR_UNPLUGGED	Joystick is unplugged.

joySetThreshold *Mmsystem.pas*

Syntax

```
joySetThreshold(
uJoyID: UINT;               {joystick identifier}
uThreshold: UINT            {joystick threshold}
): MMRESULT;                {returns an error code}
```

Description

The joySetThreshold function sets the movement threshold for the joystick. The distance the joystick axis has to move before a WM_JOYMOVE message is generated defines the joystick threshold.

Parameters

uJoyID: The joystick identifier of the joystick whose threshold is to be set. This parameter can be set to JOYSTICKID1 or JOYSTICKID2.

uThreshold: Specifies the new threshold movement value.

Return Value

The function will return a success or failure result code as shown in the table below.

See Also

joySetCapture

Example

Please see Listing 5-5 under joySetCapture.

Table 5-19: joySetThreshold return values

Value	Description
JOYERR_NOERROR	The function succeeded.
MMSYSERR_NODRIVER	Joystick driver is not present.
JOYERR_PARMS	Joystick identifier (uJoyID) is invalid.

LoadKeyboardLayout Windows.pas

Syntax

LoadKeyboardLayout(	
pwszKLID: PChar;	{input locale identifier}
Flags: UINT	{layout options}
): HKL;	{returns a keyboard layout handle}

Description

LoadKeyboardLayout loads the specified input locale identifier (which includes a keyboard layout) into the system. Several input locale identifiers may be loaded simultaneously, but only one will be active at a time.

Parameters

pwszKLID: A pointer to a null-terminated string containing the name of the input locale identifier. This null-terminated string is the hexadecimal value of the layout ID. See the example for how the primary language identifier and sublanguage identifier are combined for the language ID.

Flags: Specifies how the keyboard layout is to be loaded. This parameter can contain one value from Table 5-20.

Return Value

If the function succeeds, it returns the handle of the requested keyboard layout that was loaded. If the function failed, or if no matching keyboard layout was found, it returns zero. To get extended error information, call the GetLastError function.

See Also

ActivateKeyboardLayout, GetKeyboardLayoutName, UnloadKeyboardLayout

Example

Listing 5-6: Loading a keyboard layout

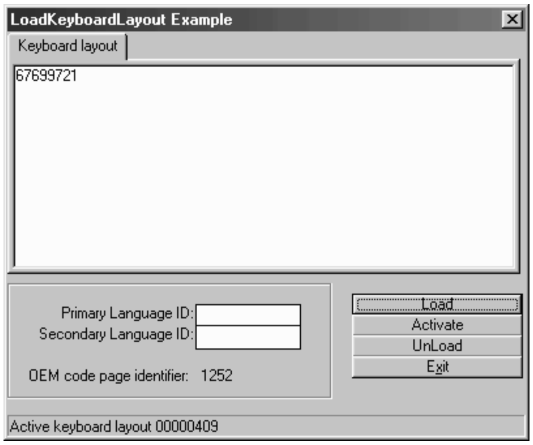
```
var
  Form1: TForm1;
  List : array [0..MAX_HKL] of HKL;    // list of keyboard handles

const
  {Delphi 6 does not define all of the available ActivateKeyboardLayout flags}
  KLF_SETFORPROCESS   = $00000100;
  KLF_SHIFTLOCK       = $00010000;
  KLF_RESET           = $40000000;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var
  MyLangID: WORD;  // holds a language identifier
```

```
begin
  {load the keyboard layout specified by language IDs}
  MyLangID:=MakeLangID(WORD(StrToInt(Edit1.Text)), WORD(StrToInt(Edit2.Text)));
  if LoadKeyboardLayout(PChar('0000' + IntToHex(MyLangID,4)),KLF_ACTIVATE) = 0
  then ShowMessage('Error loading keyboard layout');
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  {activate the highlighted keyboard layout}
  if (ActivateKeyboardLayout(StrToInt(ListBox1.Items[Listbox1.Itemindex]),
      KLF_REORDER) = 0) then
    ShowMessage('Error activating the keyboard layout');

  {clear the keyboard layout list and repopulate it}
  ListBox1.Clear;
  FormCreate(Sender);
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  pwszKLID: PChar;          // holds the name of a keyboard layout
  MyListIndex: Integer;     // specifies a list index
begin
  {get the keyboard layout lists}
  GetKeyboardLayoutList(MAX_HKL, List);

  {allocate a buffer for the keyboard layout name string}
  GetMem(pwszKLID, KL_NAMELENGTH);

  {retrieve the name string for active keyboard layout}
  GetKeyboardLayoutName(pwszKLID);
  ShowMessage('The active keyboard layout is '+pwszKLID);
  StatusBar1.SimpleText:= 'Active keyboard layout ' +  pwszKLID;

  {retrieve the code page identifier}
  StaticText1.Caption:=IntTostr(GetACP);

  {free the string memory}
  FreeMem(pwszKLID);

  {list all the keyboard layouts in the list box}
  MyListIndex := 0;
  While (List[MyListIndex] <> 0) do
  begin
    ListBox1.Items.Add(IntToStr(List[MyListIndex]));
    Inc(MyListIndex);
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  {unload keyboard layout}
  if not UnloadKeyboardLayout(StrToInt(ListBox1.Items[Listbox1.ItemIndex])) then
    ShowMessage('Error Unloading Keyboard Layout');
end;
```

```
function MakeLangID(PrimLang, SubLang:WORD): WORD;
begin
  {make a Language ID by combining the Primary language ID and Sub language ID}
  Result := (SubLang SHL 10) + PrimLang;
end;
```

Table 5-20: LoadKeyboardLayout flags values

Value	Description
KLF_ACTIVATE	The function will load the layout if not already loaded and set it to the currently active layout. If it is already loaded and the KLF_REORDER flag is not specified, the function will rotate the keyboard layout list and set the next layout as the active layout.
KLF_NOTELLSHELL	Prevents a ShellProc hook from receiving an HSHELL_LANGUAGE message until the entire list of layouts is loaded.
KLF_REORDER	This will make the given layout the active layout by rotating the internal layout list when more than one keyboard layout is loaded.
KLF_REPLACELANG	Loads the new layout if it is the same language as the currently active keyboard layout. If this flag is not set and the requested layout has the same language as the active layout, the new keyboard layout is not loaded and the function returns zero.
KLF_SUBSTITUTE_OK	Specifies that the substitute layout is loaded from the system registry under the key HKEY_CURRENT_USER\Keyboard Layout\Substitutes. For example, if the key indicates the value name "00000409" with value "00010409," it loads the Dvorak U.S. English layout.
KLF_SETFORPROCESS	**Windows 2000 and later**: Activates the indicated locale identifier (and physical keyboard layout) for the entire process. The current thread's active window receives a WM_INPUTLANGCHANGE message. Must be used with the KLF_ACTIVATE flag.


LoadKeyboardLayout Example

67699721

Primary Language ID:
Secondary Language ID:
OEM code page identifier: 1252

Active keyboard layout 00000409

Figure 5-6: The keyboard layouts

MapVirtualKey Windows.pas

Syntax

```
MapVirtualKey(
uCode: UINT;            {key code, scan code, or virtual key}
uMapType: UINT          {flags for translation mode}
): UINT;                {returns translated key code}
```

Description

The MapVirtualKey function converts a virtual key code to a scan code or character value, or it converts a scan code into a virtual key code. The uMapType parameter determines which conversion is performed.

Parameters

uCode: The key code which can be a virtual key code or scan code. How this value is interpreted depends on the translation mode flag specified in the uMapType parameter.

uMapType: Specifies a translation mode. This parameter can contain one value from the following table.

Return Value

If the function succeeds, it returns a scan code, virtual key code, or character value, depending on its parameters. If the function fails, or there is no translation, it returns zero.

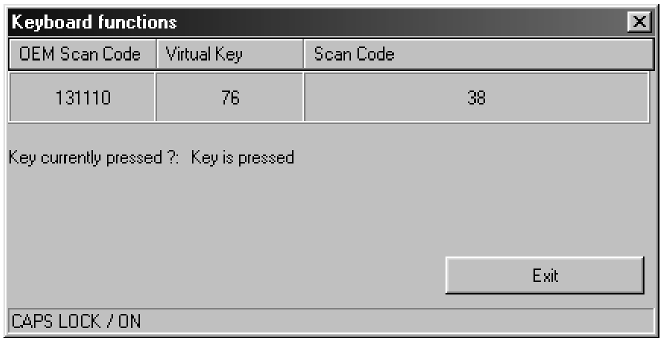
See Also

GetAsyncKeyState, GetKeyboardState, GetKeyState, SetKeyboardState

Example

Listing 5-7: Using MapVirtualKey to translate keyboard characters

```
var
  Form1: TForm1;
  Key_Value: Word;      // holds a virtual key code

implementation

{$R *.DFM}

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;Shift: TShiftState);
begin
  {store the pressed virtual key value}
  Key_Value := Key;

  {display the OEM scan code}
  Panel1.Caption:=IntToStr(OEMKeyScan(Key));

  {display the virtual key code for the lowercase form of the keystroke}
  Panel2.Caption:=IntToStr(MapVirtualKey(Key,2));
  {display the scan code of the pressed key}
  Panel3.Caption:=IntToStr(MapVirtualKey(VkKeyScan(Char(Key)),0));
end;

procedure TForm1.GetCapsKeyState(sender: TObject);
var
  KeyState: TKeyboardState;       // holds the keyboard state array
  Key_State: Word;                // holds the keypressed state
begin
  {retrieve the current state of the keyboard}
  GetKeyboardState(KeyState);

  {indicate if the caps lock key is on or off}
  if (KeyState[VK_CAPITAL] and $01) = $01 then
    StatusBar1.SimpleText := 'CAPS LOCK / ON'
  else
    StatusBar1.SimpleText:= 'CAPS LOCK / OFF';

  {indicate if a key is currently pressed}
  Key_State := GetAsyncKeyState(vkKeyScan(CHAR(Key_Value)));
  if Boolean(HiByte(Key_State)) then
    StaticText2.Caption := 'Key is pressed'
  else
    StaticText2.Caption := 'Key is not pressed';
end;

procedure TForm1.Timer2Timer(Sender: TObject);
var
  KeyState: TKeyboardState;       // holds the keyboard state array
begin
  {retrieve the current state of the keyboard}
  GetKeyboardState(KeyState);

  {toggle the CAPSLOCK key}
  if (KeyState[VK_CAPITAL] and $01) = $01 then
    KeyState[VK_CAPITAL] := 0
  else
    KeyState[VK_CAPITAL] := $81;

  {set the new state of the keyboard}
  SetKeyboardState(KeyState);
end;
```



OEM Scan Code	Virtual Key	Scan Code
131110	76	38

Key currently pressed ?: Key is pressed

CAPS LOCK / ON

Figure 5-7: The translated keyboard characters

Table 5-21: MapVirtualKey uMapType values

Value	Description
0	uCode is a virtual key code to be translated to a scan code. The value returned does not differentiate between left and right Ctrl and Shift keys; it only returns values for the left-hand control keys.
1	uCode is a scan code to be translated to a virtual key code. The value returned does not differentiate between left and right Ctrl and Shift keys; it only returns values for the left-hand control keys.
2	uCode is a virtual key code to be translated to an unshifted character value.
3	**Windows NT/2000 and later**: uCode is a scan code to be translated to a virtual key code. The value returned does differentiate between left and right Ctrl and Shift keys.

MapVirtualKeyEx Windows.pas

Syntax

```
MapVirtualKeyEx(
uCode: UINT;               {key code, scan code, or virtual key}
uMapType: UINT;            {flags for translation mode}
dwhkl: HKL                 {keyboard layout handle}
): UINT;                   {returns a translated key code}
```

Description

MapVirtualKeyEx converts a virtual key code to a scan code or character value, or it converts scan code into a virtual key code. The uMapType parameter determines which conversion is performed.

The difference between the MapVirtualKeyEx and MapVirtualKey functions is that MapVirtualKeyEx translates the character using the language of the physical keyboard layout, as specified by the keyboard layout handle in the dwhkl parameter. MapVirtualKeyEx will not translate a virtual key code to a scan code and distinguish between left and right keys, such as VK_SHIFT, VK_CONTROL, or VK_MENU. An application can get the proper scan code that distinguishes between left and right keys by setting the uCode parameter to VK_LSHIFT, VK_RSHIFT, VK_LCONTROL, VK_RCONTROL, VK_LMENU, or VK_RMENU.

Parameters

uCode: The virtual key code or a scan code to be translated. How this value is interpreted depends on the translation mode flag in the uMapType parameter.

uMapType: Specifies the translation mode. This parameter can contain one value from the following table.

dwhkl: Specifies the keyboard layout handle, which is used to translate characters into their corresponding virtual key codes. The keyboard layout handle can be obtained by calling the GetKeyboardLayout or LoadKeyboardLayout functions.

Return Value

If the function succeeds, it returns a scan code, virtual key code, or character value, depending on its parameters. If the function fails, or there is no translation, it returns zero.

See Also

GetAsyncKeyState, GetKeyboardState, GetKeyState, MapVirtualKey, SetKeyboardState

Example

Please see Listing 5-11 under VkKeyScanEx.

Table 5-22: MapVirtualKeyEx uMapType values

Value	Description
0	uCode is a virtual key code to be translated to a scan code. The value returned does not differentiate between left and right Ctrl and Shift keys; it only returns values for the left-hand control keys.
1	uCode is a scan code to be translated to a virtual key code. The value returned does not differentiate between left and right Ctrl and Shift keys; it only returns values for the left-hand control keys.
2	uCode is a virtual key code to be translated to an unshifted character value.
3	**Windows NT/2000 and later**: uCode is a scan code to be translated to a virtual key code. The value returned does differentiate between left and right Ctrl and Shift keys.

mouse_event Windows.pas

Syntax

```
mouse_event(
dwFlags: DWORD;          {mouse activity codes}
dx: DWORD;               {horizontal location or change}
dy: DWORD;               {vertical location or change}
dwData: DWORD;           {wheel movement amount}
dwExtraInfo: DWORD       {application-defined data}
);                       {this procedure does not return a value}
```

Description

The mouse_event function simulates mouse activity. The system generates mouse messages as if the mouse was actually moved or a mouse button was actually pressed.

Parameters

dwFlags: Specifies which kind of mouse activity to simulate. This parameter can contain one or more values from Table 5-23.

dx: Specifies the horizontal location or change in location. If the dwFlags parameter contains the MOUSEEVENTF_ABSOLUTE flag, this parameter specifies a location.

Otherwise, this parameter specifies the amount of mickeys (a measurement of mouse distance) to move.

dy: Specifies the vertical location or change in location. If the dwFlags parameter contains the MOUSEEVENTF_ABSOLUTE flag, this parameter specifies a location. Otherwise, this parameter specifies the amount of mickeys (a measurement of mouse distance) to move.

dwData: Specifies the amount of wheel movement if the dwFlags parameter contains the MOUSEEVENTF_WHEEL flag. A positive value indicates wheel movement away from the user; a negative value indicates wheel movement toward the user. This value is in terms of WHEEL_DELTA, approximately 120 mickeys. If dwFlags contains either MOUSEEVENTF_XDOWN or MOUSEEVENTF_XUP, this parameter specifies which X button was pressed or released and can contain a combination of values from Table 5-24. If the dwFlags parameter does not contain the MOUSEEVENTF_WHEEL, MOUSEEVENTF_XDOWN, or MOUSEEVENTF_XUP flags, dwData should be set to zero.

dwExtraInfo: 32-bits of additional application-defined data. To retrieve this data, call the GetMessageExtraInfo function.

See Also

GetMessageExtraInfo*, SystemParametersInfo

Example

Listing 5-8: Using mouse_event to control the mouse programmatically

```
var
  Form1: TForm1;
  MouseButtonIsDown: boolean;   // indicates if the mouse button is down

implementation

{$R *.DFM}

procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  {if the mouse button is down, draw a line}
  if MouseButtonIsDown then
    Image1.Canvas.LineTo(X,Y);
end;

procedure TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  {if the mouse button is not down, move the initial drawing position}
  if not MouseButtonIsDown then
    Image1.Canvas.MoveTo(X,Y);

  {indicate that the mouse button is down so that drawing will occur}
  MouseButtonIsDown := TRUE;

  {if the right mouse button was clicked...}
  if Button = MBRight then
  begin
    {...while the mouse button is held down...}
    while MouseButtonIsDown = TRUE do
    begin
      {...simulate mouse movement by the specified amounts. the image continues to receive
       regular mouse messages as if the mouse was under user control}
      mouse_event(MOUSEEVENTF_MOVE,SpinEdit1.Value,SpinEdit2.Value,0,0);

      {update the screen and pause for a short amount of time}
      Application.ProcessMessages;
      Sleep(10);
    end;
  end;
end;

procedure TForm1.Image1MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  {set the mouse button down variable to off}
  MouseButtonIsDown := FALSE;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  {initialize the initial drawing position}
  Image1.Canvas.MoveTo(10,10);
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  {initialize the mouse button down variable}
  MouseButtonIsDown := FALSE;
end;
```

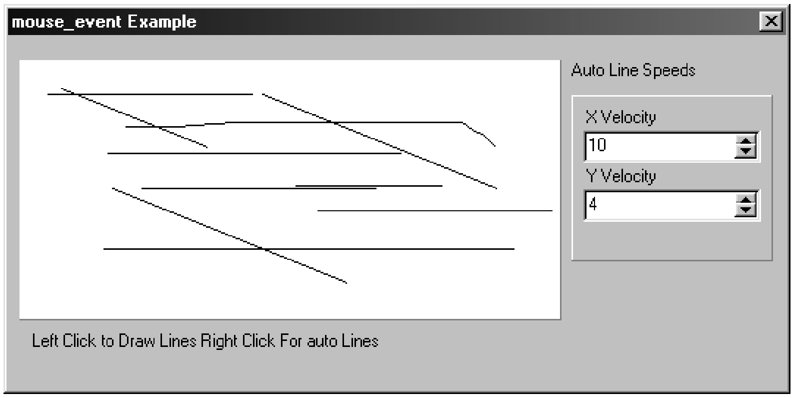


Figure 5-8: Drawing automatic lines with simulated mouse movement

Table 5-23: mouse_event dwFlags values

Value	Description
MOUSEEVENTF_ABSOLUTE	dx and dy contain normalized absolute coordinates. Otherwise, those parameters contain the change in position since the last reported position.
MOUSEEVENTF_MOVE	Movement occurred.
MOUSEEVENTF_LEFTDOWN	The left button changed to down.
MOUSEEVENTF_LEFTUP	The left button changed to up.
MOUSEEVENTF_RIGHTDOWN	The right button changed to down.
MOUSEEVENTF_RIGHTUP	The right button changed to up.
MOUSEEVENTF_MIDDLEDOWN	The middle button changed to down.
MOUSEEVENTF_MIDDLEUP	The middle button changed to up.
MOUSEEVENTF_WHEEL	**Windows NT/2000 and later**: The wheel has been moved, if the mouse has a wheel. The amount of movement is provided in the dwData parameter. Cannot be used in combination with MOUSEEVENT-F_XDOWN or MOUSEEVENTF_XUP.
MOUSEEVENTF_XDOWN	**Windows 2000/XP and later**: An X button was pressed. Cannot be used in combination with MOUSEEVENTF_WHEEL.
MOUSEEVENTF_XUP	**Windows 2000/XP and later**: An X button was released. Cannot be used in combination with MOUSEEVENTF_WHEEL.
MOUSEEVENTF_VIRTUALDESK	**Windows 2000/XP and later**: Map coordinates to entire virtual desktop.

Table 5-24: mouse_event dwData values

Value	Description
XBUTTON1	**Windows 2000/XP and later**: Indicates the first X button
XBUTTON2	**Windows 2000/XP and later**: Indicates the second X button.

OEMKeyScan Windows.pas

Syntax

```
OEMKeyScan(
wOemChar: Word            {ASCII value of OEM character}
): DWORD;                 {returns scan code data}
```

Description

This function retrieves the OEM scan code for the OEM ASCII character value (between $00 and $FF) and the state of the Shift, Ctrl, and Alt keys. OEMKeyScan works only for characters that can be produced with a single keystroke.

Parameters

wOemChar: Specifies the OEM ASCII character value whose OEM scan code is to be retrieved.

Return Value

If the function succeeds, the low-order byte of the return value contains the OEM scan code, and the high-order byte contains the status of Shift, Ctrl, and Alt keys as shown in the following table. If the function fails, it returns $FFFFFFFF.

See Also

MapVirtualKey, VkKeyScan

Example

Please see Listing 5-7 under MapVirtualKey.

Table 5-25: OEMKeyScan return values

Value	Description
1	The Shift key is pressed.
2	The Ctrl key is pressed.
4	The Alt key is pressed.

ReleaseCapture Windows.pas

Syntax

```
ReleaseCapture: BOOL;            {returns TRUE or FALSE}
```

Description

The ReleaseCapture function releases mouse capture by a window. The normal flow of mouse messages to the underlying window is restored.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE.

See Also

GetCapture

Example

Please see Listing 5-10 under SwapMouseButton.

SetCapture Windows.pas

Syntax

```
SetCapture(
hWnd: HWND           {handle of window capturing mouse messages}
): HWND;             {returns the previous capture handle}
```

Description

The SetCapture function captures mouse input messages and sends them to the window specified by the hWnd parameter. If the mouse has been captured, all of the mouse input is directed to the capturing window, even when the cursor is outside the boundary of that window. When the window no longer requires the mouse input, it should call the ReleaseCapture function.

Parameters

hWnd: Specifies the handle of the window that is to capture the mouse input.

Return Value

If the function succeeds, it returns the handle of the window that previously had the mouse capture. If the function fails, or no window previously had the mouse capture, it returns zero.

See Also

ReleaseCapture

Example

Please see Listing 5-10 under SwapMouseButton.

SetCaretBlinkTime Windows.pas

Syntax

```
SetCaretBlinkTime(
uMSeconds: UINT            {caret blink time in milliseconds }
): BOOL;                   {returns TRUE or FALSE}
```

Description

SetCaretBlinkTime changes the cursor blink rate to the time specified in the uMSeconds parameter.

Parameters

uMSeconds: Specifies the new caret blink interval in milliseconds.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetCaretBlinkTime

Example

Listing 5-9: Modifying caret position and blink time

```
procedure TForm1.SpinEdit1Change(Sender: TObject);
begin
  {change the caret blink rate to the spinedit value.}
  SetCaretBlinkTime(SpinEdit1.Value);

  {set focus back to the memo to demonstrate the blink rate}
  Memo1.SetFocus;
end;

procedure TForm1.SpinEdit1MouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin
  {display the caret blink time in the spinedit box}
  SpinEdit1.Value := GetCaretBlinkTime;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  lpPoint: TPoint;      // holds the current caret position
begin
  {retrieve the caret position}
  GetCaretPos(lpPoint);

  {display the caret position}
  SpinEdit2.Value := lpPoint.X;
  SpinEdit3.Value := lpPoint.Y;

  {make sure the caret remains in the Memo box at the specified position}
  Memo1.SetFocus;
  SetCaretPos(lpPoint.X, lpPoint.Y);
end;

procedure TForm1.SpinEdit2Change(Sender: TObject);
begin
  {change the caret position in the memo}
  Memo1.SetFocus;
  SetCaretPos(SpinEdit2.Value, SpinEdit3.Value);
end;
```

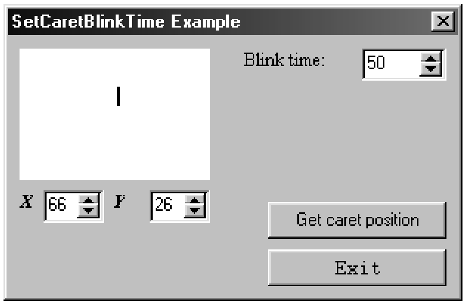


Figure 5-9: The caret position and blink time

SetCaretPos Windows.pas

Syntax

```
SetCaretPos(
X: Integer;              {X coordinate for new caret position}
Y: Integer               {Y coordinate for new caret position}
): BOOL;                 {returns TRUE or FALSE}
```

Description

The SetCaretPos function moves the caret to the coordinates specified by the X and Y parameters.

If the window's class style contains the CS_OWNDC style flag, the coordinates of the caret are mapped to the window's device context. This function will move the caret even if the cursor is hidden.

Parameters

X: Specifies the horizontal location of the new caret position.

Y: Specifies the vertical location of the new caret position.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetCaretPos

Example

Please see Listing 5-9 under SetCaretBlinkTime.

SetCursorPos Windows.pas

Syntax

```
SetCursorPos(
X: Integer;              {X coordinate of the cursor}
Y: Integer               {Y coordinate of the cursor}
): BOOL;                 {returns TRUE or FALSE}
```

Description

The SetCursorPos function relocates the mouse cursor to the location specified by the X and Y parameters in screen coordinates. If the cursor is confined to a rectangular region by calling the ClipCursor function, the system translates the coordinates to the appropriate coordinates within the rectangular region.

Parameters

X: Specifies the new x-coordinate for the cursor.

Y: Specifies the new y-coordinate for the cursor.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

ClipCursor, GetCursorPos, SetCaretPos

Example

Please see Listing 5-10 under SwapMouseButton.

SetDoubleClickTime   Windows.pas

Syntax

```
SetDoubleClickTime(
Interval: UINT              {interval between clicks in milliseconds}
): BOOL;                    {returns TRUE or FALSE}
```

Description

The SetDoubleClickTime function changes the time interval between the first and second mouse click defining a double-click.

Parameters

Interval: Specifies the new time interval between clicks in milliseconds.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetDoubleClickTime

Example

Please see Listing 5-10 under SwapMouseButton.

SetKeyboardState Windows.pas

Syntax

```
SetKeyboardState(
var KeyState: TKeyboardState      {array of the virtual key states}
): BOOL;                          {returns TRUE or FALSE}
```

Description

The SetKeyboardState function sets the status of all 256 virtual keys. The status of each virtual key is stored in an array of 256 bytes, identified by the KeyState parameter. Use the virtual key codes as an index into this array to specify individual virtual key states (i.e., KeyState[VK_SHIFT]).

Parameters

KeyState: Points to a TKeyboardState structure, which is an array of 256 bytes. Each index in the array should be set to a value indicating the state of individual virtual keys. If the high-order bit of an array value is 1, that key is pressed. If the low-order bit is 1, the key is toggled on, such as the Caps, Shift, or Alt keys. TKeyboardState is defined as follows:

```
TKeyboardState = array[0..255] of Byte;      {virtual key code states}
```

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetAsyncKeyState, GetKeyboardState, GetKeyState, MapVirtualKey

Example

Please see Listing 5-7 under MapVirtualKey.

SwapMouseButton Windows.pas

Syntax

```
SwapMouseButton(
fSwap: BOOL                 {reverse or restore mouse buttons flag}
): BOOL;                    {returns TRUE or FALSE}
```

Description

The SwapMouseButton function exchanges or restores the mouse button messages generated by the mouse buttons. If the buttons are swapped, the left mouse button will generate right mouse button messages (i.e., WM_RBUTTONDOWN), and the right mouse button will generate left mouse button messages.

Parameters

fSwap: If this parameter is set to TRUE, the mouse buttons are interchanged left for right. If this parameter is set to FALSE, the mouse buttons are restored to their original configuration.

Return Value

If the function succeeds and the mouse buttons were reversed previously, it returns TRUE. If the function fails, or the mouse buttons were not reversed previously, it returns FALSE.

See Also

SetDoubleClickTime

Example

Listing 5-10: Controlling mouse activity

```
var
  Form1: TForm1;
  SwapFlag: Boolean;       // tracks mouse button swapping
  glpRect: TRect;          // cursor coordinates

implementation

{$R *.DFM}

procedure TForm1.PanelClipRegionMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
Var
  lpWRect: TRect;      // holds window coordinates
  lpCPoint: TPoint;    // holds cursor coordinates
begin
  {retrieve the panel coordinates}
  GetWindowRect(PanelClipRegion.Handle, lpWRect);

  {retrieve the cursor position}
  GetCursorPos(lpCPoint);

  {display the cursor position in terms of the panel}
  Windows.ScreenToClient(PanelClipRegion.Handle,lpCPoint);
  EditXPos.Text:=IntToStr(lpCPoint.x);
  EditYPos.Text:=IntToStr(lpCPoint.y);

  {confine the cursor within the panel}
  ClipCursor(@lpWRect);
end;

procedure TForm1.ShapeMouseMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  {fill in the appropriate rectangle for the mouse button pressed}
  if Button = mbLeft then
    Shape2.Brush.Color := clRed
  else
  if Button = mbRight then
    Shape3.Brush.Color := clRed;
end;

procedure TForm1.ShapeMouseMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  {restore the rectangle's original color when the mouse button is released}
  Shape2.Brush.Color := clWhite;
  Shape3.Brush.Color := clWhite;
end;

procedure TForm1.ButtonSwapClick(Sender: TObject);
begin
  {toggle the mouse button swap flag}
  SwapFlag := not SwapFlag;

  {swap the mouse buttons}
  SwapMouseButton(SwapFlag);
end;

procedure TForm1.ButtonReleaseCursorClick(Sender: TObject);
var
  lpWRect: TPoint;         // holds mouse cursor coordinates
begin
  {set the mouse clipping region to the original region}
  ClipCursor(@glpRect);

  {move the mouse to the top left corner of the form}
  lpWRect.x:= Left;
  lpWRect.y:= Top;
  SetCursorPos(lpWRect.x,lpWRect.y);
end;

procedure TForm1.ButtonSetCursorPosClick(Sender: TObject);
begin
  {place the mouse cursor at the position indicated by the edit boxes.
   note that this position will be relative to the screen}
  if not SetCursorPos(StrToInt(EditXPos.Text), StrToInt(EditYPos.Text)) then
    ShowMessage('Error setting cursor position');
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  {save the original cursor bounds}
  GetClipCursor(glpRect);

  {get the double click time}
  SpinEditClickTime.Value := GetDoubleClickTime;
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  {indicate that the form received a mouse message (demonstrates mouse capture)}
  StatusBar1.SimpleText := 'The form received a mouse message';

  {bring the window to the top, if it lost focus}
  BringWindowToTop(Form1.Handle);
end;

procedure TForm1.ButtonMouseCaptureClick(Sender: TObject);
begin
  {make sure the mouse is not currently captured}
  ReleaseCapture;

  {capture the mouse and send all mouse messages to the form}
  if ((GetCapture = 0) and (SetCapture(Form1.Handle) <> 0))
    then ShowMessage('Error setting the mouse capture');
end;

procedure TForm1.BitBtnApplyClick(Sender: TObject);
begin
  {set the double click interval}
  if not SetDoubleClickTime(SpinEditClickTime.Value) then
    ShowMessage('Error setting the double click time');
end;


procedure TForm1.ButtonReleaseCaptureClick(Sender: TObject);
begin
  {release the mouse capture}
  if not ReleaseCapture then
    ShowMessage('Error releasing mouse capture');
end;

procedure TForm1.BitBtnExitClick(Sender: TObject);
begin
   Close;
end;
```

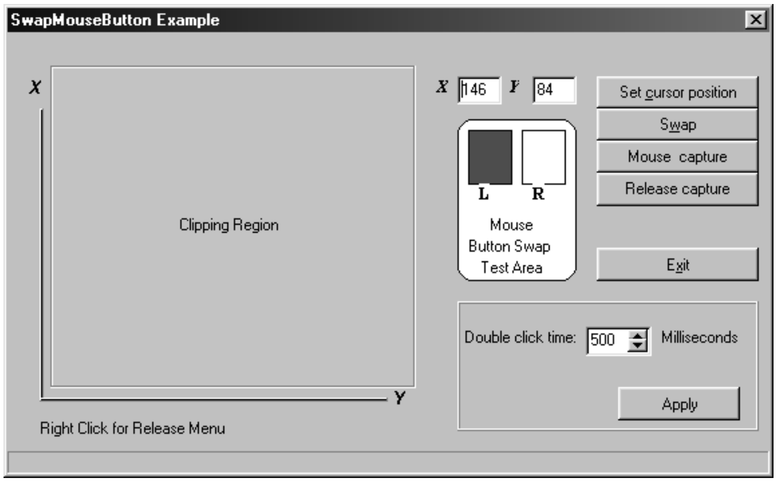


Figure 5-10: The mouse functions testbed

UnloadKeyboardLayout *Windows.pas*

Syntax

```
UnloadKeyboardLayout(
hkl: HKL                    {input locale identifier}
): BOOL;                    {returns TRUE or FALSE}
```

Description

UnloadKeyboardLayout removes the specified input locale identifier from the list of loaded input locale identifiers.

Parameters

hkl: Specifies the input locale identifier to unload.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

ActivateKeyboardLayout, GetKeyboardLayoutName, LoadKeyboardLayout

Example

Please see Listing 5-6 under LoadKeyboardLayout.

VkKeyScan *Windows.pas*

Syntax

```
VkKeyScan(
ch: Char                    {ASCII character code}
): SHORT;                   {returns a translated code}
```

Description

The VkKeyScan function translates the specified character code to a virtual key code and returns the status of Shift, Ctrl, and Alt keys. Numeric keypad keys are not translated.

Parameters

ch: Specifies the character value of the key. This value is translated into a virtual key code.

Return Value

If the function succeeds, the low-order byte of the return value contains the virtual key code, and the high-order byte contains a code specifying the state of the Shift, Ctrl, and Alt keys. See the table below for the high-order byte codes. If the function fails, both the high- and low-order bytes contain –1.

See Also

GetAsyncKeyState, GetKeyboardState, GetKeyNameText, MapVirtualKey, VkKeyScanEx

Example

Please see Listing 5-7 under MapVirtualKey.

Table 5-26: VkKeyScan return values

Value	Description
1	The Shift key is pressed.
2	The Ctrl key is pressed.
4	The Alt key is pressed.

VkKeyScanEx Windows.pas

Syntax

```
VkKeyScanEx(
ch: Char;              {character value to translate}
dwhkl: HKL             {input locale identifier}
): SHORT;              {returns a translated code}
```

Description

The VkKeyScanEx function translates the specified character code to a virtual key code and returns the status of the Shift, Ctrl, and Alt keys. Numeric keypad keys are not translated. The difference between the VkKeyScan and VkKeyScanEx functions is that the VkKeyScanEx function takes an extra parameter that specifies the input locale identifier. The translation will be performed in the context of that input locale identifier's keyboard layout. The input locale identifier is obtained from the GetKeyboardLayout or LoadKeyboardLayout functions.

Parameters

ch: Specifies the character value of the key to be translated into a virtual key code.

dwhkl: Specifies the input locale identifier used to translate the character to its corresponding virtual key code.

Return Value

If the function succeeds, the low-order byte of the return value contains the virtual key code, and the high-order byte contains a code specifying the state of the Shift, Ctrl, and Alt keys. See the following table for these high-order byte codes. If the function fails, both the high- and low-order bytes contain –1.

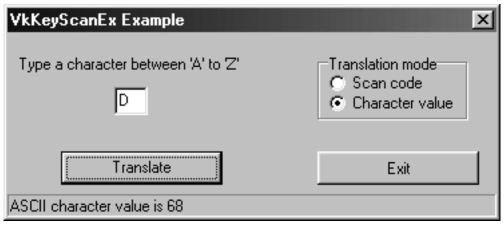
See Also

GetAsyncKeyState, GetKeyboardState, GetKeyNameText, MapVirtualKey, VkKeyScan

Example

Listing 5-11: Translating scan codes to ASCII values

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyKeyboardHandle: HKL;   // holds a keyboard layout handle
  MyChar: Char;            // input character from user
  Mode: Integer;           // type of translation
  ScanString: string;      // result message
begin
  {initialize the displayed message}
  ScanString := 'Scan code is ';

  {if the edit box has text within it, retrieve the first character}
  if Boolean(Edit1.GetTextLen) then
      MyChar:= Edit1.Text[1];

  {if the character is an uppercase letter}
  if IsCharUpper(MyChar) then
  begin
    {retrieve the indicated translation mode}
    if RadioGroup1.ItemIndex = 0 then
      Mode := 0
    else
    begin
      Mode := 2;
      ScanString := 'ASCII character value is ';
    end;

    {retrieve the current keyboard layout}
    MyKeyboardHandle := GetKeyboardLayout(0);

    {display the translated character}
    StatusBar1.SimpleText := ScanString + IntToStr(MapVirtualKeyEx(
                             VkKeyScanEx(MyChar, MyKeyboardHandle), Mode,
                             MyKeyboardHandle));
  end;
end;

procedure TForm1.BitBtn1Click(Sender: TObject);
begin
   Close;
end;
```



ASCII character value is 68

Figure 5-11: The translated character

Table 5-27: VkKeyScanEx Return Values

Value	Description
1	The Shift key is pressed.
2	The Ctrl key is pressed.
4	The Alt key is pressed.

Chapter 6

String and Atom Functions

Although the Object Pascal run-time library provides a rich set of string manipulation and formatting routines, Windows provides a set of equivalent functions that are useful when dealing with internationalization. The Windows functions also give the application access to internal string tables used in reporting error messages. In addition, Windows provides a simple mechanism for sharing string information between processes.

Atom Tables

Windows applications can store strings of up to 255 characters in what is called an atom table. Each process has access to the global atom table and its own local atom table. The global atom table has a maximum of 37 entries and is shared with other processes. The local atom table is private to the process. The local atom table has a default maximum size of 37 entries, but that maximum can be increased when it is created with the InitAtomTable function. The string is added to the atom table by using either the AddAtom or GlobalAddAtom functions and deleted from the table with the DeleteAtom or GlobalDeleteAtom functions. When a string is added to the atom table, the returned value is an atom number that uniquely defines that string within the appropriate table.

Atom tables are normally used to store strings. However, they can also be used to store 16-bit integers. The same functions are used to store integers and strings, but the integer data for the atom table should be given as MakeIntAtom(integer value). MakeIntAtom is a Windows macro that produces a pointer to a null-terminated string that is compatible with the AddAtom and GlobalAddAtom functions. From there, the data is treated as a string. The atom numbers for integer storage will be in the range 1 to 49151 ($0001 to $BFFF), while atom numbers for string storage will be in the range 49152 to 65535 ($C000 to $FFFF). The atom number 0 is used as an error flag.

The most common use of the global atom table is for sharing string data in DDE applications. Rather than passing strings of data, the calling application stores the string in the global atom table and then passes only the 16-bit atom number. The DDE server can then look up the string in the global atom table using the atom number. This technique could also be used to share string data between regular applications by passing the atom number as a parameter inside a user-defined Windows message.

The local atom table exists for the duration of the process. When a process is terminated, the atom table is destroyed. The global atom table exists as long as Windows is running and is not reset until Windows is restarted. Atoms in the global atom table remain there even after the process that stored it is terminated.

It is a good idea to always remove atoms from the atom tables when finished with them. This applies to both the global and the local atom tables. Since there are limits on the size of the global atom table, and also by default for the local atom table (37 entries), it is wise to use the space sparingly and only as long as needed. There may be other applications with entries in the global atom table, so the limit for an application will often be less than 37.

Every atom in both atom tables is reference counted. If the AddAtom or GlobalAddAtom functions make an attempt to add a string to the atom table that is already there, a new entry in the table is not made. The reference count for the existing entry will be incremented and the existing string's unique atom number is returned as the result. When an atom is added that did not previously exist, a new entry is made and the reference count for the new atom is set to one. The DeleteAtom and GlobalDeleteAtom functions decrement the reference count by one and test it. If it is found to be 0, then the atom table entry is deleted. The sole exception to this rule is integer atoms. Integers stored in atom tables do not use reference counts. The deletion functions will delete the integer value from the atom table immediately. Unfortunately, there is no direct way to determine the reference count of an atom. In order to insure that an atom was deleted from the atom table, the application should continually delete the atom using either the DeleteAtom or GlobalDeleteAtom functions until the function fails. An application should only delete atoms that it placed into the atom tables. Never delete an atom that was placed in the atom tables by other processes.

String Formatting

Several of the string functions involve converting characters to uppercase or lowercase. This processing is valid for either single-byte character structures (ANSI) or two-byte character formats (UNICODE). For Windows 95/98/Me, the case conversions and tests are made using the default locale set in the control panel. This could have been set at the time Windows was installed, or it can be altered later. For Windows NT/2000 and later, the conversions and tests are made based on the language driver selected in the control panel or at Windows setup. If no language is selected, Windows NT/2000 makes the conversion based on the default mapping of the code page for the process locale.

Delphi vs. the Windows API

Several functions detailed in this chapter have no representation in Delphi's VCL. The most notable functions would be those that deal with atoms, which can be very useful in certain instances when data needs to be shared between applications in a very efficient manner. GetDateFormat, GetTimeFormat, FormatMessage, and wvsprintf are functions that provide very powerful string formatting capabilities. They provide access to system data and message resources, and give output that is interpreted within the context of specified locales. These functions provide a rich mixture of options and capabilities. There is a significant overlap of functionality with the formatting functions in Delphi's Object Pascal and the VCL. However, knowledge of these functions will be useful when designing Delphi applications for international flexibility.

String and Atom Functions

The following string and atom functions are covered in this chapter.

Table 6-1: String and atom functions

Function	Description
AddAtom	Adds an atom to the local atom table.
CharLower	Converts characters to lowercase.
CharLowerBuff	Converts a range of characters to lowercase.
CharNext	Increments a pointer to the next character in the string.
CharPrev	Decrements a pointer to the previous character in the string.
CharToOem	Converts characters to the OEM character set.
CharToOemBuff	Converts a range of characters to the OEM character set.
CharUpper	Converts characters to uppercase.
CharUpperBuff	Converts a range of characters to uppercase.
CompareString	Compares two strings.
DeleteAtom	Deletes an atom from a local atom table.
EnumSystemCodePages	Lists available and installed system code pages.
EnumSystemLocales	Lists available system locales.
FindAtom	Finds an atom in the local atom table.
FormatMessage	Formats a string with arguments.
GetACP	Retrieves the current ANSI code page for the system.
GetAtomName	Retrieves an atom string from the local atom table.
GetCPInfo	Retrieves code page information.
GetCPInfoEx	Retrieves extended code page information.
GetDateFormat	Retrieves the date in the specified format.
GetOEMCP	Retrieves the current OEM code page identifier for the system.
GetTimeFormat	Retrieves the time in the specified format.
GlobalAddAtom	Adds an atom to the global atom table.
GlobalDeleteAtom	Deletes an atom from the global atom table.
GlobalFindAtom	Finds an atom in the global atom table.
GlobalGetAtomName	Retrieves an atom string from the global atom table.
InitAtomTable	Initializes the size of the local atom table.
IsCharAlpha	Determines if a character is an alphabetic character.
IsCharAlphaNumeric	Determines if a character is alphanumeric.

Function	Description
IsCharLower	Determines if a character is lowercase.
IsCharUpper	Determines if a character is uppercase.
lstrcat	Concatenates two null-terminated strings.
lstrcmp	Compares two null-terminated strings, case sensitive.
lstrcmpi	Compares two null-terminated strings, case insensitive.
lstrcpy	Copies one null-terminated string into another.
lstrlen	Retrieves the length of a null-terminated string.
MakeIntAtom	Creates an integer atom.
OemToChar	Converts a character from the OEM character set to ANSI.
OemToCharBuff	Converts a range of characters from OEM character set to ANSI.
ToAscii	Translates a virtual key code into a Windows character.
wvsprintf	Formats a string with supplied arguments.

AddAtom Windows.pas

Syntax

```
AddAtom(
lpString: PChar              {the string to add to atom table}
): ATOM;                     {returns the newly added atom}
```

Description

This function adds the specified string to the local atom table and returns the atom number. The string can be no longer than 255 characters. If the string already exists in the table, its reference count is incremented. This local atom table is local to the process only and is not shared with other processes.

Local atom tables have a default size of 37 entries. This maximum size can be increased when the local atom table is created. If the string has 255 or fewer characters and the AddAtom function still fails, the most likely cause would be that the table is full.

Note: Strings are not case sensitive. If an existing string differs from the added string only by case, it is treated as an identical string.

Parameters

lpString: A pointer to a null-terminated string to be added to the local atom table.

Return Value

If the function succeeds, it returns the atom number for the string that was added to the local atom table. The atom value is a 16-bit number in the range 49152 to 65535 ($C000 to $FFFF) for strings or in the range 1 to 49151 ($0001 to $BFFF) for integers. If the function fails, it returns zero. To get extended error information, call the GetLastError function.

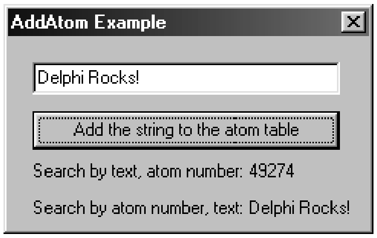
See Also

DeleteAtom, FindAtom, GetAtomName, GlobalAddAtom, GlobalDeleteAtom, GlobalFindAtom, GlobalGetAtomName, MakeIntAtom

Example

Listing 6-1: Adding a string to the local atom table

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  {create an atom table for 200 possible atoms}
  InitAtomTable(200);
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  MyAtom: Atom;          // the returned atom number
  TextTest: PChar;       // string for search result
  AtomTest: Atom;        // atom number from search results
begin
  {store string in local atom table}
  MyAtom := AddAtom(PChar(Edit1.Text));

  {search the table for atom number, given the string}
  AtomTest := FindAtom(PChar(Edit1.Text));
  Label1.Caption := 'Search by text, atom number: ' + IntToStr(Atomtest);

  {search by atom number to get the string}
  TextTest := StrAlloc(256);
  GetAtomName(MyAtom, TextTest, 256);
  Label2.Caption := 'Search by atom number, text: ' + string(TextTest);

  {always clean up entries}
  DeleteAtom(MyAtom);
end;
```



Search by text, atom number: 49274

Search by atom number, text: Delphi Rocks!

Figure 6-1: The atom

CharLower Windows.pas

Syntax

CharLower(
lpsz: PChar {a pointer to the character or string to convert}
): PChar; {returns a pointer to the converted character or string}

Description

This function converts a single character or every character in a null-terminated string to lowercase. Under Windows NT/2000, the function uses the currently selected language driver for the conversion; under Windows 95/98/Me, the conversion is based on the default locale.

Parameters

lpsz: A pointer to the null-terminated string to convert. For single character conversion, load the character into the lower word of lpsz and set the upper 16 bits to zero. CharLower first tests the upper word of the lpsz parameter to determine whether the parameter should be interpreted as a character or as a pointer. In Delphi, the parameter can be typecast as an "array of char" with the single-byte character loaded into the first array element.

Return Value

This function returns either a single character in the lower word and the upper word is zero or a pointer to the null-terminated string containing the converted string. This function does not indicate an error upon failure.

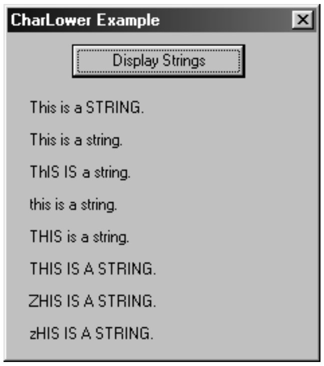
See Also

CharLowerBuff, CharUpper, CharUpperBuff

Example

Listing 6-2: Converting characters and strings to lowercase

```
procedure TForm1.Button1Click(Sender: TObject);
type
  CharArray= array[1..4] of char;
var
  MyString: PChar;       // a pointer to the string
  StartLoc: PChar;       // the start of the string
  NumChars: Integer;     // the number of characters in the string
  MyChar: Char;          // a single character
  MyCharLoc: Pointer;    // a character location within the string
begin
  MyString := 'This is a STRING.';
  Label1.Caption := string(MyString);

  StartLoc := CharNext(MyString);            // do not convert first letter.
  StartLoc := CharLower(StartLoc);           // converts to lowercase.
  Label2.Caption := string(MyString);        // Displays: This is a string.

  StartLoc := CharNext(StartLoc);            // skip another character
  NumChars := CharUpperBuff(StartLoc, 5);    // puts "is is" to uppercase.
  Label3.Caption := string(MyString);        // Displays:  ThIS IS a string.

  NumChars := strlen(MyString);
  NumChars := CharLowerBuff(MyString,NumChars);
  Label4.Caption := string(MyString);        // Displays:   this is a string.
  StartLoc := CharPrev(MyString, StartLoc);  // point to prev character
  StartLoc := CharPrev(MyString, StartLoc);  // points to start of string
  NumChars := CharUpperBuff(StartLoc, 4);    // converts "this" to upper
  Label5.Caption := string(MyString);        // Displays:  THIS is a string.

  StartLoc := CharUpper(StartLoc);
  Label6.Caption := string(MyString);        // Displays:  THIS IS A STRING.

  MyChar := 'z';                             // assign as lowercase
  ZeroMemory(@StartLoc,4);                   // prepare variable with zeroes.
  CharArray(StartLoc)[1] := MyChar;          // load character
  StartLoc := CharUpper(StartLoc);           // convert character
  MyChar := CharArray(StartLoc)[1];          // put it back
  MoveMemory(MyString, @MyChar, 1);          // put it in the string.
  Label7.Caption := string(MyString);        // Displays:  ZHIS IS A STRING.

  StartLoc := CharLower(StartLoc);           // convert character
  MyChar := CharArray(StartLoc)[1];          // put it back again
  MoveMemory(MyString, @MyChar, 1);          // put it in the string.
  Label8.Caption := string(MyString);        // Displays: zHIS IS A STRING.
end;
```



```
This is a STRING.
This is a string.
ThIS IS a string.
this is a string.
THIS is a string.
THIS IS A STRING.
ZHIS IS A STRING.
zHIS IS A STRING.
```

Figure 6-2: The lowercase converted strings

CharLowerBuff   Windows.pas

Syntax

CharLowerBuff(	
lpsz: PChar;	{a pointer to the string to convert}
cchLength: DWORD	{the number of characters to convert}
): DWORD;	{returns the number of characters processed}

Description

CharLowerBuff converts a specified number of characters in the string pointed to by the lpsz parameter to lowercase. Under Windows NT/2000, the function uses the currently selected language driver for the conversion; under Windows 95/98/Me, the conversion is based on the default locale. With this function, it is possible to convert only a portion of a string by pointing the lpsz parameter to the starting position to convert and giving the number of characters to convert in the cchLength parameter. The lpsz parameter does not have to point to the beginning of the string, and the cchLength parameter does not have to reflect the true length of the string.

Parameters

lpsz: A pointer to the null-terminated string that is to be converted to lowercase.

cchLength: Indicates the number of characters to convert. If the string is a Unicode string, then this count is the number of wide (two-byte) character positions. This function will travel past a null character if the cchLength value is larger than the length of the string.

Return Value

If this function succeeds, it returns the number of characters that were processed; otherwise, it returns zero.

See Also

CharLower, CharUpper, CharUpperBuff

Example

Please see Listing 6-2 under CharLower.

CharNext   Windows.pas

Syntax

```
CharNext(
lpsz: PChar                 {a pointer to the current character}
): PChar;                   {returns a pointer to the next character}
```

Description

This function increments the specified pointer to the next character in the string.

Parameters

lpsz: A pointer to the specified character in a null-terminated string.

Return Value

If the function succeeds, it returns a pointer to the next character in a string following the character pointed to by the lpsz parameter. The return value will point to the null terminator if lpsz is already at the end of the string. If the function fails, it returns the lpsz parameter.

See Also

CharPrev

Example

Please see Listing 6-2 under CharLower.

CharPrev Windows.pas

Syntax

```
CharPrev(
lpszStart: PChar;          {a pointer to the start of a string}
lpszCurrent: PChar         {a pointer to the current character}
): PChar;                  {returns a pointer to the previous character}
```

Description

This function returns a pointer to the previous character in the string pointed to by the lpszCurrent parameter.

Parameters

lpszStart: A pointer to the beginning of a string. This is provided so that CharPrev can tell if the lpszCurrent parameter is already at the beginning of the string.

lpszCurrent: A pointer to the current character.

Return Value

If this function succeeds, it returns a pointer to the character prior to the one pointed to by the lpszCurrent parameter. It will point to the beginning of the string if the lpszStart parameter is equal to lpszCurrent. If the function fails, it returns the lpszCurrent parameter.

See Also

CharNext

Example

Please see Listing 6-2 under CharLower.

CharToOem Windows.pas

Syntax

```
CharToOem(
lpszSrc: PChar;            {a pointer to the string to translate}
lpszDst: PChar             {a pointer to the translated string}
): BOOL;                   {always returns TRUE}
```

Description

CharToOem translates each character in the given string into the OEM-defined character set.

Parameters

lpszSrc: A pointer to the source string that is to be translated to an OEM character set string.

lpszDst: A pointer to the destination translated string. If the character set is ANSI (single-byte characters), then the source and destination strings can be the same string. In this case, the translation will be performed in place. If the character set is Unicode (double-byte characters), there must be a separate buffer for lpszDst.

Return Value

This function always returns TRUE.

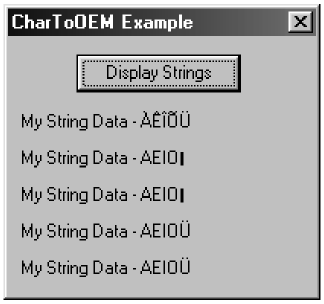
See Also

CharToOemBuff, OemToChar, OemToCharBuff

Example

Listing 6-3: Converting characters to the OEM character set and back

```
procedure TForm1.Button1Click(Sender: TObject);
var
  KeyedIn: PChar;                  // points to input string
  OEMstr: PChar;                   // OEM character set version
  ANSIstr: PChar;                  // ANSI character set version
  OEMbuff: array[1..100] of Char;  // string space
  ANSIbuff: array[1..100] of Char;
begin
  {point PChars to string space}
  OEMstr := @OEMbuff;
  ANSIstr := @ANSIbuff;
  KeyedIn := 'My String Data - ÀÊÎÕÜ';
  Label1.Caption := string(KeyedIn);

  {CharToOem converts string to OEM character set}
  CharToOem(KeyedIn, OEMstr);
  Label2.Caption := string(OEMstr);

  {CharToOemBuff is the counted character version}
  CharToOemBuff(KeyedIn, OEMstr, StrLen(KeyedIn));
  Label3.Caption := string(OEMstr);

  {convert from OEM character set to ANSI characters}
  OemToChar(OemStr, ANSIstr);
  Label4.Caption := string(ANSIstr);

  {OemToCharBuff is the counted character version}
  OemToCharBuff(OemStr, ANSIStr, StrLen(OemStr));
  Label5.Caption := string(ANSIstr);
end;
```



```
My String Data - ÀÊÎÕÜ
My String Data - AEIO▌
My String Data - AEIO▌
My String Data - AEIOÜ
My String Data - AEIOÜ
```

Figure 6-3: The converted characters

CharToOemBuff   Windows.pas

Syntax

```
CharToOemBuff(
lpszSrc: PChar;             {a pointer to the string to translate}
lpszDst: PChar;             {a pointer to the translated string}
cchDstLength: DWORD         {the number of characters to translate}
): BOOL;                    {returns TRUE}
```

Description

CharToOemBuff translates the specified number of characters in the given string into the OEM-defined character set.

Parameters

lpszSrc: A pointer to the source string that is to be translated.

lpszDst: A pointer to the destination translated string. If the character set is ANSI (single-byte characters), then the source and destination strings can be the same string. In this case, the translation will be performed in place. If the character set is Unicode (double-byte characters), there must be a separate buffer for lpszDst.

cchDstLength: The number of characters to translate. If the character set Unicode (double-byte characters), then this is the number of byte pairs (single characters) that will be translated in the destination string.

Return Value

This function always returns TRUE.

See Also

CharToOem, OemToChar, OemToCharBuff

Example

Please see Listing 6-3 under CharToOem.

CharUpper   Windows.pas

Syntax

```
CharUpper(
lpsz: PChar             {a pointer to the character or string to convert}
): PChar;               {returns a pointer to the converted character or string}
```

Description

This function converts a single character or every character in a null-terminated string to uppercase. Under Windows NT/2000, the function uses the currently selected language driver for the conversion; under Windows 95/98/Me, the conversion is based on the default locale.

Parameters

lpsz: A pointer to the null-terminated string to convert. For single-character conversion, load the character into the lower word of lpsz and set the upper 16 bits to zero. CharUpper first tests the upper word of the lpsz parameter to determine whether the parameter should be interpreted as a character or as a pointer. In Delphi, the parameter can be typecast as an "array of char" with the single-byte character loaded into the first array element.

Return Value

This function returns either a single character in the lower word and the upper word is zero or a pointer to the null-terminated string containing the converted string. This function does not indicate an error upon failure.

See Also

CharLower, CharLowerBuff, CharUpperBuff

Example

Please see Listing 6-2 under CharLower.

CharUpperBuff Windows.pas

Syntax

```
CharUpperBuff(
lpsz: PChar;                {a pointer to the string to convert}
cchLength: DWORD            {the number of characters to convert}
): DWORD;                   {returns the number of characters processed}
```

Description

CharUpperBuff converts a specified number of characters in the string pointed to by the lpsz parameter to uppercase. Under Windows NT/2000, the function uses the currently selected language driver for the conversion; under Windows 95/98/Me, the conversion is based on the default locale. With this function, it is possible to convert only a portion of a string by pointing the lpsz variable to the starting position to convert and giving the number of characters to convert in the cchLength parameter. The lpsz parameter does not have to point to the beginning of the string, and the cchLength parameter does not have to reflect the true length of the string.

Parameters

lpsz: A pointer to the null-terminated string that is to be converted to uppercase.

cchLength: Indicates the number of characters to convert. If the string is a Unicode string, this count is the number of wide (two-byte) character positions. This function will travel past a null character if the cchLength value is larger than the string's length.

Return Value

If this function succeeds, it returns the number of characters that were processed; otherwise, it returns zero.

See Also

CharLower, CharLowerBuff, CharUpper

Example

Please see Listing 6-2 under CharLower.

CompareString Windows.pas

Syntax

```
CompareString(
Locale: LCID;              {the locale identifier}
dwCmpFlags: DWORD;         {options for the comparison}
lpString1: PChar;          {a pointer to the first string}
cchCount1: Integer;        {the size in characters of the first string}
lpString2: PChar;          {a pointer to the second string}
cchCount2: Integer         {the size in characters of the second string}
): Integer;                {returns a comparison result code}
```

Description

CompareString performs a comparison of two strings based on the specified locale. The return value of 2 specifies that the two strings are equal in a lexical sense according to the specified flags, even though it is possible for the strings to be different. If the strings are of different length but are compared as lexically equal up to the length of the shortest string, then the longer string will be specified as the greater in the comparison.

If the SORT_STRINGSORT flag is not specified in the dwCmpFlags parameter, the sort will ignore the hyphen and apostrophe characters. This means that "IT'S" will be equal to "ITS" and "DON'T" will be equal to "DONT". Some words might be spelled with an optional hyphen. This default behavior assures that the presence of the hyphen or apostrophe will not affect the ordering of strings in a list. For a stricter character-based sort, use the SORT_STRINGSORT flag. This flag treats the hyphen and apostrophe characters in their normal collating sequence.

In determining which sort function to use, note that CompareString has the availability of using the SORT_STRINGSORT option, where lstrcmp and lstrcmpi will always ignore hyphen and apostrophe characters when comparing strings.

In the Arabic character sets, CompareString will ignore the Arabic Kashidas. This is equivalent to ignoring the hyphen and apostrophe characters if the SORT_STRING-SORT flag is not set. However, there is no option for determining whether or not the Arabic Kashidas will be ignored; they will always be ignored in Arabic character sets.

For fastest execution, set the cchCount1 and cchCount2 parameters to –1 and set the dwCmpFlags parameter to either zero or NORM_IGNORECASE.

Parameters

Locale: The locale identifier that provides the basis for the comparison. In place of a local identifier, this parameter can be one value from Table 6-2.

dwCmpFlags: Flags which determine how the comparison will be made. This parameter can be set to zero to indicate default string comparison, or it can be set to any combination of values from Table 6-3.

lpString1: A pointer to the first string to compare.

cchCount1: Specifies the length of the first string in characters (single bytes for ANSI, double bytes for Unicode). If this value is –1, then CompareString will take the null terminator as the end of the string.

lpString2: A pointer to the second string to compare.

cchCount2: Specifies the length of the second string in characters (single bytes for ANSI, double bytes for Unicode). If this value is –1, then CompareString will take the null terminator as the end of the string.

Return Value

If the function succeeds, the return value will be one of the three comparison return codes in Table 6-4. If the function fails, it returns zero. To get extended error information, call the GetLastError function.

See Also

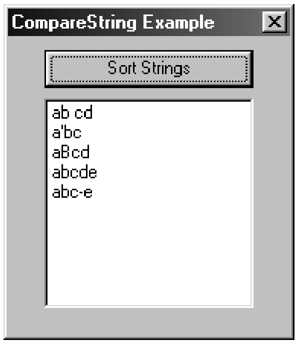
GetSystemDefaultLCID, GetUserDefaultLCID, lstrcmp, lstrcmpi

Example

Listing 6-4: Using CompareString to perform a sort

```
procedure TForm1.Button1Click(Sender: TObject);
var
  L1: array[1..5] of PChar;  // data to sort
  Ltemp: PChar;              // temp for sort swapping
  Xsort,Xloop: Integer;      // loop counters for sort
  CompareResult: Integer;    // result of CompareString
  CompareFlags: DWORD;       // flags parameter
  CompareLocale: LCID;       // locale parameter
begin
  {define some data for sorting}
  L1[1] := 'abc-e';
  L1[2] := 'abcde';
  L1[3] := 'aBcd ';
  L1[4] := 'ab cd';
  L1[5] := 'a''bc ';

  {display the initial strings}
  for Xloop := 1 to 5 do
    ListBox1.Items.Add(string(L1[Xloop]));

  {pause}
  Application.ProcessMessages;
  Sleep(2000);

  {define flags to specify collation (sort) order}
  CompareFlags := NORM_IGNORECASE
```

```
                and SORT_STRINGSORT
                and NORM_IGNORENONSPACE;

  CompareLocale := LOCALE_USER_DEFAULT;

  {do a simplified bubblesort}
  for Xloop := 1 to 4 do
  for Xsort := 1 to 4 do begin
    CompareResult := CompareString(CompareLocale,
                     CompareFlags,
                     L1[Xsort],
                     -1,       // entire length of string
                     L1[Succ(Xsort)],
                     -1);      // entire length of string
    if CompareResult = 0 then begin    // error condition
      ShowMessage('CompareString error!');
      exit;
    end;
    if CompareResult = 3 then begin    // first > second
      {perform a swap}
      Ltemp := L1[xsort];
      L1[Xsort] := l1[Succ(Xsort)];
      L1[Succ(Xsort)] := Ltemp;
    end;
  end;

  {display the results}
  ListBox1.Clear;
  for Xloop := 1 to 5 do
    ListBox1.Items.Add(string(L1[Xloop]));

  // produces a list:
  // ab cd /  a'bc  /  aBcd  /  abcde  /  abc-e
  // these were interpreted with the above options as
  // ab cd /  abc   /  abcd  /  abcde  /  abce
  // spaces are interpreted as spaces
  // apostrophes and hyphens are deleted (ignored).
end;
```



```
ab cd
a'bc
aBcd
abcde
abc-e
```

Figure 6-4: The sorted strings

Listing 6-5: Comparing two strings for equality

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyResult: Integer;
begin
  {compare the strings}
  MyResult := CompareString(LOCALE_USER_DEFAULT,
                            NORM_IGNORECASE,
                            PChar(Edit1.Text),
                            -1,
                            PChar(Edit2.Text),
                            -1);
  if MyResult = 1 then
  begin   // first parameter is greater
    Label1.Caption := 'SMALLER';
    Label2.Caption := 'GREATER';
    Label3.Caption := '<';
  end;
  if MyResult = 2 then
  begin
    Label1.Caption := ' equal ';
    Label2.Caption := ' equal ';
    Label3.Caption := '=';
  end;
  if MyResult = 3 then
  begin
    Label1.Caption := 'GREATER';
    Label2.Caption := 'SMALLER';
    Label3.Caption := '>';
  end;
  if MyResult = 0 then
    ShowMessage('Error in CompareString');
end;
```

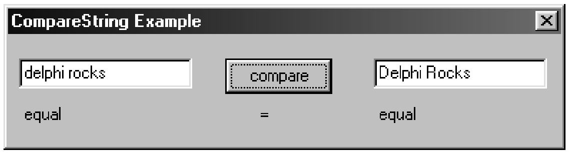


Figure 6-5: The two strings are equal

Table 6-2: CompareString Locale values

Value	Description
LOCALE_SYSTEM_DEFAULT	The system's default locale.
LOCALE_USER_DEFAULT	The user's default locale.

Table 6-3: CompareString dwCmpFlags values

Value	Description
NORM_IGNORECASE	Ignore uppercase vs. lowercase They are treated as equal if this flag is set.
NORM_IGNOREKANATYPE	The Hiragana and the Katakana characters will be treated as equivalent character sets.

Value	Description
NORM_IGNORENONSPACE	Ignore non-spacing characters.
NORM_IGNORESYMBOLS	Ignore symbols.
NORM_IGNOREWIDTH	Character set width is ignored when comparing an ANSI character string to a UNICODE character string. The same character in the ANSI set and UNICODE set are regarded as equal.
SORT_STRINGSORT	Punctuation characters are treated as symbols.

Table 6-4: CompareString return values

Value	Description
1	The first string is less in value than the second string in lexical comparison.
2	The two strings have equal lexical values according to the flags that are provided.
3	The first string is greater in value than the second string in lexical comparison.

DeleteAtom *Windows.pas*

Syntax

```
DeleteAtom(
nAtom: ATOM          {the atom number to delete}
): ATOM;             {returns zero or the nAtom value}
```

Description

DeleteAtom reduces the reference count for the specified atom in the local atom table by one. If the reference count for the specified atom is zero, the entry is deleted from the atom table. To make a deletion from the global atom table, use GlobalDeleteAtom.

Parameters

nAtom: The atom number to delete from the local atom table.

Return Value

If this function succeeds, it returns zero; otherwise, it returns the atom number in the nAtom parameter. To get extended error information, call the GetLastError function.

See Also

AddAtom, FindAtom, GlobalAddAtom, GlobalDeleteAtom, GlobalFindAtom

Example

Please see Listing 6-1 under AddAtom.

EnumSystemCodePages   Windows.pas

Syntax

```
EnumSystemCodePages(
lpCodePageEnumProc: TFNCodepageEnumProc;    {a pointer to the callback
                                             function}
dwFlags: DWORD                              {code page selection options}
): BOOL;                                    {returns TRUE or FALSE}
```

Description

This function enumerates code pages that are installed or supported by the system. The callback function is called for each code page that is identified as meeting the selection criteria. The callback function receives each code page identifier and can store it programmatically according to the need of the calling routine. The process will continue until all code pages have been processed or the callback function returns zero.

Parameters

lpCodePageEnumProc: The address of the callback function provided by the caller of EnumSystemCodePages. The callback function receives a code page identifier for each installed or supported code page.

dwFlags: Determines which code pages to report to the callback function. This parameter can be one value from the following table.

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

Callback Syntax

```
EnumSystemCodePagesProc(
AnotherCodePage: PChar     {a pointer to a string containing a code page identifier}
): Integer;                {indicates if enumeration should continue}
```

Description

This callback function is called once for each code page identifier installed or supported by the system and can perform any desired task.

Parameters

AnotherCodePage: A pointer to a null-terminated string containing a code page identifier.

Return Value

To continue enumeration, the callback function should return one; otherwise, it should return zero.

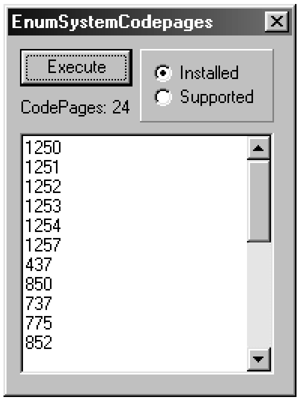
See Also

EnumSystemLocales

Example

Listing 6-6: Enumerating the system code pages

```
{the callback function prototype}
function EnumCodePageProc(AnotherCodePage: PChar): Integer; stdcall;

implementation

function EnumCodePageProc(AnotherCodePage: PChar):integer;
// callback function is called as many times as there are Code Pages.
// A single call to EnumSystemCodePages triggers the series of callbacks.
begin
  {display the code page}
  CodePages.Memo1.Lines.Add(string(AnotherCodePage));

  {continue enumeration}
  Result := 1;
end;

procedure TCodePages.ButtonExecuteClick(Sender: TObject);
var
  MyFlags: Integer; // parameter specified by Radio Buttons in this example.
begin
  {initialize for enumeration}
  Memo1.Clear;
  MyFlags := CP_SUPPORTED;  // set flags from radio buttons
  if RBinstalled.Checked then MyFlags := CP_INSTALLED;

  {enumerate code pages}
  if not EnumSystemCodePages(@EnumCodePageProc,MyFlags)
    then ShowMessage('Error getting system code pages');

  Label1.Caption := 'CodePages: '+IntToStr(Memo1.Lines.Count);
end;
```



```
CodePages: 24
1250
1251
1252
1253
1254
1257
437
850
737
775
852
```

Figure 6-6: The system code page list

Table 6-5: EnumSystemCodePages dwFlags values

Value	Description
CP_INSTALLED	Report only code pages that are currently installed.
CP_SUPPORTED	Report all code pages that are supported by the system.

EnumSystemLocales Windows.pas

Syntax

```
EnumSystemLocales(
lpLocaleEnumProc: TFNLocaleEnumProc;    {a pointer to the callback function}
dwFlags: DWORD                          {locale selection options}
): BOOL;                                {returns TRUE or FALSE}
```

Description

This function enumerates locales that are installed or supported by the system. The callback function is called for each locale that is identified as meeting the selection criteria. The callback function receives each locale identifier and can store it programmatically according to the needs of the calling routine. The process will continue until all locales have been processed or the callback function returns zero.

Parameters

lpLocaleEnumProc: The address of the callback function provided by the caller of EnumSystemLocales. The callback function receives a locale code for each installed or supported locale.

dwFlags: Determines which locales to report to the callback function. This parameter can be one value from the following table.

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

Callback Syntax

```
EnumSystemLocalesProc(
AnotherLocale: PChar          {a pointer to a string containing a locale code}
): Integer;                   {indicates if enumeration should continue}
```

Description

This callback function is called once for each locale installed or supported by the system and can perform any desired task.

Parameters

AnotherLocale: A pointer to a null-terminated string containing a locale code.

Return Value

To continue enumeration, the callback function should return one; otherwise, it should return zero.

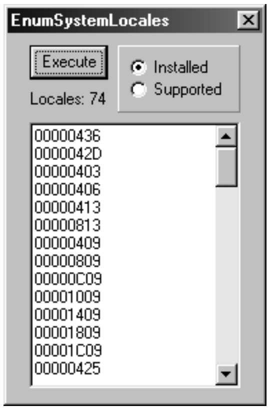
See Also

EnumSystemCodePages

Example

Listing 6-7: Enumerating system locales

```
{the callback function prototype}
function EnumLocalesProc(AnotherLocale: PChar): Integer; stdcall;

implementation

function EnumLocalesProc(AnotherLocale: PChar): Integer;
// callback function is called for each locale.
// A single call to EnumSystemLocales
//    triggers the series of callbacks.
begin
  {display the locale}
  Locales.Memo1.Lines.Add(AnotherLocale);

  {continue enumeration}
  EnumLocalesProc := 1;
end;

procedure TLocales.ButtonExecuteClick(Sender: TObject);
var
  MyFlags: Integer;
begin
  {initialize for enumeration}
  Memo1.Clear;
  MyFlags := LCID_INSTALLED;
  If RBsupported.Checked then MyFlags := LCID_SUPPORTED;

  {enumerate the locales}
  if not EnumSystemLocales(@EnumLocalesProc,MyFlags)
    then ShowMessage('Error in getting locales.');

  Label1.Caption := 'Locales: '+IntToStr(Memo1.Lines.Count);
end;
```



Locales: 74

```
00000436
0000042D
00000403
00000406
00000413
00000813
00000409
00000809
00000C09
00001009
00001409
00001809
00001C09
00000425
```

Figure 6-7: The system locales list

Table 6-6: EnumSystemLocales dwFlags values

Value	Description
LCID_INSTALLED	Report only locales that are currently installed.
LCID_SUPPORTED	Report all locales that are supported by the system.

FindAtom Windows.pas

Syntax

```
FindAtom(
lpString: PChar          {a pointer to the string to search for in the local atom table}
): ATOM;                 {returns the atom number for the string}
```

Description

FindAtom searches the local atom table for the string pointed to by the lpString parameter and returns the atom number if it is found. The string comparison is not case sensitive. To find an atom in the global atom table, use the GlobalFindAtom function.

Parameters

lpString: A pointer to the null-terminated string to search for in the local atom table.

Return Value

If the function succeeds, it returns the atom number for the specified string; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

AddAtom, DeleteAtom, GlobalAddAtom, GlobalDeleteAtom, GlobalFindAtom

Example

Please see Listing 6-1 under AddAtom.

FormatMessage Windows.pas

Syntax

```
FormatMessage(
dwFlags: DWORD;          {formatting and option flags}
lpSource: Pointer;       {a pointer to the message source}
dwMessageId: DWORD;      {the message identifier}
dwLanguageId: DWORD;     {the language identifier}
lpBuffer: PChar;         {a pointer to a buffer for output}
nSize: DWORD;            {the size of the message buffer in bytes}
Arguments: Pointer       {a pointer to an array of message arguments}
): DWORD;                {returns the number of bytes stored in the output buffer}
```

Description

FormatMessage prepares a message from a message identifier, a message table, a selected language, and a variety of formatting options. It can be used for either system or user-defined messages. FormatMessage will take the message identifier and search a message table for a suitable message that is available in the selected language. If no language is specified, it will search for a suitable language from a prioritized list of reasonable language alternatives. On finding a message string, it will process it according to the message arguments and copy the result to an output buffer.

If the dwFlags parameter contains the FORMAT_MESSAGE_FROM_STRING flag, the string pointed to by lpSource may contain special codes that designate where arguments are to be placed in the message and how they are to be formatted. The FormatMessage function reads the string, reads the corresponding arguments, and places the formatted result in the output buffer. The possible formatting instructions are described in Table 6-9.

Parameters

dwFlags: A set of flags which determine how FormatMessage will operate and the meaning of the lpSource parameter. This parameter can be one or more values from Table 6-7. The low-order byte (see Table 6-8) specifies how FormatMessage outputs line breaks and also the maximum width of an output line.

lpSource: A pointer to the source of the message. This will be either a module handle or a pointer to a null-terminated string, depending on the flags present in the dwFlags parameter. If neither the FORMAT_MESSAGE_FROM_HMODULE nor the FORMAT_MESSAGE_FROM_STRING flags are specified, the value of lpSource is ignored.

dwMessageId: The 32-bit message identifier that is used to search a message table. This parameter is ignored if the dwFlags parameter contains the FORMAT_MESSAGE_FROM_STRING flag.

dwLanguageId: The 32-bit language identifier that specifies which language is used when retrieving the message definition from a message resource table. This parameter is ignored if the dwFlags parameter contains the FORMAT_MESSAGE_FROM_STRING flag. If no message is found in the specified language, then the function returns a value of ERROR_RESOURCE_LANG_NOT_FOUND. If this parameter contains zero and the FORMAT_MESSAGE_FROM_STRING flag is not set in the dwFlags parameter, then the function searches for a message definition for a language in the following order of priority:

1. Language-neutral message definition, if present.
2. Thread LANGID. This is the language of the thread's locale.
3. User default LANGID. This is the language of the user's default locale.
4. System default LANGID. This is the language of the system's default locale.
5. U.S. English.
6. Any language.

lpBuffer: A pointer to a buffer for the output message. The caller prepares this buffer before FormatMessage is called unless the dwFlags parameter contains the FORMAT_MESSAGE_ALLOCATE_BUFFER flag. If this flag is set, then FormatMessage uses LocalAlloc to allocate the required amount of space, storing the buffer's address in the lpBuffer parameter. Note that the FORMAT_MESSAGE_ALLOCATE_BUFFER flag indicates that lpBuffer is a pointer to a pointer to a buffer.

nSize: If the FORMAT_MESSAGE_ALLOCATE_BUFFER flag is present in the dwFlags parameter, nSize specifies the minimum number of bytes to allocate for the output buffer. This allocation is carried out by FormatMessage and deallocated by the caller with LocalFree. If the FORMAT_MESSAGE_ALLOCATE_BUFFER flag is not set, then nSize indicates the maximum size of the output buffer.

Arguments: A pointer to an array of 32-bit arguments that are used to fill in the insertion points in the string pointed to by the lpSource parameter. The substring "%1" within the message is the location where the first argument will be placed; "%2" is where the second argument is placed, and so on. The output formatting of the argument will depend on additional codes in the string associated with that argument. If there are no additional formatting codes for an argument, the argument is treated as a PChar.

Return Value

If the function succeeds, it returns the number of bytes copied to the output buffer, excluding the null terminator character. If the function fails, it returns zero. To get extended error information, call the GetLastError function.

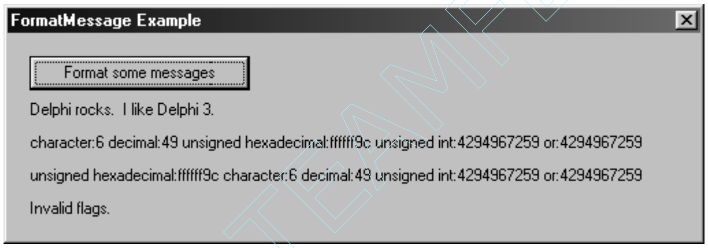
See Also

wvsprintf

Example

Listing 6-8: Formatting messages

```
{Whoops! Delphi does not automatically import this function, so
 we must do it manually}
function LocalHandle(Mem: Pointer): HLOCAL; stdcall;

implementation

{link in the function}
function LocalHandle; external kernel32 name 'LocalHandle';

procedure TForm1.Button1Click(Sender: TObject);
{ Examples of how to use FormatMessage.        }
{ allowed printf (c style) formatting options: }
{ x is unsigned hexadecimal format             }
{ u is unsigned integer                        }
{ d is decimal integer                         }
{ c is a character                             }
{ lu is accepted for unsigned long integer     }
{     but the l is not required.               }
{ o for octal is not supported                 }
{ e, f, F, g, and G formats are not supported. }
{ Use some other technique for floating point. }
const
  MyMessageSize = 200;
var
  MyMessageDefinition: Pchar;        // message format
  OutputMessage: PChar;              // holds a formatted message
  MyArguments: array[1..5] of PChar; // pointers to numeric arguments
  MyMessageID: Integer;              // holds a message identifier
  MyLanguageID: Integer;             // holds a language identifier
  MyMemoryHandle: HLOCAL;            // a handle to a formatted message
begin
  {Get space for output message}
  GetMem(OutputMessage,MyMessageSize);

  {format a message}
  MyMessageDefinition := 'Delphi %1. I like %2.';
  MyArguments[1] := 'rocks';
  MyArguments[2] := 'Delphi 3';
  FormatMessage(FORMAT_MESSAGE_FROM_STRING or
              FORMAT_MESSAGE_ARGUMENT_ARRAY,
              MyMessageDefinition,0,0,
              OutputMessage,MyMessageSize,
              @MyArguments);
  MyArguments[1] := '';
  MyArguments[2] := '';

  {displays: Delphi rocks. I like Delphi 3}
  Label1.Caption := string(OutputMessage);

  {examples of numeric substitution. Arguments contain the data.
   This uses "C" style printf formatting}
  Integer(MyArguments[1]) := 54;
  Integer(MyArguments[2]) := 49;
  Integer(MyArguments[3]) := -100;
  Integer(MyArguments[4]) := -37;
  Integer(MyArguments[5]) := -37;

  {format the message}
  MyMessageDefinition :=
      'character:%1!c! decimal:%2!d! unsigned hexadecimal:%3!x!'
    + ' unsigned int:%4!u! or:%5!lu!';
  FormatMessage(FORMAT_MESSAGE_FROM_STRING or
              FORMAT_MESSAGE_ARGUMENT_ARRAY,
              MyMessageDefinition,0,0,
              OutputMessage,MyMessageSize,
              @MyArguments);
  Label2.Caption := string(OutputMessage);

  {format the message differently}
  MyMessageDefinition :=
      'unsigned hexadecimal:%3!x! character:%1!c! decimal:%2!d!'
    + ' unsigned int:%4!u! or:%5!lu!';
  FormatMessage(FORMAT_MESSAGE_FROM_STRING or
              FORMAT_MESSAGE_ARGUMENT_ARRAY,
              MyMessageDefinition,0,0,
              OutputMessage,MyMessageSize,
                @MyArguments);
  Label3.Caption := string(OutputMessage);

  {free output message space}
  Freemem(OutputMessage);

  {retrieve the system string for an error message}
  MyMessageID := ERROR_INVALID_FLAGS; // any system message ID
  MyLanguageID := 0;                  // default language

  {Use the option where FormatMessage allocates its own message buffer.}
  FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM or
                FORMAT_MESSAGE_ALLOCATE_BUFFER,nil,
                MyMessageID,MyLanguageID,
                @OutputMessage,0,nil);
  Label4.Caption := string(OutputMessage);

  {return message memory}
  MyMemoryHandle := LocalHandle(OutputMessage);
  if (LocalFree(MyMemoryHandle)  0) then
    ShowMessage('Error freeing memory');

end;
```



```
Delphi rocks.  I like Delphi 3.
character:6 decimal:49 unsigned hexadecimal:ffffff9c unsigned int:4294967259 or:4294967259
unsigned hexadecimal:ffffff9c character:6 decimal:49 unsigned int:4294967259 or:4294967259
Invalid flags.
```

Figure 6-8: The formatted messages

Table 6-7: FormatMessage dwFlags values

Value	Description
FORMAT_MESSAGE_ALLOCATE_BUFFER	Indicates that the lpBuffer parameter is a pointer to a pointer for the output buffer. The output buffer has no memory allocation when FormatMessage is called. The function will use LocalAlloc to allocate enough memory to store the message. The nSize parameter specifies a minimum size to allocate for the buffer. FormatMessage then creates the buffer and places the address of the buffer at the address pointed to by the lpBuffer parameter. The application should declare a pointer for the output buffer and place the address of the pointer in the lpBuffer parameter. After the buffer is no longer needed, the caller should free the memory with a call to LocalFree.
FORMAT_MESSAGE_IGNORE_INSERTS	Specifies that FormatMessage will ignore the Arguments parameter and pass the insert sequences to the output buffer without processing the values in the Arguments array.

Value	Description
FORMAT_MESSAGE_FROM_STRING	Indicates that lpSource is a pointer to a message string which may contain insert sequences. This flag cannot be combined with either FORMAT_MESSAGE_FROM_HMODULE or FORMAT_MESSAGE_FROM_SYSTEM.
FORMAT_MESSAGE_FROM_HMODULE	Indicates that lpSource is a handle to a message table resource. If this option is set and lpSource is NIL, the resources in the current process will be searched. This flag cannot be used in combination with FORMAT_MESSAGE_FROM_STRING.
FORMAT_MESSAGE_FROM_SYSTEM	Indicates that FormatMessage should search the system's message table resources for the message definition. If this flag is specified in combination with FORMAT_MESSAGE_FROM_HMODULE, FormatMessage will search the message table resources in the system and then in the module specified by lpSource. This flag cannot be used in combination with FORMAT_MESSAGE_FROM_STRING. This flag allows the application to pass the value returned from GetLastError to retrieve a system-defined error message string.
FORMAT_MESSAGE_ARGUMENT_ARRAY	Specifies that the Arguments parameter is a pointer to an array of 32-bit values to be inserted into the message string.

Table 6-8: FormatMessage dwFlags low-order byte values

Value	Description
0	There are no specifications for line breaks for the output buffer. FormatMessage will simply copy any line breaks in the message definition to the output buffer.
any non-zero value that is not FORMAT_MESSAGE_MAX_WIDTH_MASK	Indicates the maximum width of an output line. A line break will be used in the output buffer to send any remaining characters to the next line if it exceeds this value. Any line breaks that occur in the message definition will be ignored. FormatMessage will only break the line at a white space. Hard-coded line breaks in the message definition (%n) will be copied to the output buffer, and a new line count will begin for the next line.
FORMAT_MESSAGE_MAX_WIDTH_MASK	Regular line breaks in the message string will be ignored. Hard-coded line breaks will be copied to the output buffer. No other line breaks will be generated by FormatMessage regardless of line width.

Table 6-9: FormatMessage arguments formatting code values

Value	Description
%0 (zero)	Terminates the output message without a newline character. Use this for a prompt or any situation where the cursor should remain at the end of the output without a carriage return/line feed being generated.
%n!*C-style printf format string*!	An argument insert point for the nth argument. "%1" is the first argument, "%2" is the second argument, and so on, up to a possible total of 99 arguments. The formatting instructions must include the exclamation points or it will simply be interpreted as part of the message. If no printf formatting is specified, !s! is used as a default, meaning that the corresponding argument is interpreted as a PChar variable pointing to a null-terminated string which is inserted in the message at that point in place of the "%n". If the printf format code is a numeric formatting code showing width and precision, then the "*" character can be given for either or both numbers. If a single "*" is given, then the next (%n+1) argument is used for the number. If two "*" characters are in the formatting code, then the next two arguments (%n+1 and %n+2) are used for the width and precision of the formatted number. The C-style formatting for floating-point numbers using the o, e, E, f, g, and G options are not supported. However, x, d, c, u, and lu are supported. Integers in octal format or real numbers in scientific notation should be formatted using other functions.
%%	Passes a single % symbol to the output buffer.
%n (the letter "n")	Places a hard line break in the output buffer. This is used when the message definition is determining the line width for a multiple-line message.
%space	Forces a space in the output buffer. There can be several of these in sequence. This is useful for forcing spaces, including trailing spaces.
%. (period)	Forces a period to the output buffer regardless of position in the line. This can be used to place a period at the beginning of the line. A period without a percent sign would indicate the end of the message.
%!	Places an exclamation mark in the output buffer without being interpreted as a beginning or ending terminator for a formatting specification.

GetACP Windows.pas

Syntax

```
GetACP: UINT;      {returns an ANSI code page identifier}
```

Description

GetACP gets the current ANSI code page identifier for the system. If a code page is not defined, then the default code page identifier is returned.

Return Value

If the function succeeds, it returns a 32-bit ANSI code page identifier from Table 6-10; otherwise, it returns zero.

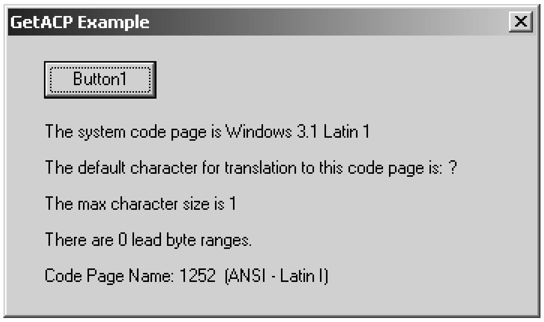
See Also

GetCPInfo, GetCPInfoEx, GetOEMCP

Example

Listing 6-9: Retrieving the current ANSI code page

```
  {Delphi does not define the TCPInfoEx structure}
  TCPInfoEx = record
    MaxCharSize: UINT;                                 { max length (bytes) of a char }
    DefaultChar: array[0..MAX_DEFAULTCHAR - 1] of Byte; { default character }
    LeadByte: array[0..MAX_LEADBYTES - 1] of Byte;     { lead byte ranges }
    UnicodeDefaultChar: WCHAR;                         { default unicode char }
    CodePage: UINT;                                    { the code page }
    CodePageName: array[0..MAX_PATH] of Char;          { code page name }
  end;

  {Delphi does not import GetCPInfoEx}
  function GetCPInfoEx(CodePage: UINT; dwFlags: DWORD;
                       var lpCPInfoEx: TCPInfoEx): BOOL; stdcall;

var
  Form1: TForm1;

implementation

{$R *.DFM}

{import the function}
function GetCPInfoEx; external 'kernel32.dll' name 'GetCPInfoExA';

procedure TForm1.Button1Click(Sender: TObject);
var
  SystemCodePage: Integer;       // holds the system code page
  SystemCPInfo: TCPInfo;         // holds code page info
  SystemCPInfoEx: TCPInfoEx;     // holds extended code page info
  Leadbytecount: Integer;        // indicates lead byte count
  LeadX: Integer;                // loop counter
begin
  {retrieve the system code page}
  SystemCodePage := GetACP;
  case SystemCodePage of
    874:  Label1.Caption := 'The system code page is Thai';
    932:  Label1.Caption := 'The system code page is Japan';
    936:  Label1.Caption := 'The system code page is Chinese (PRC, Singapore)';
    949:  Label1.Caption := 'The system code page is Korean';
    950:  Label1.Caption := 'The system code page is Chinese (Taiwan)';
    1200: Label1.Caption := 'The system code page is Unicode';
    1250: Label1.Caption := 'The system code page is Windows 3.1 East Europe';
    1251: Label1.Caption := 'The system code page is Windows 3.1 Cyrillic';
    1252: Label1.Caption := 'The system code page is Windows 3.1 Latin 1';
    1253: Label1.Caption := 'The system code page is Windows 3.1 Greek';
    1254: Label1.Caption := 'The system code page is Windows 3.1 Turkish';
    1255: Label1.Caption := 'The system code page is Hebrew';
    1256: Label1.Caption := 'The system code page is Arabic';
    1257: Label1.Caption := 'The system code page is Baltic';
  else
    Label1.Caption := 'The system code page is ' +IntToStr(SystemCodePage);
  end;

  {the TCPinfo parameter is a VAR parameter,
   just give the variable, not the address of the variable}
  if not GetCPInfo(SystemCodePage, SystemCPInfo)
    then Label2.Caption := 'Error in getting CP info.'
    else Label2.Caption := 'The default character for translation '
                + 'to this code page is: '
                + Char(SystemCPInfo.DefaultChar[0]);

  {display the character size}
  Label3.Caption := 'The max character size is '
                    + IntToStr(SystemCPInfo.MaxCharSize);

  {determine lead byte count}
  LeadByteCount := 0;
  for leadX := 5 downto 0 do
    if SystemCPInfo.LeadByte[2*leadx] = 0 then LeadByteCount := LeadX;

  Label4.Caption := 'There are '+IntToStr(LeadByteCount)+' lead byte ranges.';

  {get some extended code page information}
  if not GetCPInfoEx(SystemCodePage, 0, SystemCPInfoEx)
    then Label5.Caption := 'Error in getting extended CP info.'
    else Label5.Caption := 'Code Page Name: ' + SystemCPInfoEx.CodePageName;
end;
```



```
The system code page is Windows 3.1 Latin 1
The default character for translation to this code page is: ?
The max character size is 1
There are 0 lead byte ranges.
Code Page Name: 1252  (ANSI - Latin I)
```

Figure 6-9: The current ANSI code page information

Table 6-10: GetACP return values

Value	Description
874	Thai
932	Japanese
936	Chinese (PRC, Singapore)
949	Korean
950	Chinese (Taiwan, Hong Kong)
1200	Unicode (BMP of ISO 10646)
1250	Windows 3.1 Eastern European

Value	Description
1251	Windows 3.1 Cyrillic
1252	Windows 3.1 Latin 1 (U.S., Western Europe)
1253	Windows 3.1 Greek
1254	Windows 3.1 Turkish
1255	Hebrew
1256	Arabic
1257	Baltic

GetAtomName Windows.pas

Syntax

```
GetAtomName(
nAtom: ATOM;              {an atom number}
lpBuffer: PChar;          {a pointer to a buffer receiving the name}
nSize: Integer            {the length of the lpBuffer buffer}
): UINT;                  {returns the number of characters copied to the buffer}
```

Description

This function finds the string associated with the specified atom number in the local atom table. If an integer value was stored, the value will be returned in a string format. For retrieving a string from an atom number from the global atom table, use the GlobalGetAtomName function.

Parameters

nAtom: The atom number whose string is to be retrieved.

lpBuffer: A pointer to a string buffer where the function will place the results of the search. The string buffer should be large enough to receive the string value plus the null terminator.

nSize: Specifies the size of the buffer pointed to by the lpBuffer parameter.

Return Value

If the function succeeds, the buffer pointed to by the lpBuffer parameter will contain the string associated with the specified atom number, and the function returns the number of characters copied to this buffer. If the function fails, it returns zero. To get extended error information, call the GetLastError function.

See Also

AddAtom, DeleteAtom, FindAtom, GlobalAddAtom, GlobalDeleteAtom, GlobalFindAtom, GlobalGetAtomName

Example

Please see Listing 6-1 under AddAtom.

GetCPInfo Windows.pas

Syntax

```
GetCPInfo(
CodePage: UINT;                 {the code page identifier}
var lpCPInfo: TCPInfo           {a pointer to a TCPInfo structure}
): BOOL;                        {returns TRUE or FALSE}
```

Description

This function retrieves information about a specified code page and places the results in a TCPInfo structure.

Parameters

CodePage: The 32-bit code page identifier for which information is requested. This parameter can also be set to one value from the following table.

lpCPInfo: A pointer to a TCPInfo structure that receives the code page information. The TCPInfo data structure is defined as:

```
TCPInfo = record
    MaxCharSize: UINT;                      {max length of a char in bytes}
    DefaultChar: array[0..MAX_DEFAULTCHAR – 1]
    of Byte;                                {the default character}
    LeadByte: array[0..MAX_LEADBYTES – 1]
    of Byte;                                {lead byte ranges}
end;
```

MaxCharSize: Specifies the default length, in bytes, of a character in this code page (1 for ANSI, 2 for UNICODE).

DefaultChar: Specifies the default character used in character translations to this code page.

LeadByte: Specifies an array of lead byte ranges. Lead byte ranges are only used in double-byte character set code pages.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetACP, GetCPInfoEx, GetOEMCP

Example

Please see Listing 6-9 under GetACP.

Table 6-11: GetCPInfo CodePage values

Value	Description
CP_ACP	Uses the system default ANSI code page.
CP_MACCP	Uses the system default Macintosh code page.
CP_OEMCP	Uses the system default OEM code page.

GetCPInfoEx Windows.pas

Syntax

```
GetCPInfo(
CodePage: UINT;                {the code page identifier}
dwFlags: DWORD;                {reserved}
var lpCPInfoEx: TCPInfoEx      {a pointer to a TCPInfoEx structure}
): BOOL;                       {returns TRUE or FALSE}
```

Description

This function retrieves extended information about a specified code page and places the results in a TCPInfoEx structure.

Note: This function is available only under Windows 2000 and later.

Parameters

CodePage: The 32-bit code page identifier for which information is requested. This parameter can also be set to one value from the following table.

dwFlags: Reserved, must be set to zero.

lpCPInfoEx: A pointer to a TCPInfoEx structure that receives the extended code page information. The TCPInfoEx data structure is defined as:

```
TCPInfoEx = record
      MaxCharSize: UINT;                               {max length of a char in bytes}
      DefaultChar: array[0..MAX_DEFAULTCHAR – 1] of Byte; {default character}
      LeadByte: array[0..MAX_LEADBYTES – 1] of Byte;   {lead byte ranges}
      UnicodeDefaultChar: WCHAR;                       {default Unicode char}
      CodePage: UINT;                                  {the code page}
      CodePageName: array[0..MAX_PATH] of Char;        {code page name}
end;
```

MaxCharSize: Specifies the default length, in bytes, of a character in this code page (1 for ANSI, 2 for UNICODE).

DefaultChar: Specifies the default character used in character translations to this code page.

LeadByte: Specifies an array of lead byte ranges. Lead byte ranges are only used in double-byte character set code pages.

UnicodeDefaultChar: Specifies the default Unicode character used in character translations to this code page.

CodePage: Specifies the code page value.

CodePageName: Contains the localized name of the code page.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetACP, GetCPInfo, GetOEMCP

Example

Please see Listing 6-9 under GetACP.

Table 6-12: GetCPInfoEx CodePage values

Value	Description
CP_ACP	Uses the system default ANSI code page.
CP_MACCP	Uses the system default Macintosh code page.
CP_OEMCP	Uses the system default OEM code page.

GetDateFormat Windows.pas

Syntax

```
GetDateFormat(
Locale: LCID;               {the locale for specifying the date format}
dwFlags: DWORD;             {formatting options}
lpDate: PSystemTime;        {a pointer to the date to be formatted}
lpFormat: PChar;            {a pointer to the date format string}
lpDateStr: PChar;           {a pointer to a buffer for the formatted date}
cchDate: Integer            {the size of the output buffer}
): Integer;                 {returns the number of characters copied to the buffer}
```

Description

This function formats a given date according to a format string and specified options. A specific date or the system date can be used.

Parameters

Locale: The locale identifier to which the date is formatted. If the lpFormat parameter is NIL, the date is formatted according to the default date formatting for this locale. If the lpFormat parameter contains a formatting string, then the locale will only be used for formatting information not specified in the lpFormat string. Alternatively, this parameter can be set to one of the values from Table 6-13.

dwFlags: Indicates formatting options if the lpFormat parameter is set to NIL. If the lpFormat parameter is not set to NIL, then dwFlags must be zero. If the lpFormat parameter is NIL, the dwFlags parameter can be any combination of values from Table 6-14.

lpDate: A pointer to a TSystemTime structure that contains the date to format. If this parameter is set to NIL, the current system date will be used. The TSystemTime structure is defined as:

```
TSystemTime = record
      wYear: Word;                {indicates the year}
      wMonth: Word;               {indicates the month}
      wDayOfWeek: Word;           {indicates the day of the week}
      wDay: Word;                 {indicates the day of the month}
      wHour: Word;                {indicates the hour}
      wMinute: Word;              {indicates the minute)
      wSecond: Word;              {indicates the seconds}
      wMilliseconds: Word;        {indicates the milliseconds}
end;
```

Please see the FileTimeToSystemTime function for a description of this data structure.

lpFormat: A pointer to a string containing the format picture. If this parameter is NIL, the default date format for the locale will be used. This string is a formatting picture containing formatting codes from Table 6-15. Spaces in the format will show in the output as they appear. Other than spaces, text or other literal characters may appear in single quotations.

lpDateStr: A pointer to the string buffer that receives the formatted date string.

cchDate: Indicates the size of the output buffer pointed to by the lpDateStr parameter in characters. If this parameter is zero, the function returns the number of characters that would be required to hold the formatted output and the lpDateStr parameter is ignored.

Return Value

If the function succeeds, it returns the number of characters copied to the lpDateStr buffer; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

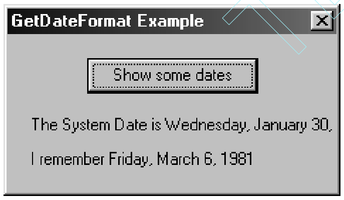
GetTimeFormat

Example

Listing 6-10: Formatting dates

```
const
  {Delphi 6 does not define these GetDateFormat flags}
  DATE_YEARMONTH    = $00000008;  // use year month picture
  DATE_LTRREADING   = $00000010;  // add marks for left-to-right reading order layout
  DATE_RTLREADING   = $00000020;  // add marks for right-to-left reading order layout
  LOCALE_USE_CP_ACP = $40000000;  // use the system ACP

procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  MyTime: PSystemTime;                      // pointer to TSystemTime structure
  MySystemTime: TSystemTime;                // TSystemTime structure in memory
  MyDate: PChar;                            // pointer to output buffer
  MyDateBuffer: array[1..40] of char;       // output buffer
  MyFormat: PChar;                          // format for formatting date
begin
  {initialize pointers}
  MyDate := @MyDateBuffer;
  MyTime := @MySystemTime;

  {display the system date}
  GetDateFormat(
      LOCALE_USER_DEFAULT,     // user locale
      DATE_LONGDATE,           // long date format
      nil,                     // get the system date
      nil,                     // use local formatting
      MyDate,                  // output buffer
      40);                     // size of output buffer
  Label1.Caption := 'The System Date is ' + MyDate;

  {initialize system time structure}
  FillChar(MyTime^,SizeOf(TSystemTime),0);
  MyTime^.wYear := 1981;       // a special date
  MyTime^.wMonth :=   3;
  MyTime^.wDay :=     6;
  MyFormat := 'dddd, MMMM d, yyyy';

  GetDateFormat(LOCALE_USER_DEFAULT, 0, MyTime, MyFormat, MyDate, 40);
  Label2.Caption := 'I remember ' + MyDate;
end;
```



The System Date is Wednesday, January 30,

I remember Friday, March 6, 1981

Figure 6-10: The formatted date output

Table 6-13: GetDateFormat Locale values

Value	Description
LOCALE_SYSTEM_DEFAULT	The system's default locale.
LOCALE_USER_DEFAULT	The user's default locale.

Table 6-14: GetDateFormat dwFlags values

Value	Description
LOCALE_NOUSEROVERRIDE	Indicates that the default date format for the locale given in the Locale parameter will be used. Without this flag, any other formatting option will be used to override the locale default date format.
LOCALE_USE_CP_ACP	Uses the system ANSI code page instead of the locale code page for format translation.
DATE_SHORTDATE	Uses the short date format as defined by the regional settings. This cannot be used in combination with DATE_LONGDATE or DATE_YEARMONTH.
DATE_LONGDATE	Uses the long date format as defined by the regional settings. This cannot be used in combination with DATE_SHORTDATE or DATE_YEARMONTH.
DATE_YEARMONTH	Uses the year/month format as defined by the regional settings. This cannot be used in combination with DATE_LONGDATE or DATE_SHORTDATE.
DATE_USE_ALT_CALENDAR	If an alternate calendar exists, use it to format the date string. The date format for the alternate calendar will be used instead of any other overriding specification. With this flag set, other formatting commands will be used only if there is no default date formatting defined for the alternate calendar.
DATE_LTRREADING	Adds marks for a left-to-right reading layout. Cannot be used in combination with DATE_RTLREADING.
DATE_RTLREADING	Adds marks for a right-to-left reading layout. Cannot be used in combination with DATE_LTRREADING.

Table 6-15: GetDateFormat lpFormat values

Value	Description
d	Day of the month. Single-digit days will contain no leading zero.
dd	Day of the month. Single-digit days will contain a leading zero.
ddd	Day of the week abbreviated to three letters. The abbreviation is determined by the specified locale.
dddd	Day of the week, unabbreviated. The specified locale provides the unabbreviated day name.
M	Month with no leading zero for single-digit months.
MM	Month with a leading zero for single-digit months.
MMM	Month as a three-letter abbreviation. The abbreviation is determined by the specified locale.
MMMM	Month as its full unabbreviated name. The abbreviation is determined by the specified locale.

Value	Description
y	Last two digits of the year, with no leading zeroes for single-digit years.
yy	Last two digits of the year, with a leading zero for single-digit years.
yyyy	Year as a full four-digit number.
gg	Period/era as defined by the specified locale. This is ignored if the specified date does not have an associated era or period string.

GetOEMCP Windows.pas

Syntax

GetOEMCP: UINT; {returns an OEM code page identifier}

Description

GetOEMCP gets the current OEM code page identifier for the system. If a code page is not defined, then the default code page identifier is returned.

Return Value

If the function succeeds, it returns a 32-bit OEM code page identifier from the following table; otherwise, it returns zero.

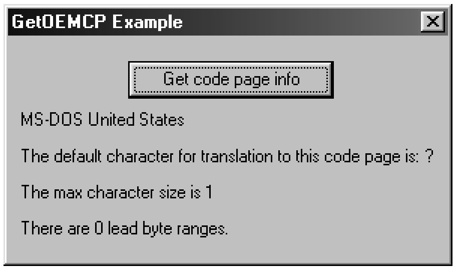
See Also

GetACP, GetCPInfo

Example

Listing 6-11: Retrieving the current OEM code page

```
procedure TForm1.Button1Click(Sender: TObject);
var
  SystemCodePage: Integer;       // holds the system code page
  SystemCPInfo: TCPinfo;         // holds code page info
  LeadByteCount: Integer;        // indicates lead byte count
  LeadX: Integer;                // loop counter
begin
  {retrieve the system code page}
  SystemCodePage := GetOEMCP;
  case SystemCodePage of
    437:  Label1.Caption := 'MS-DOS United States';
    708:  Label1.Caption := 'Arabic (ASMO 708)';
    709:  Label1.Caption := 'Arabic (ASMO 449+, BCON V4)';
    710:  Label1.Caption := 'Arabic (Transparent Arabic)';
    720:  Label1.Caption := 'Arabic (Transparent ASMO)';
    737:  Label1.Caption := 'Greek (formerly 437G)';
    775:  Label1.Caption := 'Baltic';
    850:  Label1.Caption := 'MS-DOS Multilingual (Latin I)';
    852:  Label1.Caption := 'MS-DOS Slavic (Latin II)';
    855:  Label1.Caption := 'IBM Cyrillic (primarily Russian)';
    857:  Label1.Caption := 'IBM Turkish';
    860:  Label1.Caption := 'MS-DOS Portuguese';
    861:  Label1.Caption := 'MS-DOS Icelandic';
    862:  Label1.Caption := 'Hebrew';
    863:  Label1.Caption := 'MS-DOS Canadian-French';
    864:  Label1.Caption := 'Arabic';
    865:  Label1.Caption := 'MS-DOS Nordic';
    866:  Label1.Caption := 'MS-DOS Russian (former USSR)';
    869:  Label1.Caption := 'IBM Modern Greek';
    874:  Label1.Caption := 'Thai';
    932:  Label1.Caption := 'Japan';
    936:  Label1.Caption := 'Chinese (PRC, Singapore)';
    949:  Label1.Caption := 'Korean';
    950:  Label1.Caption := 'Chinese (Taiwan, Hong Kong)';
    1361: Label1.Caption := 'Korean (Johab)';
  else
    Label1.Caption := 'The system code page is ' +IntToStr(SystemCodePage);
  end;

  {the TCPinfo parameter is a VAR parameter,
   just give the variable, not the address of the variable}
  if not GetCPInfo(SystemCodePage,SystemCPInfo)
    then Label2.Caption := 'Error in getting CP info.'
    else Label2.Caption := 'The default character for translation '
                + 'to this code page is: '
                + Char(SystemCPInfo.DefaultChar[0]);

  {display the character size}
  Label3.Caption := 'The max character size is '
                    + IntToStr(SystemCPInfo.MaxCharSize);
  {determine lead byte count}
  LeadByteCount := 0;
  for leadX := 5 downto 0 do
    if SystemCPInfo.LeadByte[2*leadx] = 0 then LeadByteCount := LeadX;

  Label4.Caption := 'There are '+IntToStr(LeadByteCount)+' lead byte ranges.';
end;
```



```
MS-DOS United States
The default character for translation to this code page is: ?
The max character size is 1
There are 0 lead byte ranges.
```

Figure 6-11: The OEM code page information

Table 6-16: GetOEMCP return values

Value	Description
437	MS-DOS United States
708	Arabic (ASMO 708)
709	Arabic (ASMO 449+, BCON V4)

Value	Description
710	Arabic (Transparent Arabic)
720	Arabic (Transparent ASMO)
737	Greek (formerly 437G)
775	Baltic
850	MS-DOS Multilingual (Latin I)
852	MS-DOS Slavic (Latin II)
855	IBM Cyrillic (primarily Russian)
857	IBM Turkish
860	MS-DOS Portuguese
861	MS-DOS Icelandic
862	Hebrew
863	MS-DOS Canadian-French
864	Arabic
865	MS-DOS Nordic
866	MS-DOS Russian (former USSR)
869	IBM Modern Greek
874	Thai
932	Japanese
936	Chinese (PRC, Singapore)
949	Korean
950	Chinese (Taiwan, Hong Kong)
1361	Korean (Johab)

GetTimeFormat   Windows.pas

Syntax

```
GetTimeFormat(
Locale: LCID;               {the locale identifier}
dwFlags: DWORD;             {formatting options}
lpTime: PSystemTime;        {a pointer to the time to be formatted}
lpFormat: PChar;            {a pointer to the time format string}
lpTimeStr: PChar;           {a pointer to a buffer for the formatted time}
cchTime: Integer            {the size of the output buffer}
): Integer;                 {returns the number of characters copied to the buffer}
```

Description

This function formats a given time according to a format string and specified options. A specific time or the system time can be used.

Parameters

Locale: The locale identifier to which the time is formatted. If the lpFormat parameter is NIL, the time is formatted according to the default time formatting for this locale. If the lpFormat parameter contains a formatting string, then the locale will only be used for formatting information not specified in the lpFormat string. Alternatively, this parameter can be set to one of the values from Table 6-17.

dwFlags: Specifies the time formatting options. This parameter can be any combination of values from Table 6-18.

lpTime: A pointer to a TSystemTime structure that contains the time to format. If this parameter is set to NIL, the current system time will be used. The TSystemTime structure is defined as:

```
TSystemTime = record
      wYear: Word;              {indicates the year}
      wMonth: Word;             {indicates the month}
      wDayOfWeek: Word;         {indicates the day of the week}
      wDay: Word;               {indicates the day of the month}
      wHour: Word;              {indicates the hour}
      wMinute: Word;            {indicates the minute)
      wSecond: Word;            {indicates the seconds}
      wMilliseconds: Word;      {indicates the milliseconds}
end;
```

Please see the FileTimeToSystemTime function for a description of this data structure.

lpFormat: A pointer to a string containing the format picture. If this parameter is NIL, the default date format for the locale will be used. This string is a formatting picture containing formatting codes from Table 6-19. The format codes are case sensitive. Any characters in a format string that are in single quotes will appear as in the output string without being used as a format specifier. A typical time of "12:45 AM" could be formatted with an lpFormat parameter of "hh':'mm ss tt". If one of the time markers ("t" or "tt") is set and the TIME_NOTIMEMARKER flag is not set in the dwFlags parameter, the time marker information will be provided based on the specified locale.

lpTimeStr: A pointer to the string buffer that receives the formatted time string.

cchTime: Indicates the size of the output buffer pointed to by the lpTimeStr parameter in characters. If this parameter is zero, the function returns the number of characters that would be required to hold the formatted output and the lpTimeStr parameter is ignored.

Return Value

If the function succeeds, it returns the number of characters copied to the lpDateStr buffer; otherwise, it returns zero. To get extended error information, call the GetLastError function. No errors are returned for a bad format string. GetTimeFormat will simply use whatever it can and produce its best output using the format provided. If "hhh" or "ttt" are provided in the format string, the values of "hh" and "tt" will be used instead.

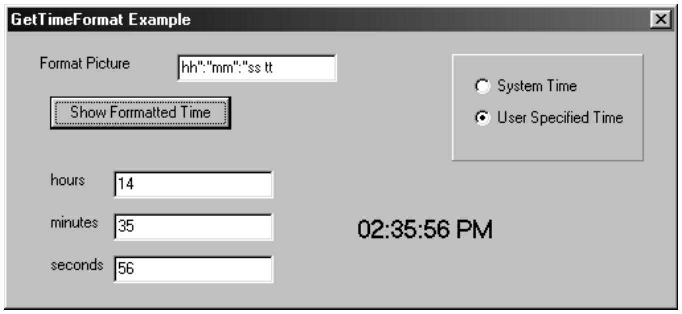
See Also

GetDateFormat

Example

Listing 6-12: Retrieving a formatted time string

```
const
  {Delphi 6 does not define these GetTimeFormat flags}
  LOCALE_USE_CP_ACP = $40000000;     // use the system ACP

procedure TForm1.Button1Click(Sender: TObject);
var
  PMySystemTime: PSystemTime;        // pointer to a TSystemTime
  TMySystemTime: TSystemTime;        // TSystemTime structure in memory
  MyOutput: array[0..20] of Char;    // output buffer
  PMyOutput: PChar;                  // pointer to output
  OutputSize: Integer;               // size of output buffer
  APIresult: Integer;                // function result for error trapping
begin
  {initialize pointers}
  PMySystemTime := @TMySystemTime;
  PMyOutput := @MyOutput;
  OutputSize := SizeOf(MyOutput);    // find size of buffer

  {determine the requested time format}
  FillChar(TMySystemTime,SizeOf(TMySystemTime),0);
  if RadioButton1.Checked then PMySystemTime := nil;
  if RadioButton2.Checked then
  begin
    TMySystemTime.wHour   := StrToInt(Edit2.Text);
    TMySystemTime.wMinute := StrToInt(Edit3.Text);
    TMySystemTime.wSecond := StrToInt(Edit4.Text);
  end;
  {get the time for the specified format}
  APIresult := GetTimeFormat(LOCALE_SYSTEM_DEFAULT,
                             0, PMySystemTime,
                             PChar(Edit1.Text),
                             PMyOutput, OutputSize);
  If (APIresult = 0) and (GetLastError = ERROR_INVALID_PARAMETER) then
    ShowMessage('Invalid Parameter');

  {display the time}
  Label1.Caption := PMyOutput;
end;
```


GetTimeFormat Example

Format Picture: hh":"mm":"ss tt

( ) System Time
(●) User Specified Time

hours: 14
minutes: 35
seconds: 56

02:35:56 PM

Figure 6-12: The formatted time output

Table 6-17: GetTimeFormat Locale values

Value	Description
LOCALE_SYSTEM_DEFAULT	The system's default locale.
LOCALE_USER_DEFAULT	The user's default locale.

Table 6-18: GetTimeFormat dwFlags values

Value	Description
LOCALE_NOUSEROVERRIDE	When provided, this forces the function to use the system default time format for the specified locale. Otherwise, when not set, the function formats the output using any overrides in the locale's default time format. This flag can be used only if the lpFormat parameter is set to NIL.
LOCALE_USE_CP_ACP	Uses the system ANSI code page instead of the locale code page for format translation.
TIME_NOMINUTESORSECONDS	Do not put minutes or seconds in the output.
TIME_NOSECONDS	Do not put seconds in the output.
TIME_NOTIMEMARKER	Do not use a time marker.
TIME_FORCE24HOURFORMAT	Use a 24-hour time format regardless of any locale settings.

Table 6-19: GetTimeFormat lpFormat values

Value	Description
h	Hours with no leading zero in a 12-hour format.
hh	Hours with leading zeroes in a 12-hour format.
H	Hours with no leading zeroes in a 24-hour format.
HH	Hours with leading zeroes in a 24-hour format.
m	Minutes with no leading zeroes.
mm	Minutes with leading zeroes.
s	Seconds with no leading zeroes.
ss	Seconds with leading zeroes.
t	Single character time marker, A for AM and P for PM.
tt	Multicharacter time marker, AM or PM.

GlobalAddAtom Windows.pas

Syntax

```
GlobalAddAtom(
lpString: PChar        {the string to add to the atom table}
): ATOM;               {returns the newly added atom}
```

Description

This function adds the specified string to the global atom table and returns the atom number. The string can be no longer than 255 characters. If the string already exists in the table, its reference count is incremented. This global atom table is shared system-wide with all other processes. Global atom tables have a set size of 37 entries. Global atoms are not automatically deleted when the application terminates. Atoms added with GlobalAddAtom must be removed with a matching call to GlobalDeleteAtom before terminating the application.

Parameters

lpString: A pointer to a null-terminated string to be added to the global atom table.

Return Value

If the function succeeds, it returns the atom number for the string that was added to the global atom table. The atom value is a 16-bit number in the range 49152 to 65535 ($C000 to $FFFF) for strings or in the range 1 to 49151 ($0001 to $BFFF) for integers. If the function fails, it returns zero. To get extended error information, call the GetLastError function.

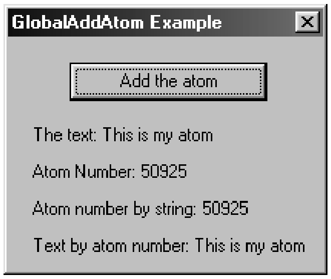
See Also

AddAtom, DeleteAtom, FindAtom, GetAtomName, GlobalDeleteAtom, GlobalFindAtom, GlobalGetAtomName, MakeIntAtom

Example

Listing 6-13: Adding a string to the global atom table

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyAtom: Atom;         // the returned atom
  AtomText: PChar;      // the string we will store
  TextTest: PChar;      // for testing search results
  AtomTest: Atom;       // for testing search results
begin
  {add the string to the global atom table}
  AtomText := 'This is my atom';
  Label1.Caption := 'The text: ' + string(AtomText);

  MyAtom := GlobalAddAtom(AtomText);
  Label2.Caption := 'Atom Number: ' + IntToStr(MyAtom);

  {search the table for atom number, given the string}
  AtomTest := GlobalFindAtom(AtomText);
  Label3.Caption := 'Atom number by string: ' + IntToStr(AtomTest);

  {search by atom number to get the string}
  TextTest := StrAlloc(256);
  GlobalGetAtomName(MyAtom, Texttest,256);
  Label4.Caption := 'Text by atom number: ' + string(TextTest);

  {clean up}
  GlobalDeleteAtom(MyAtom);
  StrDispose(TextTest);
end;
```



```
The text: This is my atom
Atom Number: 50925
Atom number by string: 50925
Text by atom number: This is my atom
```

Figure 6-13: The atom was added

GlobalDeleteAtom Windows.pas

Syntax

```
GlobalDeleteAtom(
nAtom: ATOM                    {the atom number to delete}
): ATOM;                       {returns zero or the nAtom value}
```

Description

GlobalDeleteAtom reduces the reference count for the specified atom in the global atom table by one. If the reference count for the specified atom is zero, the entry is deleted from the atom table. To make a deletion from the local atom table, use DeleteAtom.

Parameters

nAtom: The atom number to delete from the global atom table.

Return Value

If this function succeeds, it returns zero; otherwise, it returns the atom number in the nAtom parameter. To get extended error information, call the GetLastError function.

See Also

AddAtom, FindAtom, DeleteAtom, GlobalAddAtom, GlobalFindAtom

Example

Please see Listing 6-13 under GlobalAddAtom.

GlobalFindAtom Windows.pas

Syntax

```
GlobalFindAtom(
lpString: PChar          {a pointer to the string to search for in the local atom table}
): ATOM;                 {returns the atom number for the string}
```

Description

GlobalFindAtom searches the global atom table for the string pointed to by the lpString parameter and returns the atom number if it is found. The string comparison is not case sensitive. To find an atom in the local atom table, use the FindAtom function.

Parameters

lpString: A pointer to the null-terminated string to search for in the global atom table.

Return Value

If the function succeeds, it returns the atom number for the specified string; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

AddAtom, FindAtom, DeleteAtom, GlobalAddAtom, GlobalDeleteAtom

Example

Please see Listing 6-13 under GlobalAddAtom.

GlobalGetAtomName Windows.pas

Syntax

```
GlobalGetAtomName(
nAtom: ATOM;             {an atom number}
lpBuffer: PChar;         {a pointer to a buffer receiving the name}
nSize: Integer           {the length of the lpBuffer buffer}
): UINT;                 {returns the number of characters copied to the buffer}
```

Description

This function finds the string associated with the specified atom number in the global atom table. If an integer value was stored, the value will be returned in a string format. For retrieving a string from an atom number from the local atom table, use the GetAtomName function.

Parameters

nAtom: The atom number whose string is to be retrieved.

lpBuffer: A pointer to a string buffer where the function will place the results of the search. The string buffer should be large enough to receive the string value plus the null terminator.

nSize: Specifies the size of the buffer pointed to by the lpBuffer parameter.

Return Value

If the function succeeds, the buffer pointed to by the lpBuffer parameter will contain the string associated with the specified atom number, and the function returns the number of characters copied to this buffer. If the function fails, it returns zero. To get extended error information, call the GetLastError function.

See Also

AddAtom, DeleteAtom, FindAtom, GetAtomName, GlobalAddAtom, GlobalDeleteAtom, GlobalFindAtom

Example

Please see Listing 6-13 under GlobalAddAtom.

InitAtomTable Windows.pas

Syntax

```
InitAtomTable(
nSize: DWORD        {the desired number of entries in local atom table}
): BOOL;            {returns TRUE or FALSE}
```

Description

This function will initialize the local atom table to a specified number of entries. InitAtomTable does not need to be called before using any other local atom function. If InitAtomTable is not called, the local atom table defaults to a size of 37 entries. However, if the local atom table will be set to a larger size, InitAtomTable should be called before any other local atom function.

Parameters

nSize: The requested number of table entries. For optimal performance, this value should be a prime number.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE.

See Also

AddAtom, DeleteAtom, FindAtom, GetAtomName, GlobalAddAtom, GlobalDeleteAtom, GlobalFindAtom, GlobalGetAtomName

Example

Please see Listing 6-1 under AddAtom.

IsCharAlpha Windows.pas

Syntax

```
IsCharAlpha(
ch: Char            {the character to test}
): BOOL;            {returns TRUE or FALSE}
```

Description

IsCharAlpha tests a character to see if it is an alphabetic character. The language that is selected at setup or in the Control Panel will determine how the test is performed.

Parameters

ch: The character to be tested.

Return Value

If the function succeeds and the character is an alphabetic character, it returns TRUE. If the function fails or the character is not an alphabetic character, it returns FALSE.

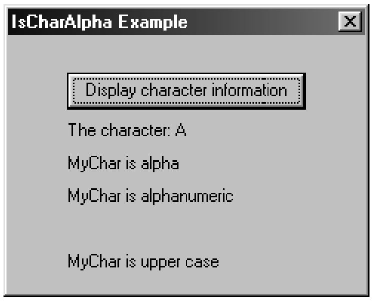
See Also

IsCharAlphaNumeric

Example

Listing 6-14: Testing character attributes

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Mychar: Char;  // holds the character to test
begin
  {initialize the test variable}
  MyChar := 'A';
  Label1.Caption := 'The character: ' + MyChar;

  {retrieve character information}
  if IsCharAlpha(MyChar)        then Label2.Caption := 'MyChar is alpha';
  if IsCharAlphaNumeric(MyChar) then Label3.Caption := 'MyChar is alphanumeric';
  if IsCharLower(MyChar)        then Label4.Caption := 'MyChar is lowercase';
  if IsCharUpper(MyChar)        then Label5.Caption := 'MyChar is uppercase';
end;
```



```
The character: A
MyChar is alpha
MyChar is alphanumeric

MyChar is upper case
```

Figure 6-14: The character attributes

IsCharAlphaNumeric ***Windows.pas***

Syntax

IsCharAlphaNumeric(
ch: Char {the character to test}
): BOOL; {returns TRUE or FALSE}

Description

IsCharAlphaNumeric tests a character to see if it is an alphabetic or a numeric character. The language that is selected at setup or in the Control Panel will determine how the test is performed.

Parameters

ch: The character to be tested.

Return Value

If the function succeeds and the character is an alphanumeric character, it returns TRUE. If the function fails or the character is not alphanumeric, it returns FALSE.

See Also

IsCharAlpha

Example

Please see Listing 6-14 under IsCharAlpha.

IsCharLower Windows.pas

Syntax

```
IsCharLower(
ch: Char                {the character to test}
): BOOL;                {returns TRUE or FALSE}
```

Description

IsCharLower tests the specified character to determine whether or not it is lowercase.

Parameters

ch: The character to be tested.

Return Value

If the function succeeds and the character is lowercase, it returns TRUE. If the function fails or the character is not lowercase, it returns FALSE.

See Also

IsCharUpper

Example

Please see Listing 6-14 under IsCharAlpha.

IsCharUpper Windows.pas

Syntax

```
IsCharUpper(
ch: Char                {the character to test}
): BOOL;                {returns TRUE or FALSE}
```

Description

IsCharUpper tests the specified character to determine whether or not it is uppercase.

Parameters

ch: The character to be tested.

Return Value

If the function succeeds and the character is uppercase, it returns TRUE. If the function fails or the character is not uppercase, it returns FALSE.

See Also

IsCharLower

Example

Please see Listing 6-14 under IsCharAlpha.

lstrcat Windows.pas

Syntax

lstrcat(	
lpString1: PChar;	{the base string and destination address}
lpString2: PChar	{the string added to the base string}
): PChar;	{returns a pointer to the concatenated string}

Description

lstrcat concatenates two null-terminated strings together, saving the concatenated string in the buffer pointed to by the lpString1 parameter. The resultant null-terminated string consists of the string pointed to by the lpString1 parameter followed by the string pointed to by the lpString2 parameter.

Parameters

lpString1: A pointer to a null-terminated string. This string must be large enough to contain both strings.

lpString2: A pointer to a null-terminated string. This string is added to the end of the string pointed to by the lpString1 parameter.

Return Value

If the function succeeds, it returns a pointer to the concatenated strings; otherwise, it returns NIL. To get extended error information, call the GetLastError function.

See Also

lstrcmp, lstrcmpi, lstrcpy, lstrlen

Example

Listing 6-15: Concatenating two strings

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyTotalString: PChar;      // holds the entire string
  MyStringToAppend: PChar;   // holds the string to append
begin
  {allocate memory for the entire string}
  MyTotalString := StrAlloc(255);
  {copy the text in edit1 to the full string}
  MyTotalString := lstrcpy(MyTotalString, PChar(Edit1.Text));

  {point the append string to the text in edit2}
  MyStringToAppend := PChar(Edit2.Text);

  {concatenate both strings}
  MyTotalString := lstrcat(MyTotalString, MyStringToAppend);

  {display the concatenated string}
  Label1.Caption := StrPas(MyTotalString);

  {dispose of allocated memory}
  StrDispose(MyTotalString);
end;
```

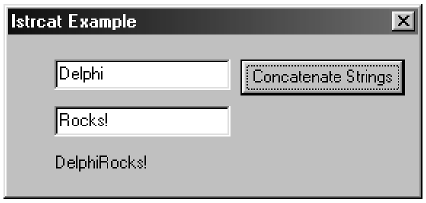


Figure 6-15: The concatenated strings

lstrcmp Windows.pas

Syntax

lstrcmp(	
lpString1: PChar;	{a pointer to the first string to compare}
lpString2: PChar	{a pointer to the second string to compare}
): Integer;	{returns the comparison results}

Description

lstrcmp is a simple string compare function. CompareString should be strongly considered instead of lstrcmp due to flexibility. lstrcmp has no options and is only interpreted in the context of the currently installed locale. If no locale was installed at setup or selected in the Control Panel, then Windows will use a default locale. The two strings are compared using a word sort comparison, similar to the comparison that is used by the CompareString function without the SORT_STRINGSORT flag. If the strings are identical up to the end of the shorter string, then the shorter string is deemed smaller in value. This function has fewer options than CompareString, but consequently, it is a little faster.

Parameters

lpString1: A pointer to the first null-terminated string in the comparison.

lpString2: A pointer to the second null-terminated string in the comparison.

Return Value

The function returns a number indicating the equality of the strings. This number will be one value from Table 6-20. This function does not indicate an error if it fails.

See Also

CompareString, lstrcat, lstrcmpi, lstrcpy, lstrlen

Example

Listing 6-16: Comparing two strings

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyResult: Integer;   // holds the result of the string compare
begin
  MyResult := lstrcmp(PChar(Edit1.Text), PChar(Edit2.Text));
  if MyResult > 0 then
  begin   // first parameter is greater
    Label1.Caption := 'GREATER';
    Label2.Caption := 'SMALLER';
    Label3.Caption := '>';
  end;
  if MyResult < 0 then
  begin
    Label1.Caption := 'SMALLER';
    Label2.Caption := 'GREATER';
    Label3.Caption := '<';
  end;
  if MyResult = 0 then
  begin
    Label1.Caption := ' equal ';
    Label2.Caption := ' equal ';
    Label3.Caption := '=';
  end;
end;
```

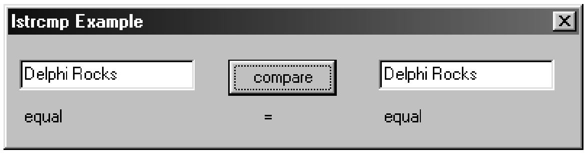


Figure 6-16: The two strings are equal

Table 6-20: lstrcmp return values

Value	Description
negative numbers	lpString1 is less than lpString2.
zero	lpString1 and lpString2 are equal.
positive numbers	lpString1 is greater than lpString2.

lstrcmpi Windows.pas

Syntax

lstrcmpi(
lpString1: PChar; {a pointer to the first string to compare}
lpString2: PChar {a pointer to the second string to compare}
): Integer; {returns the comparison results}

Description

This function behaves exactly like lstrcmp, except the string comparison is done on a case-insensitive basis.

Parameters

lpString1: A pointer to the first null-terminated string in the comparison.

lpString2: A pointer to the second null-terminated string in the comparison.

Return Value

The function returns a number indicating the equality of the strings. This number will be one value from Table 6-21. This function does not indicate an error if it fails.

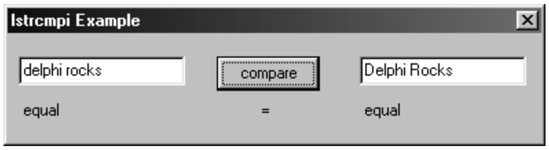
See Also

CompareString, lstrcat, lstrcmp, lstrcpy, lstrlen

Example

Listing 6-17: Comparing two strings

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyResult: Integer;    // holds the result of the comparison
begin
  MyResult := lstrcmpi(PChar(Edit1.Text), PChar(Edit2.Text));
  if MyResult > 0 then
  begin   // first parameter is greater
    Label1.Caption := 'GREATER';
    Label2.Caption := 'SMALLER';
    Label3.Caption := '>';
  end;
  if MyResult < 0 then
  begin
    Label1.Caption := 'SMALLER';
    Label2.Caption := 'GREATER';
    Label3.Caption := '<';
  end;
  if MyResult = 0 then
  begin
    Label1.Caption := ' equal ';
    Label2.Caption := ' equal ';
    Label3.Caption := '=';
  end;
end;
```



delphi rocks | = | Delphi Rocks
equal | = | equal

Figure 6-17: The strings are equal

Table 6-21: lstrcmpi return values

Value	Description
negative numbers	lpString1 is less than lpString2.
zero	lpString1 and lpString2 are equal.
positive numbers	lpString1 is greater than lpString2.

lstrcpy Windows.pas

Syntax

```
lstrcpy(
lpString1: PChar;        {a pointer to the destination buffer}
lpString2: PChar         {a pointer to the string to copy}
): PChar;                {returns a pointer to the destination buffer}
```

Description

lstrcpy copies the string pointed to by the lpString2 parameter to the buffer pointed to by the lpString1 parameter. It is a general purpose string copy routine that can be used for any null-terminated data structure regardless of length. The destination buffer must be allocated prior to calling lstrcpy.

Parameters

lpString1: A pointer to the destination string buffer. This buffer must be large enough to hold the entire string pointed to by the lpString2 parameter, including the null terminating character.

lpString2: A pointer to the string being copied.

Return Value

If the function succeeds, it returns a pointer to the destination string buffer; otherwise, it returns NIL. To get extended error information, call the GetLastError function.

See Also

lstrcat, lstrcmp, lstrcmpi, lstrlen

Example

Please see Listing 6-15 under lstrcat.

lstrlen Windows.pas

Syntax

lstrlen(	
lpString: PChar	{a pointer to a string}
): Integer;	{returns the number of characters in the string}

Description

lstrlen finds the length in characters of the string pointed to by the parameter. The null terminator is not included in this count.

Parameters

lpString: A pointer to a string for which the length is returned.

Return Value

This function returns the number of characters in the string pointed to by the lpString parameter. This function does not indicate an error upon failure.

See Also

lstrcat, lstrcmp, lstrcmpi, lstrcpy

Example

Listing 6-18: Finding the length of a string

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyLength: Integer;   // holds the string length
begin
  MyLength := lstrlen(PChar(Edit1.Text));
  Label1.Caption := 'The length of the string is ' + IntToStr(MyLength);
end;
```

MakeIntAtom Windows.pas

Syntax

MakeIntAtom(	
wInteger: WORD	{an integer value}
): PChar;	{returns an integer atom}

Description

This function converts the integer identified by the wInteger value into an atom suitable for use with the AddAtom or GlobalAddAtom functions. DeleteAtom and GlobalDeleteAtom will always succeed for integer atoms, which are not reference counted like string atoms. GetAtomName and GlobalGetAtomName will return a null-terminated string with the first character as a pound (#) character; the remaining characters are the digits of the integer passed to MakeIntAtom.

Parameters

wInteger: A 16-bit value that is converted into an integer atom.

Return Value

If the function succeeds, it returns a pointer to a null-terminated string representing the integer that is suitable for use with the AddAtom and GlobalAddAtom functions. If the function fails, it returns NIL.

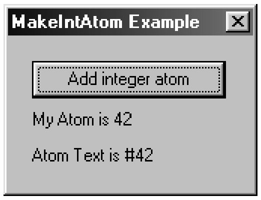
See Also

AddAtom, DeleteAtom, GlobalAddAtom, GlobalDeleteAtom

Example

Listing 6-19: Adding an integer atom

```
procedure TForm1.Button1Click(Sender: TObject);
var
  My16: word;              // 16-bit value to put in local atom table
  MyAtom: Atom;            // Atom number in local atom table
  MyString : PChar;        // Resulting string in atom table
begin
  {make space for reading atom table}
  MyString := StrAlloc(256);

  {value to store in atom table}
  My16 := 42;

  {store the 16-bit atom}
  MyAtom := AddAtom(MakeIntAtom(My16));

  {display atom information}
  Label1.Caption := 'My Atom is ' + IntToStr(MyAtom);
  GetAtomName(MyAtom, MyString, 256);
  Label2.Caption := 'Atom Text is ' + MyString;

  {clean up}
  DeleteAtom(MyAtom);
  StrDispose(MyString);
end;
```



My Atom is 42
Atom Text is #42

Figure 6-18: The integer atom

OemToChar Windows.pas

Syntax

```
OemToChar(
lpszSrc: PChar;           {a pointer to the string to translate}
lpszDst: PChar            {a pointer to the translated string}
): BOOL;                  {always returns TRUE}
```

Description

OemToChar translates the given string from the OEM character set to an ANSI or wide-character (Unicode) string.

Parameters

lpszSrc: A pointer to the string containing OEM characters.

lpszDst: A pointer to the translated string. If the destination character set is an ANSI set (single-byte characters), then the source and destination can be the same string. In this case, the translation will be performed in place. If the destination character set is a Unicode (double-byte) character set, there must be a separate buffer for lpszDst.

Return Value

This function always returns TRUE.

See Also

CharToOem, CharToOemBuff, OemToCharBuff

Example

Please see Listing 6-3 under CharToOem.

OemToCharBuff Windows.pas

Syntax

```
CharToOemBuff(
lpszSrc: PChar;               {a pointer to the string to translate}
lpszDst: PChar                {a pointer to the translated string}
cchDstLength: DWORD           {the number of characters to translate}
): BOOL;                      {always returns TRUE}
```

Description

CharToOemBuff translates the specified number of characters from the OEM source string to the destination string that can be an ANSI string (single-byte) or Unicode (double-byte) character set.

Parameters

lpszSrc: A pointer to the OEM source string that is to be translated.

lpszDst: A pointer to the destination string. If the destination character set is an ANSI set (single-byte characters), then the source and destination can be the same string. In this case, the translation will be performed in place. If the destination character set is a Unicode (double-byte) character set, there must be a separate buffer for lpszDst.

cchDstLength: Specifies the number of characters in the OEM source string to translate.

Return Value

This function always returns TRUE.

See Also

CharToOem, CharToOemBuff, OemToChar

Example

Please see Listing 6-3 under CharToOem.

ToAscii Windows.pas

Syntax

```
ToAscii(
uVirtKey: UINT;                  {a virtual key code to be translated}
uScanCode: UINT;                 {the hardware keyboard scan code}
const KeyState: TKeyboardState;  {the keyboard state}
lpChar: PChar;                   {a pointer to a buffer receiving the translated key}
uFlags: UINT;                    {menu active flags}
): Integer;                      {returns a conversion code}
```

Description

ToAscii translates a virtual key code and scan state into a Windows character, using the input language and the system keyboard layout.

Parameters

uVirtKey: The virtual key code to be translated to a Windows character.

uScanCode: The hardware scan code of the key to be translated. The high-order bit is set if the key is not pressed. The value of the uVirtKey parameter is the primary code used for translation. uScanCode is used to distinguish between a keypress and a key release and for translating ALT+*number key* combinations.

KeyState: A pointer to a 256-byte array indicating the current keyboard state. Each byte indicates the state for a single key. The most significant bit is set if the key is down. If the low bit is set, the CAPS LOCK key is toggled on. SCROLL LOCK and NUM LOCK are ignored.

lpChar: A pointer to a string buffer receiving the translated Windows character.

uFlags: Specifies if a menu is active. A value of 1 indicates that a menu is active; 0 indicates no menu is active.

Return Value

The function returns one value from the following table. ToAscii does not indicate an error upon failure.

See Also

OemKeyScan, VkKeyScan

Example

Listing 6-20: Converting a virtual key code

```
procedure TForm1.Button1Click(Sender: TObject);
var
  WindowsChar: array[0..1] of Char;  // translation will be put here
  VirtualKey: Integer;               // virtual key code to be translated
  ScanCode: Integer;                 // keyboard scan code, to detect keypress
  MyKeyState: TKeyboardState;        // array of key states for each key
  ReturnCode: Integer;               // API return code
begin
  VirtualKey :=  Ord('A');           // ascii char or a virtual key code
  ScanCode := $1E;                   // letter a or A on keyboard hardware

  {setting MyKeyState entries to $00 makes a lowercase a}
  Fillchar(MyKeyState, SizeOf(MyKeyState),$00);
  ReturnCode := ToAscii(VirtualKey, ScanCode, MyKeyState, @WindowsChar, 0);
  Label1.Caption := 'Return Code is ' + IntToStr(ReturnCode);
  Label2.Caption := 'Translated Character is ' + WindowsChar;
end;
```

Table 6-22: ToAscii return values

Value	Description
negative number	The specified key is a dead key (accent or diacritic key).
0	The specified key has no translation for the specified state of the keyboard.
1	One Windows character was produced by the translation and copied to the output buffer.
2	Two Windows characters were produced by the translation. This means an accent or diacritic key was required, and no Windows character was available for that combination.

wvsprintf *Windows.pas*

Syntax

wvsprintf(	
Output: PChar;	{a pointer to an output string buffer}
Format: PChar;	{a pointer to the format string}
arglist: va_list	{a pointer to a list of arguments}
): Integer;	{returns the number of characters copied to the output buffer}

Description

This function places a string of formatted text into the buffer pointed to by the Output parameter, according to the format string and variables given as parameters. Values are placed into the argument list specified by the arglist parameter, which are then processed by wvsprintf and inserted into the output according to what is specified in the format string.

Parameters

Output: A pointer to a string buffer receiving the formatted text. The calling process must allocate this buffer.

Format: A pointer to the format string that has the information on how the output is to be presented. It will generally contain one or more format specifiers, each of which will use a data element from the arglist parameter. Each format specifier will begin with a percent sign (%) followed by additional codes from Table 6-23. The general format for a format specifier is:

```
%[-][#][0][width][.precision]type
```

The brackets indicate optional fields. The simplest specifier would be %s, which would take the string for the next PChar argument in the arglist parameter and put it in place of the format specifier in the output buffer. If the % symbol is not followed by any of the codes listed in Table 6-24, the next character is generated as output. For example, %% would produce a single percent sign as output. Some of the specifiers are case sensitive. Those that are not case sensitive will have both representations shown in Table 6-24. Table 6-23 shows the purpose of each of the fields in a format specifier that can exist in a format string. Table 6-24 shows all the possible entries in the type field, which is the last field in the specifier.

arglist: A pointer to the array of parameters for the format specifiers. Each format specifier in the Format parameter that uses a variable will use a parameter in the arglist. The size of the array will be whatever is needed to provide enough 4-byte elements to satisfy the format specifiers. Each array element can be a PChar or an integer value up to 4 bytes in length. See the FormatMessage function for more information and examples on using argument arrays.

Return Value

If the function succeeds, the returned value is the number of characters copied to the output buffer, not counting the final null termination character. If the function fails, it returns a number that is less than the size of the minimum expected output buffer. It is not adequate to check the result against the length of the format string, because it is possible that the final output is shorter than the format string. In the integer example (in the Button2Click procedure in Listing 6-21), the format string contains format specifications like “%1ld,” which is four characters. The substituted value is only one character. Therefore, the error check must compare against the length of the format string, less six characters (two instances of three characters less). To get extended error information, call the GetLastError function.

See Also

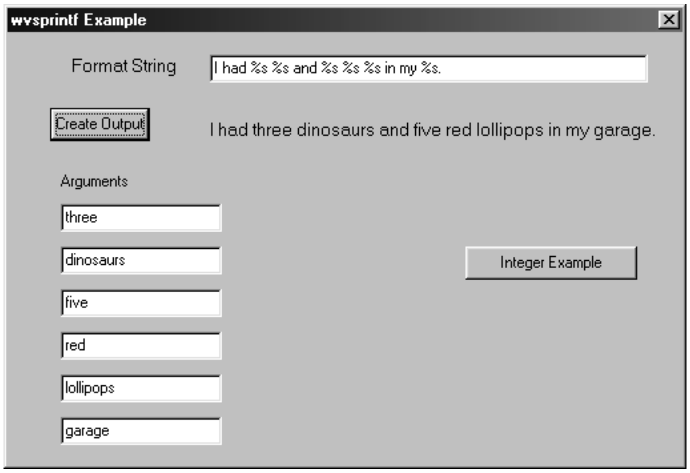
FormatMessage, GetDateFormat, GetTimeFormat

Example

Listing 6-21: Formatting an array of arguments

```
{Delphi does not link in LocalHandle, so we must do it explicitly}
function LocalHandle(Mem: Pointer): HLOCAL; stdcall;

implementation

{link in LocalHandle}
function LocalHandle; external kernel32 name 'LocalHandle';

procedure MyErrorHandler(ErrorNumber:integer);
var
  ErrorMessage: PChar;        // pointer to message area.
  MyMemoryHandle: HLOCAL;
begin
  {display the system error message if something went wrong}
  FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM or
       FORMAT_MESSAGE_ALLOCATE_BUFFER,
       nil,
       ErrorNumber,    // ID of error message to fetch
       0,
       @ErrorMessage,  // points to PChar pointer
       0,
       nil);
  ShowMessage('wvsprintf error: ' + ErrorMessage);
  MyMemoryHandle := LocalHandle(ErrorMessage);
  if (LocalFree(MyMemoryHandle)  0) then
    ShowMessage('Error freeing memory');
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  MyOutputPtr: PChar;                   // points to output buffer
  MyOutput: array[1..100] of Char;      // output buffer
  APIresult: Integer;                   // function result
  MyArguments: array[0..1] of Integer;  // list of arguments
  MyFormat: string;                     // format spec / template
begin
  {initialize the format string and parameters}
  MyOutputPtr := @MyOutput;
  MyFormat := 'There were %1ld dogs and %1ld birds in my house.';
  MyArguments[0] := 3;
  MyArguments[1] := 2;

  {display the formatted string}
  APIresult := wvsprintf(MyOutputPtr, PChar(MyFormat), @MyArguments);
  Label1.Caption := MyOutputPtr;

  {if there was a problem, display the error message}
  if APIresult < (length(MyFormat)-6) then
```

```
    MyErrorHandler(GetLastError);
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  MyOutputPtr: PChar;                    // points to output buffer
  MyOutput: array[1..200] of Char;       // output buffer
  APIresult: Integer;                    // result of function call
  MyArguments: array[0..5] of PChar;     // argument array of strings
  MyStrings: array[0..5] of string;      // data space for strings
  MyFormatPtr: PChar;                    // points to format string
  FormatStr: string;                     // format string
begin
  {initialize the format string and format arguments}
  MyOutputPtr    := @MyOutput;
  FormatStr      := Edit1.Text;
  MyFormatPtr    := Pchar(FormatStr);
  MyStrings[0]   := Edit2.Text;
  MyArguments[0] := (PChar(MyStrings[0]));
  MyStrings[1]   := Edit3.Text;
  MyArguments[1] := (PChar(MyStrings[1]));
  MyStrings[2]   := Edit4.Text;
  MyArguments[2] := (PChar(MyStrings[2]));
  MyStrings[3]   := Edit5.Text;
  MyArguments[3] := (PChar(MyStrings[3]));
  MyStrings[4]   := Edit6.Text;
  MyArguments[4] := (PChar(MyStrings[4]));
  MyStrings[5]   := Edit7.Text;
  MyArguments[5] := (PChar(MyStrings[5]));

  {display the formatted string}
  APIresult := wvsprintf(MyOutputPtr, MyFormatPtr, @MyArguments);
  Label1.Caption := MyOutputPtr;

  {if there was a problem, display the error message}
  if APIresult < (strlen(MyFormatPtr)) then
    MyErrorHandler(GetLastError);
end;
```



I had %s %s and %s %s %s in my %s.

I had three dinosaurs and five red lollipops in my garage.

three
dinosaurs
five
red
lollipops
garage

Figure 6-19: The formatted string

Table 6-23: wvsprintf Format specifier values

Value	Description
-	Pad the output to the right and justify to the left. If this is omitted, pad to the left and justify to the right.
#	Prefix hexadecimal output with 0x or 0X, uppercase or lowercase according to the specification of the type parameter.
0	Any padding is to be done with zeroes. If omitted, pad with spaces.
width	Total minimum width of the output. Padding will be performed according to the previous fields, but the output will never be truncated. If omitted, all output will be generated subject to the precision field.
.precision	Copy the specified digits to output for numerical specifications or the number of characters for string specifications. The value is not truncated. This serves as a padding specification, not an absolute width specification. If this field is omitted, or given as a 0 or as a single period, the precision is 1.
type	Specification on how to perform the output for strings, characters, or numbers. See the following table for possible field entries.

Table 6-24: wvsprintf Format type values

Type	Description
C	Single character. Values of zero are ignored. Wchar for Unicode programs and char otherwise.
C	Single character. Char for Unicode programs and Wchar otherwise. Consider using the c (lowercase) instead of C (uppercase) for general use.
d	Signed decimal integer. Same as i.
hc, hC	Single character. Always interpreted as a char, even in Unicode programs.
hs, hS	Null-terminated string (PChar), even in Unicode programs.
i	Signed decimal integer. Same as d.
lc, lC	Single character. Ignores zero values. Always treated as Wchar, regardless of whether or not the program is a Unicode program.
ld, li	Long signed decimal integer.
ls, lS, s	Wide null-terminated string (lpwstr), PChar to a wide string, even when not in a Unicode program.
lu	Interpreted as a long unsigned integer.
lx, lX	Long unsigned hexadecimal integer, in lowercase or uppercase.
S	Null-terminated string, always single-byte character width even in Unicode programs.
u	Interpreted as unsigned integer.
x, X	Unsigned hexadecimal integer, lowercase or uppercase.

Chapter 7

Clipboard Manipulation Functions

Sharing information between applications allows a Windows user to be more productive. Information could be prepared in one application, such as a graphical drawing program, and then copied and pasted into another application, such as a word processor. The ability to copy and paste information from one application to another is a standard expectation of Windows users, and the clipboard manipulation functions provide the means to implement this functionality.

Clipboard Internals

The clipboard is little more than a system-hosted environment for data storage and retrieval. It can be thought of as a group of storage bins, with each bin holding a handle to information in a specific format. The clipboard supports a number of predefined formats for holding information of various types, such as graphics and text, but the application can also define its own formats. The predefined formats are listed under the GetClipboardData and SetClipboardData functions.

To place information on the clipboard, an application must first call the OpenClipboard function. It should then call the EmptyClipboard function, which deletes all information on the clipboard in every format and assigns the ownership of the clipboard to the window passed in the hWndNewOwner parameter of the OpenClipboard function. Next, the application calls SetClipboardData, passing it a flag indicating the format associated with the type of data and a handle to the data itself. The CloseClipboard function completes the operation. When an application wishes to retrieve data from the clipboard, it uses the GetClipboardData function, passing it a flag indicating the format for the requested data. If data exists on the clipboard in this format, a handle to the data is returned.

When an application places data on the clipboard, it should place the data in as many formats as possible that make sense for the data. This allows a broader range of applications to retrieve and make use of the information. For example, a word processor may place text on the clipboard in a proprietary clipboard format that it uses internally. Placing text on the clipboard in the CF_TEXT and CF_UNICODETEXT formats in addition to its proprietary format will allow other Windows applications, such as Notepad, to retrieve the information in a format that it understands. If the text was placed on the clipboard in only the proprietary format, only the word processor would be able to retrieve the text.

Conversion Formats

Windows provides data conversions from one format to another for many of the predefined clipboard formats. This conversion is performed by the system on the fly and is transparent to the application. When data is available on the clipboard in a format for which there exists a conversion, an application can simply request the data by calling the GetClipboardData function and passing it the desired format without any further processing. The following table lists the format conversions available by platform.

Table 7-1: Clipboard format conversions

Clipboard Formats	Conversion Format	Supported Platform
CF_BITMAP	CF_DIB	Windows 95/98/Me, Windows NT/2000/XP
CF_DIB	CD_BITMAP	Windows 95/98/Me, Windows NT/2000/XP
CF_DIB	CF_PALETTE	Windows 95/98/Me, Windows NT/2000/XP
CF_METAFILEPICT	CF_ENHMETAFILE	Windows 95/98/Me, Windows NT/2000/XP
CF_ENHMETAFILE	CF_METAFILEPICT	Windows 95/98/Me, Windows NT/2000/XP
CF_OEMTEXT	CF_UNICODETEXT	Windows NT/2000/XP
CF_OEMTEXT	CF_TEXT	Windows 95/98/Me, Windows NT/2000/XP
CF_TEXT	CF_OEMTEXT	Windows 95/98/Me, Windows NT/2000/XP
CF_TEXT	CF_UNICODETEXT	Windows NT/2000/XP
CF_UNICODETEXT	CF_OEMTEXT	Windows NT/2000/XP
CF_UNICODETEXT	CF_TEXT	Windows NT/2000/XP

Delayed Rendering

Storing large amounts of data on the clipboard, such as multiple complex formats, GDI objects, or graphics, can take time and will eat away at resources. Fortunately, Windows allows an application to perform something called delayed rendering. To initiate delayed rendering, an application passes a zero in the hMem parameter of the SetClipboardData function. The clipboard will appear to have data in the specified format, but when an application requests this data, Windows will send the WM_RENDERFORMAT or WM_RENDERALLFORMATS message to the clipboard owner (the application that commenced the delayed rendering). Inside the handlers for these messages, the application must prepare the information and then call the SetClipboardData function, this time passing the handle to the information. This technique is useful if the application supports many different formats of complex data, as it does not have to take the time to render every format to the clipboard. The following example demonstrates delayed rendering.

Listing 7-1: Delayed rendering of information

```
type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Button1: TButton;
    Button2: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    procedure WMDestroyClipboard(var Msg: TWMDestroyClipboard);
              message WM_DESTROYCLIPBOARD;
    procedure WMRenderFormat(var Msg: TWMRenderFormat);
              message WM_RENDERFORMAT;
  end;

  {our proprietary data format}
  ProprietaryData = record
    Number: Integer;
    Text: array[0..255] of char;
  end;

var
  Form1: TForm1;
  NewFormatID: UINT;      // holds the application-defined clipboard format ID
  DataHandle: THandle;    // a handle to our data

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  {register an application-defined clipboard format}
  NewFormatID := RegisterClipboardFormat('New Format Example');
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  {Open the clipboard}
  OpenClipboard(Form1.Handle);

  {empty the clipboard contents. note that it is important to call the
   OpenClipboard function passing it the form's handle, and then call
   EmptyClipboard so that the form will be set as the clipboard owner.
   only the clipboard owner will receive the WM_RENDERFORMAT message.}
  EmptyClipboard;

  {indicate that our proprietary clipboard format is available, but
   set the data for delayed rendering}
  SetClipboardData(NewFormatID, 0);

  {close the clipboard}
  CloseClipboard;
end;
```

```
procedure TForm1.WMRenderFormat(var Msg: TWMRenderFormat);
var
  DataPointer: ^ProprietaryData;  // a pointer to our data structure
begin
  if Msg.Format = NewFormatID then
  begin
    {allocate enough memory to hold our data structure}
    DataHandle := GlobalAlloc(GMEM_DDESHARE or GMEM_MOVEABLE,
                  SizeOf(ProprietaryData));

    {retrieve a pointer to the allocated memory}
    DataPointer := GlobalLock(DataHandle);

    {set the members of the structure with the supplied values}
    DataPointer^.Number := StrToInt(Edit1.Text);
    StrCopy(DataPointer^.Text, PChar(Edit2.Text));

    {unlock the handle to the data}
    GlobalUnlock(DataHandle);

    {copy our proprietary data to the clipboard}
    SetClipboardData(NewFormatID, DataHandle);

    {indicate that the message was handled}
    Msg.Result := 0;
  end
  else
    inherited;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  RetrievedData: THandle;         // a handle to data
  DataPointer: ^ProprietaryData;  // a pointer to our data type
begin
  {open the clipboard}
  OpenClipboard(Form1.Handle);

  {retrieve the data in our application-defined format}
  RetrievedData := GetClipboardData(NewFormatID);

  {get a pointer to the data}
  DataPointer := GlobalLock(RetrievedData);

  {display the data values}
  Label1.Caption := IntToStr(DataPointer^.Number);
  Label2.Caption := string(DataPointer^.Text);

  {unlock the data handle}
  GlobalUnlock(RetrievedData);

  {close the clipboard}
  CloseClipboard;
end;
```

```
procedure TForm1.WMDestroyClipboard(var Msg: TWMDestroyClipboard);
begin
  {the clipboard is being emptied, so free our data}
  GlobalFree(DataHandle);
  inherited;
end;
```

Clipboard Viewers

A window can register itself with the system as a clipboard viewer by calling the SetClipboardViewer function. A clipboard viewer's function is to simply display the contents of the clipboard; it should never modify the clipboard in any way, nor should it leave the data it retrieves from the clipboard in a locked state. Windows maintains a list of clipboard viewers called the clipboard viewer chain, and when a window calls the SetClipboardViewer function, it is placed at the beginning of this clipboard viewer list. A clipboard viewer will receive the WM_DRAWCLIPBOARD message when the contents of the clipboard changes and the WM_CHANGECBCHAIN message when a window has been added to or removed from the clipboard viewer chain. When the clipboard viewer receives one of these messages, it should pass the message on to the other clipboard viewers in the chain. See the SetClipboardViewer function for an example of registering a clipboard viewer.

Delphi vs. the Windows API

Delphi has an incredible object available called TClipboard. It is located in the Clipbrd unit and is accessed through the global object Clipboard. This object provides almost all of the functionality that you would ever need for an application that makes use of the clipboard in a standard way. However, there is much more you can do with the clipboard than can be done through Delphi's TClipboard object. Such functionality includes delayed rendering or setting a clipboard viewer, and this can only be done through the Windows API.

Clipboard Manipulation Functions

The following clipboard manipulation functions are covered in this chapter.

Table 7-2: Clipboard manipulation functions

Function	Description
ChangeClipboardChain	Removes a window from the clipboard viewer chain.
CloseClipboard	Closes the clipboard.
CountClipboardFormats	Returns the number of formats for which there is data available on the clipboard.
EmptyClipboard	Empties the clipboard and assigns clipboard ownership.
EnumClipboardFormats	Returns the clipboard formats for which there is data available on the clipboard.
GetClipboardData	Retrieves clipboard data for the specified format.

Function	Description
GetClipboardFormatName	Retrieves the name of a user-defined clipboard format.
GetClipboardOwner	Retrieves a handle to the clipboard owner window.
GetClipboardViewer	Retrieves a handle to the first window in the clipboard viewer chain.
GetOpenClipboardWindow	Retrieves a handle to the window that currently has the clipboard opened.
GetPriorityClipboardFormat	Returns the first clipboard format from an array of clipboard formats for which there is data available on the clipboard.
IsClipboardFormatAvailable	Indicates if data is currently available in the specified format.
OpenClipboard	Opens the clipboard for modification.
RegisterClipboardFormat	Registers a user-defined clipboard format with the system.
SetClipboardData	Copies data onto the clipboard in the specified format.
SetClipboardViewer	Registers a window as a clipboard viewer and places it in the clipboard viewer chain.

ChangeClipboardChain Windows.pas

Syntax

```
ChangeClipboardChain(
hWndRemove: HWND;                {a handle to the window to remove}
hWndNewNext:HWND                 {a handle to the next window}
): BOOL;                         {returns TRUE or FALSE}
```

Description

This function removes the window identified by the hWndRemove parameter from the chain of clipboard viewer windows. The window identified by the hWndNewNext parameter replaces the previous window's position in the viewer chain. The hWndNew-Next parameter should be set to the value returned by the call to SetClipboardViewer that inserted the hWndRemove window into the chain. ChangeClipboardChain causes the WM_CHANGECBCHAIN message to be sent to the first window in the clipboard viewer chain.

Parameters

hWndRemove: A handle to the window being removed from the clipboard viewer chain.

hWndNewNext: A handle to the next window in the clipboard viewer chain.

Return Value

This function returns the result of processing the WM_CHANGECBCHAIN message by the subsequent windows in the clipboard viewer chain. Typically, these windows return FALSE when processing this message. This function will return TRUE if there is only one window in the clipboard viewer chain.

See Also

SetClipboardViewer, WM_CHANGECBCHAIN

Example

Please see Listing 7-6 under SetClipboardViewer.

CloseClipboard Windows.pas

Syntax

```
CloseClipboard: BOOL;              {returns TRUE or FALSE}
```

Description

This function closes a clipboard after an application has opened it by calling the OpenClipboard function. The clipboard must be closed before other applications can access it.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetOpenClipboardWindow, OpenClipboard

Example

Please see Listing 7-5 under SetClipboardData and Listing 7-6 under SetClipboard-Viewer.

CountClipboardFormats Windows.pas

Syntax

```
CountClipboardFormats: Integer;   {returns the number of formats currently on the
                                    clipboard}
```

Description

This function returns the number of clipboard formats that are currently on the clipboard.

Return Value

If the function succeeds, it returns the number of clipboard formats that are currently on the clipboard; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

EnumClipboardFormats, RegisterClipboardFormat

Example

Please see Listing 7-2 under EnumClipboardFormats.

EmptyClipboard Windows.pas

Syntax

```
EmptyClipboard: BOOL;      {returns TRUE or FALSE}
```

Description

This function empties the clipboard, freeing any data stored in the clipboard. The clipboard must first be opened by calling the OpenClipboard function. Once the clipboard is emptied, this function assigns clipboard ownership to the window passed in the OpenClipboard function. If the application passes zero as the window handle to the OpenClipboard function, this function will succeed, but the clipboard will not be assigned an owner.

Return Value

If this function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

OpenClipboard, SetClipboardData, WM_DESTROYCLIPBOARD

Example

Please see Listing 7-5 under SetClipboardData.

EnumClipboardFormats Windows.pas

Syntax

```
EnumClipboardFormats(
format: UINT                  {an available clipboard format ID}
): UINT;                      {returns the next available clipboard format ID}
```

Description

This function returns the clipboard data formats that are currently available on the clipboard. Clipboard formats are enumerated in the order in which they were placed on the clipboard. To begin enumeration, set the format parameter to zero and call the function. This will return the first available clipboard format. Then, call the EnumClipboardFormats function again, setting the format parameter to the value returned from the previous call. This should continue in a loop until EnumClipboardFormats returns zero. The clipboard must be opened with a call to the OpenClipboard function before enumeration can begin. For clipboard formats that have an automatic type conversion, the clipboard format will be enumerated, followed by the clipboard formats to which it can be converted.

Parameters

format: Specifies a clipboard format identifier. For a list of possible clipboard format identifiers, see the SetClipboardData function.

Return Value

If the function succeeds, it returns the next available clipboard format identifier; otherwise, it returns zero. To get extended error information, call the GetLastError function. If there are no more clipboard format identifiers left to enumerate, the function will succeed but will return zero, in which case GetLastError will return ERROR_SUCCESS.

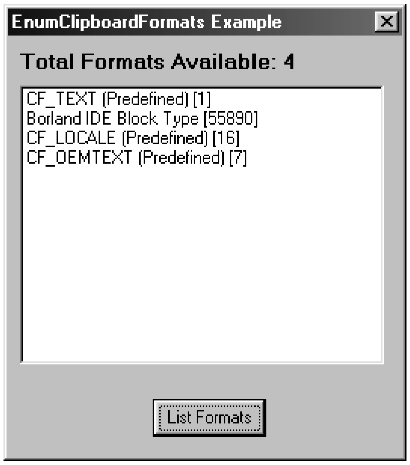
See Also

CountClipboardFormats, GetClipboardData, OpenClipboard, RegisterClipboardFormat, SetClipboardData

Example

Listing 7-2: Enumerating available clipboard formats

```
procedure TForm1.Button1Click(Sender: TObject);
const
  {an array defining all of the predefined clipboard format names}
  PredefinedClipboardNames: array[1..17] of string = ('CF_TEXT', 'CF_BITMAP',
                                  'CF_METAFILEPICT', 'CF_SYLK', 'CF_DIF',
                                  'CF_TIFF', 'CF_OEMTEXT', 'CF_DIB',
                                  'CF_PALETTE', 'CF_PENDATA', 'CF_RIFF',
                                  'CF_WAVE', 'CF_UNICODETEXT',
                                  'CF_ENHMETAFILE', 'CF_HDROP', 'CF_LOCALE',
                                  'CF_MAX');
var
  FormatID: UINT;                     // holds a clipboard format ID
  FormatName: array[0..255] of char;  // holds a clipboard format name
  Len: Integer;                       // the length of a clipboard format name
begin
  {clear out the list box}
  ListBox1.Items.Clear;

  {display the number of formats on the clipboard}
  Label1.Caption := 'Total Formats Available: '+IntToStr(CountClipboardFormats);

  {open the clipboard}
  OpenClipboard(0);

  {retrieve the first available clipboard format}
  FormatID := EnumClipboardFormats(0);

  {retrieve all clipboard formats}
  while (FormatID  0 ) do
  begin
    {get the name of the clipboard format. note that this will only return a format name
     if it is a registered format, not one of the predefined formats}
    Len := GetClipboardFormatName(FormatID,FormatName,255);
    {if len is equal to zero then it's a predefined format}
    if Len = 0 then
      ListBox1.Items.Add(PredefinedClipboardNames[FormatID]+' (Predefined)'+
                         ' [' + IntToStr(FormatID)+ ']')
    else
      {otherwise it contains a registered format name}
      ListBox1.Items.Add(FormatName+' [' + IntToStr(FormatID)+ ']');

    {retrieve the next available clipboard format}
    FormatID:=EnumClipboardFormats(FormatID);
  end;

  {we are done with the enumeration, so close the clipboard}
  CloseClipboard;
end;
```



```
CF_TEXT (Predefined) [1]
Borland IDE Block Type [55890]
CF_LOCALE (Predefined) [16]
CF_OEMTEXT (Predefined) [7]
```

Figure 7-1: The clipboard formats available after copying text from Delphi's text editor

GetClipboardData Windows.pas

Syntax

```
GetClipboardData(
uFormat: UINT            {a clipboard format identifier}
): THandle;              {a handle to the clipboard data}
```

Description

This function retrieves data from the clipboard in the format specified by the uFormat parameter. This data is retrieved in the form of a global memory handle. This handle belongs to the clipboard, and it should not be freed or left locked by the application. Consequently, the application should make a copy of the data immediately upon receiving the handle. If there is data on the clipboard in a format for which the operating system provides a data conversion, this data can be retrieved in the alternative format, and the system converts it on the fly (i.e., CF_OEMTEXT data can be retrieved as CF_TEXT).

Parameters

uFormat: Specifies a clipboard format identifier. This can be one value from the following table.

Return Value

If the function succeeds, it returns a handle to the data retrieved from the clipboard in the specified format; otherwise, it returns zero. To get extended error information, call the GetLastError function.

Example

Please see Listing 7-5 under SetClipboardData.

Table 7-3: GetClipboardData uFormat values

Value	Description
CF_BITMAP	A handle to a bitmap.
CF_DIB	A handle to a memory object containing a device-independent bitmap in the form of a TBitmapInfo data structure, followed by the bitmap image bits.
CF_DIF	Data in the form of Software Art's data interchange format.
CF_DSPBITMAP	Bitmap data in a private format unique to the application.
CF_DSPENHMETAFILE	Enhanced metafile data in a private format unique to the application.
CF_DSPMETAFILEPICT	A handle to a memory object identifying a TMetafilePict data structure that contains a metafile in a private format unique to the application.
CF_DSPTEXT	A handle to text data in a private format unique to the application.
CF_ENHMETAFILE	A handle to an enhanced metafile.
CF_GDIOBJFIRST through CF_GDIOBJLAST	A handle to an application-defined GDI object. This handle is not the actual handle to the GDI object but the handle returned from GlobalAlloc used to allocate memory for the object. The data identified by these values is not automatically freed when the clipboard is emptied; this is the responsibility of the clipboard owner.
CF_HDROP	A handle identifying files that have been dragged and dropped from the Windows Explorer.
CF_LOCALE	A handle to a locale identifier associated with the text on the clipboard. This can be used to determine the character set used when the text was copied to the clipboard. **Windows NT/2000 and later**: Windows uses the code page associated with the CF_LOCALE handle to convert the text from the CF_TEXT format to the CF_UNICODE format.

Value	Description
CF_METAFILEPICT	A handle to a memory object identifying a TMetafilePict data structure that contains a metafile.
CF_OEMTEXT	Text containing characters from the OEM character set. Each line ends with a carriage return and line feed, and a null terminating character identifies the end of the data.
CF_OWNERDISPLAY	Indicates that the clipboard owner must update and display the clipboard viewer window. The clipboard owner will receive the following messages: WM_ASKCBFORMATNAME, WM_HSCROLLCLIPBOARD, WM_PAINTCLIPBOARD, WM_SIZECLIPBOARD, and WM_VSCROLLCLIPBOARD.
CF_PALETTE	Data in the form of a color palette. When an application places a bitmap on the clipboard, it should also place the bitmap's palette on the clipboard.
CF_PENDATA	Data used for Microsoft Pen Computing extensions.
CF_PRIVATEFIRST through CF_PRIVATELAST	Private clipboard format data. Windows does not free the value associated with this type of format; the clipboard owner must free these resources in response to the WM_DESTROYCLIPBOARD message.
CF_RIFF	Complex audio data.
CF_SYLK	Data in the Microsoft Symbolic Link format.
CF_TEXT	Regular ANSI text. Each line ends with a carriage return and line feed, and a null terminating character identifies the end of the data.
CF_WAVE	Audio data in a standard Windows wave format.
CF_TIFF	An image in a tagged image file format.
CF_UNICODETEXT	**Windows NT only**: Text in Unicode format. Each line ends with a carriage return and line feed, and a null terminating character identifies the end of the data.

The TMetafilePict data structure is defined as:

```
TMetafilePict = packed record
      mm: Longint;                {the mapping mode}
      xExt: Longint;              {the width of the metafile}
      yExt: Longint;              {the height of the metafile}
      hMF: HMETAFILE;             {a handle to the metafile}
end;
```

For a description of this data structure, see the SetClipboardData function.

GetClipboardFormatName Windows.pas

Syntax

```
GetClipboardFormatName(
format: UINT;               {a clipboard format identifier}
lpszFormatName: PChar;      {a pointer to a buffer that receives the format name}
cchMaxCount: Integer        {the size of the lpszFormatName buffer}
): Integer;                 {returns the number of characters copied to the buffer}
```

Description

This function retrieves the name of a registered clipboard format, storing it in the buffer pointed to by the lpszFormatName parameter. This function only retrieves format names for clipboard formats registered with the RegisterClipboardFormat function and returns a zero for any predefined clipboard formats.

Parameters

format: Specifies a user-defined clipboard format identifier returned from the RegisterClipboardFormat function.

lpszFormatName: A pointer to a buffer receiving the name of the registered clipboard format.

cchMaxCount: Specifies the maximum number of characters to copy to the buffer pointed to by the lpszFormatName parameter. Any characters over this specified limit will be truncated.

Return Value

If the function succeeds, it returns the number of characters copied to the buffer pointed to by the lpszFormatName parameter. If the function fails, or the specified format is one of the predefined clipboard formats, it returns zero. To get extended error information, call the GetLastError function.

See Also

EnumClipboardFormats, RegisterClipboardFormat

Example

Please see Listing 7-2 under EnumClipboardFormats.

GetClipboardOwner Windows.pas

Syntax

```
GetClipboardOwner: HWND;               {returns the handle to a window}
```

Description

This function retrieves the handle to the window that owns the clipboard. The clipboard owner is generally the window that last placed data onto the clipboard. The EmptyClipboard function can reassign the clipboard owner to zero, indicating that the clipboard is not currently owned. The clipboard can contain data if it does not have an owner.

Return Value

If the function succeeds, it returns a handle to the window that currently owns the clipboard. If the function fails, or the clipboard does not have an owner, the function returns zero. To get extended error information, call the GetLastError function.

See Also

EmptyClipboard, GetClipboardViewer

Example

Please see Listing 7-6 under SetClipboardViewer.

GetClipboardViewer Windows.pas

Syntax

```
GetClipboardViewer: HWND;                  {returns a handle to a window}
```

Description

This function returns the handle to the first clipboard viewer window in the clipboard viewer chain.

Return Value

If this function succeeds, it returns the handle to the first window in the clipboard viewer chain. If the function fails, or there are no clipboard viewers, it returns zero. To get extended error information, call the GetLastError function.

See Also

GetClipboardOwner, SetClipboardViewer

Example

Please see Listing 7-6 under SetClipboardViewer.

GetOpenClipboardWindow Windows.pas

Syntax

```
GetOpenClipboardWindow: HWND;  {returns a handle to a window}
```

Description

This function returns the handle of the window that has opened the clipboard but has not yet closed it.

Return Value

If this function succeeds, it returns the handle of the window that currently has the clipboard opened. If the function fails, the clipboard is not currently open, or the clipboard is open but not associated with any window, this function returns zero. To get extended error information, call the GetLastError function.

Example

Listing 7-3: Retrieving the window opening the clipboard

```
procedure TForm1.Button1Click(Sender: TObject);
var
 TheWindow: HWND;                    // holds the handle of the window
 WindowText: array[0..255] of char;  // holds the text of the window
begin
  {open the clipboard}
  OpenClipboard(Handle);

  {retrieve the handle of the window which currently has the clipboard open}
  TheWindow := GetOpenClipboardWindow;

  {get the caption of the window}
  GetWindowText(TheWindow, WindowText, 255);

  {display the caption}
  Button1.Caption := 'Currently, '+string(WindowText)+' has the clipboard open';

  {close the clipboard}
  CloseClipboard;
end;
```

GetPriorityClipboardFormat   Windows.pas

Syntax

```
GetPriorityClipboardFormat(
var paFormatPriorityList;    {a pointer to an array of clipboard format identifiers}
cFormats: Integer            {the number of entries in the paFormatPrioityList array}
): Integer;                  {returns the first clipboard format containing data}
```

Description

This function returns the clipboard format identifier for the first format in the array pointed to by the paFormatPriorityList parameter for which data is available on the clipboard. The values in the paFormatPriorityList array should be arranged in the order of importance.

Parameters

paFormatPriorityList: A pointer to an array of clipboard format identifiers. Please see the SetClipboardData function for a list of predefined clipboard format identifiers.

cFormats: Specifies the number of entries in the array pointed to by the paFormatPriorityList parameter.

Return Value

If the function succeeds, it returns the first clipboard format identifier in the list for which data is available on the clipboard. If the clipboard contains data but not in any format listed, the function returns –1. If the function fails, or the clipboard is empty, it returns zero. To get extended error information, call the GetLastError function.

See Also

CountClipboardFormats, EnumClipboardFormats, GetClipboardFormatName, IsClipboardFormatAvailable, RegisterClipboardFormat, SetClipboardData

Example

Please see Listing 7-4 under IsClipboardFormatAvailable.

IsClipboardFormatAvailable Windows.pas

Syntax

```
IsClipboardFormatAvailable(
format: UINT                        {a clipboard format identifier}
): BOOL;                            {returns TRUE or FALSE}
```

Description

This function determines if the clipboard currently contains data in the format specified by the format parameter. If an application has a Paste menu item, then it can use this function to determine whether the Paste menu should be disabled if the application supports only specific clipboard formats.

Parameters

format: Specifies a clipboard format identifier. For a list of possible clipboard format identifiers, see the SetClipboardData function.

Return Value

If the function succeeds and the clipboard contains data in the specified format, the function returns TRUE. If the function fails, or the clipboard does not contain data in the specified format, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

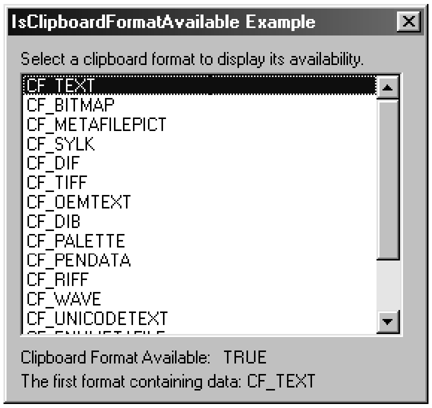
CountClipboardFormats, EnumClipboardFormats, GetPriorityClipboardFormat, OpenClipboard, RegisterClipboardFormat, SetClipboardData

Example

Listing 7-4: Interrogating clipboard format availability

```
var
  {a list of predefined clipboard formats}
  ClipboardFormats: array[1..17] of integer = (CF_TEXT, CF_BITMAP,
                                  CF_METAFILEPICT, CF_SYLK, CF_DIF, CF_TIFF,
                                  CF_OEMTEXT, CF_DIB, CF_PALETTE, CF_PENDATA,
                                  CF_RIFF, CF_WAVE, CF_UNICODETEXT,
                                  CF_ENHMETAFILE, CF_HDROP, CF_LOCALE,
                                  CF_MAX);

procedure TForm1.ListBox1Click(Sender: TObject);
begin
```

```
  {retrieve the format ID of the first available format}
  if GetPriorityClipboardFormat(ClipboardFormats, 22)>0 then
    Label4.Caption := 'The first format containing data: '+ListBox1.Items[
                      GetPriorityClipboardFormat(ClipboardFormats, 22)-1];;

  {determine if the selected clipboard format is available}
  if IsClipboardFormatAvailable(ClipboardFormats[ListBox1.ItemIndex+1]) then
    Label3.Caption := 'TRUE'
  else
    Label3.Caption := 'FALSE';
end;
```


IsClipboardFormatAvailable Example

Select a clipboard format to display its availability.

```
CF_TEXT
CF_BITMAP
CF_METAFILEPICT
CF_SYLK
CF_DIF
CF_TIFF
CF_OEMTEXT
CF_DIB
CF_PALETTE
CF_PENDATA
CF_RIFF
CF_WAVE
CF_UNICODETEXT
```

Clipboard Format Available: TRUE

The first format containing data: CF_TEXT

Figure 7-2: This clipboard format is available

OpenClipboard Windows.pas

Syntax

OpenClipboard(
hWndNewOwner: HWND {the handle of the window opening the clipboard}
): BOOL; {returns TRUE or FALSE}

Description

This function opens the clipboard, preparing it for examination or modification. The clipboard can only be opened by one window at a time. The clipboard must be closed by calling the CloseClipboard function before another window can have access to it.

Parameters

hWndNewOwner: A handle to the window that will be associated with the opened clipboard. If this parameter is set to zero, the clipboard will be opened but will not be associated with a window.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CloseClipboard, EmptyClipboard

Example

Please see Listing 7-5 under SetClipboardData and Listing 7-6 under SetClipboardViewer.

RegisterClipboardFormat Windows.pas

Syntax

```
RegisterClipboardFormat(
lpszFormat: PChar             {a pointer to a null-terminated string}
): UINT;                      {returns the new clipboard format identifier}
```

Description

This function registers an application-defined clipboard format and returns a value in the range $C000 through $FFFF. This new format identifier can be used to place application-specific data onto the clipboard. If the registered format already exists, this function simply returns the clipboard format identifier of the registered format, allowing two applications that registered the same format to share data through the clipboard.

Parameter:

lpszFormat: A pointer to a null-terminated case-sensitive string containing the name of the new clipboard format.

Return Value

If the function succeeds, it returns a new clipboard format identifier; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

CountClipboardFormats, EnumClipboardFormats, GetClipboardFormatName

Example

Please see Listing 7-5 under SetClipboardData.

SetClipboardData Windows.pas

Syntax

```
SetClipboardData(
uFormat: UINT;          {a clipboard format identifier}
hMem: THandle           {a handle to the data being copied to the clipboard}
): THandle;             {returns the handle to the data}
```

Description

This function copies the data identified by the hMem parameter onto the clipboard in the format specified by the uFormat parameter. The window copying the data must open the clipboard using the OpenClipboard function and should empty the clipboard with the EmptyClipboard function before the SetClipboardData function is called.

However, if the application is responding to a WM_RENDERFORMAT or WM_RENDERALLFORMATS message, it must not call the OpenClipboard function and can directly call SetClipboardData.

Parameters

uFormat: Specifies the clipboard format identifier for the format of the data being copied to the clipboard. This parameter can be either a user-defined clipboard format identifier, as returned by the RegisterClipboardFormat function, or one of the predefined clipboard formats listed in Table 7-4.

hMem: A handle to the data being copied to the clipboard. If this handle identifies a memory object, the memory object must have been allocated by the GlobalAlloc function using the flags GMEM_MOVEABLE and GMEM_DDESHARE. If this parameter is set to zero, the clipboard will indicate that data of the specified format exists, although it has not been placed onto the clipboard. When an application tries to retrieve this data from the clipboard, the application that copied the data will receive either a WM_RENDERFORMAT or WM_RENDERALLFORMATS message. The application must process these messages, calling SetClipboardData with a real value in the hMem parameter.

Return Value

If the function succeeds, it returns the handle of the data being copied to the clipboard; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

CloseClipboard, EmptyClipboard, GetClipboardData, GlobalAlloc*, OpenClipboard, RegisterClipboardFormat, WM_RENDERFORMAT, WM_RENDERALLFORMATS

Example

Listing 7-5: Setting and retrieving clipboard data

```
{our proprietary data format}
  ProprietaryData = record
    Number: Integer;
    Text: array[0..255] of char;
  end;

var
  Form1: TForm1;
  NewFormatID: UINT;      // holds the application-defined clipboard format ID
  DataHandle: THandle;    // a handle to our data

implementation

procedure TForm1.FormCreate(Sender: TObject);
begin
  {register an application-defined clipboard format}
  NewFormatID := RegisterClipboardFormat('New Format Example');
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  DataPointer: ^ProprietaryData;  // a pointer to our data structure
begin
  {allocate enough memory to hold our data structure}
  DataHandle := GlobalAlloc(GMEM_DDESHARE or GMEM_MOVEABLE,
                SizeOf(ProprietaryData));

  {retrieve a pointer to the allocated memory}
  DataPointer := GlobalLock(DataHandle);

  {set the members of the structure with the supplied values}
  DataPointer^.Number := StrToInt(Edit1.Text);
  StrCopy(DataPointer^.Text, PChar(Edit2.Text));

  {unlock the handle to the data}
  GlobalUnlock(DataHandle);

  {Open the clipboard}
  OpenClipboard(Form1.Handle);

  {empty the clipboard contents and assign Form1 as the clipboard owner}
  EmptyClipboard;

  {copy our proprietary data to the clipboard}
  SetClipboardData(NewFormatID, DataHandle);

  {close the clipboard}
  CloseClipboard;
end;

procedure TForm1.WMDestroyClipboard(var Msg: TWMDestroyClipboard);
begin
  {the clipboard is being emptied, so free our data}
  GlobalFree(DataHandle);
  inherited;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  RetrievedData: THandle;         // a handle to data
  DataPointer: ^ProprietaryData;  // a pointer to our data type
begin
  {open the clipboard}
  OpenClipboard(Form1.Handle);

  {retrieve the data in our application-defined format}
  RetrievedData := GetClipboardData(NewFormatID);

  {get a pointer to the data}
  DataPointer := GlobalLock(RetrievedData);

  {display the data values}
  Label1.Caption := IntToStr(DataPointer^.Number);
  Label2.Caption := string(DataPointer^.Text);
```

```
    {unlock the data handle}
    GlobalUnlock(RetrievedData);

    {close the clipboard}
    CloseClipboard;
  end;
```

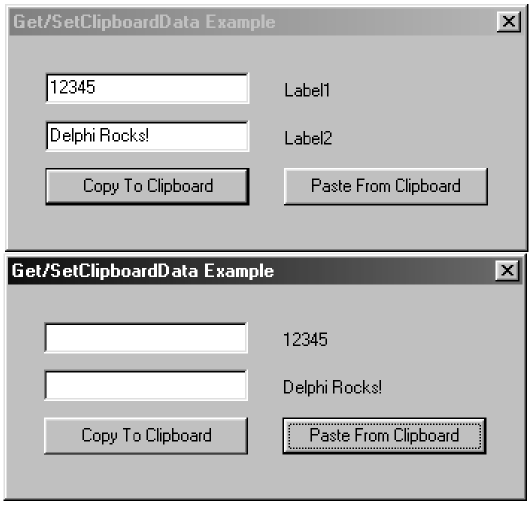


Figure 7-3: Sharing user-defined data between applications through the clipboard

Table 7-4: SetClipboardData uFormat values

Value	Description
CF_BITMAP	A handle to a bitmap.
CF_DIB	A handle to a memory object containing a device-independent bitmap in the form of a TBitmapInfo data structure, followed by the bitmap image bits.
CF_DIF	Data in the form of Software Art's data interchange format.
CF_DSPBITMAP	Bitmap data in a private format unique to the application.
CF_DSPENHMETAFILE	Enhanced metafile data in a private format unique to the application.
CF_DSPMETAFILEPICT	A handle to a memory object identifying a TMetafilePict data structure that contains a metafile in a private format unique to the application.
CF_DSPTEXT	A handle to text data in a private format unique to the application.
CF_ENHMETAFILE	A handle to an enhanced metafile.
CF_GDIOBJFIRST through CF_GDIOBJLAST	A handle to an application-defined GDI object. This handle is not the actual handle to the GDI object but the handle returned from GlobalAlloc used to allocate memory for the object. The data identified by these values is not automatically freed when the clipboard is emptied; this is the responsibility of the clipboard owner.
CF_HDROP	A handle identifying files that have been dragged and dropped from the Windows Explorer.

Value	Description
CF_LOCALE	A handle to a locale identifier associated with the text on the clipboard. This can be used to determine the character set used when the text was copied to the clipboard. **Windows NT/2000 and later**: Windows uses the code page associated with the CF_LOCALE handle to cover the text from the CF_TEXT format to the CF_UNICODE format.
CF_METAFILEPICT	A handle to a memory object identifying a TMetafilePict data structure that contains a metafile.
CF_OEMTEXT	Text containing characters from the OEM character set. Each line ends with a carriage return and line feed, and a null terminating character identifies the end of the data.
CF_OWNERDISPLAY	Indicates that the clipboard owner must update and display the clipboard viewer window. The clipboard owner will receive the following messages: WM_ASKCBFORMATNAME, WM_HSCROLLCLIPBOARD, WM_PAINTCLIP- BOARD, WM_SIZECLIPBOARD, and WM_VSCROLL-CLIPBOARD.
CF_PALETTE	Data in the form of a color palette. When an application places a bitmap on the clipboard, it should also place the bitmap's palette on the clipboard.
CF_PENDATA	Data used for Microsoft Pen Computing extensions.
CF_PRIVATEFIRST through CF_PRIVATELAST	Private clipboard format data. Windows does not free the value associated with this type of format; the clipboard owner must free these resources in response to the WM_DESTROYCLIPBOARD message.
CF_RIFF	Complex audio data.
CF_SYLK	Data in the Microsoft Symbolic Link format.
CF_TEXT	Regular ANSI text. Each line ends with a carriage return and line feed, and a null terminating character identifies the end of the data.
CF_WAVE	Audio data in a standard Windows wave format.
CF_TIFF	An image in a tagged image file format.
CF_UNICODETEXT	**Windows NT only**: Text in Unicode format. Each line ends with a carriage return and line feed, and a null terminating character identifies the end of the data.

The TMetafilePict data structure is defined as:

```
TMetafilePict = packed record
    mm: Longint;              {the mapping mode}
    xExt: Longint;            {the width of the metafile}
    yExt: Longint;            {the height of the metafile}
    hMF: HMETAFILE;           {a handle to the metafile}
end;
```

mm: Specifies the mapping mode in which the metafile was originally drawn.

xExt: Specifies the width of the rectangle within which the metafile was drawn in units corresponding to the specified mapping mode.

yExt: Specifies the height of the rectangle within which the metafile was drawn in units corresponding to the specified mapping mode.

hMF: A handle to the memory-based metafile.

SetClipboardViewer Windows.pas

Syntax

```
SetClipboardViewer(
hWndNewViewer: HWND    {a handle to the new clipboard viewer window}
): HWND;               {returns a handle to the next viewer window in the chain}
```

Description

This function adds the window identified by the hWndNewViewer parameter to the chain of clipboard viewer windows. A clipboard viewer window receives the WM_DRAWCLIPBOARD message when the clipboard contents change and the WM_CHANGECBCHAIN message when another clipboard window is added to or removed from the clipboard viewer chain. These messages must be sent to the next window in the chain, identified by the value returned from the call to SetClipboard-Viewer, after they have been processed by the current clipboard viewer. When no longer needed, the clipboard viewer window must remove itself from the clipboard viewer chain by calling the ChangeClipboardChain function.

Parameters

hWndNewViewer: A handle to the new clipboard viewer window being added to the clipboard viewer chain.

Return Value

If the function succeeds, it returns the handle to the next window in the clipboard viewer chain. If the function fails, or there are no more windows in the clipboard viewer chain, the function returns zero. To get extended error information, call the GetLastError function.

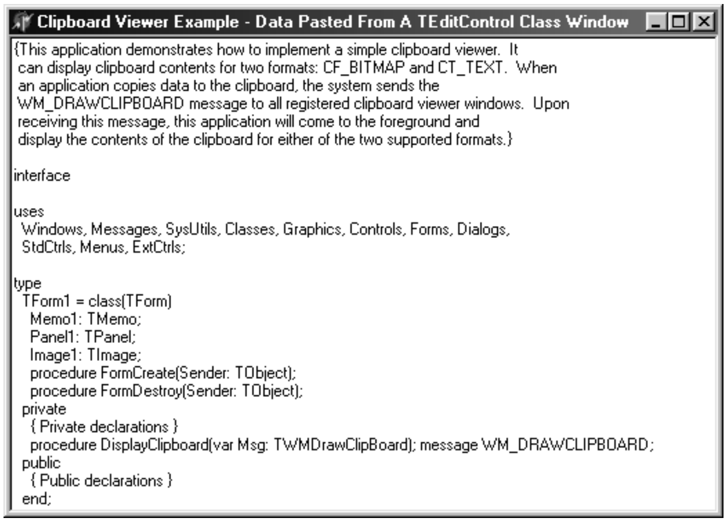
See Also

ChangeClipboardChain, GetClipboardViewer, WM_CHANGECBCHAIN, WM_DRAWCLIPBOARD

Example

Listing 7-6: Viewing the clipboard contents

```
TForm1 = class(TForm)
    Memo1: TMemo;
    Panel1: TPanel;
    Image1: TImage;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
    procedure DisplayClipboard(var Msg: TWMDrawClipBoard); message WM_DRAWCLIPBOARD;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  hNextClipboardViewerWindow: HWND;  // holds the next window in the chain

implementation

procedure TForm1.DisplayClipboard(var Msg: TWMDrawClipBoard);
var
  hClipboardData: THandle;                   // a handle to clipboard data
  lpText: PChar;                             // a pointer to text
  ClassName: array[0..255] of char;          // holds a window class name
  OwnerWindow: HWND;                         // a handle to the clipboard owner
begin
  {this example can render data in the form of text or bitmaps. if
   this format is available on the clipboard, then continue}
  if IsClipboardFormatAvailable(CF_TEXT) or
  IsClipboardFormatAvailable(CF_BITMAP) then
  begin
    {bring the window into the foreground}
    SetForegroundWindow(Form1.Handle);

    {retrieve the class name of the window that put the data onto the clipboard}
    OwnerWindow := GetClipboardOwner;
    GetClassName(OwnerWindow, ClassName, 255);

    {display the window owner class name}
    Form1.Caption := 'Clipboard Viewer Example - Data Pasted From A '+
                     string(ClassName)+' Class Window';

    {open the clipboard for examination}
    OpenClipboard(Form1.Handle);

    {if the data placed on the clipboard was text...}
    if IsClipboardFormatAvailable(CF_TEXT)then
    begin
      {...retrieve a global handle to the text}
      hClipboardData := GetClipboardData(CF_TEXT);
      if hClipboardData = 0 then
        raise Exception.Create('Error getting clipboard data');
      {convert the global handle into a pointer}
      lpText := GlobalLock(hClipboardData);

      {hide the bitmap display surface, making the memo visible}
      Panel1.Visible := FALSE;

      {display the copied text}
      SetWindowText(Memo1.Handle, lpText);

      {unlock the global handle, as we do not own it}
      GlobalUnLock(hClipboardData);
    end;
    {if the data placed on the clipboard was a bitmap...}
    if IsClipboardFormatAvailable(CF_BITMAP) then
    begin
      {...retrieve the bitmap handle}
      hClipboardData:=GetClipboardData(CF_BITMAP);
      if (hClipboardData = 0)  then
        raise Exception.Create('Error getting clipboard data');

      {show the bitmap display surface, making the memo invisible}
      Panel1.Visible := TRUE;

      {assign the bitmap to the image}
      Image1.Picture.Bitmap.Handle := hClipboardData;

      {display the copied bitmap}
      Image1.Repaint;
    end;

    {close the clipboard}
    CloseClipboard;
  end;

  {send the message to the next clipboard viewer in the chain}
  SendMessage(hNextClipboardViewerWindow, WM_DRAWCLIPBOARD, 0, 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
var
   PreviousViewer: HWND;              // a handle to the previous viewer
   WindowText: array[0..255] of char; // holds the window caption
begin
   {empty the entire clipboard contents}
   OpenClipboard(Form1.Handle);
   EmptyClipboard;
   CloseClipboard;

   {retrieve the clipboard viewer window, and display its caption}
   PreviousViewer := GetClipboardViewer;
   if PreviousViewer>0 then
   begin
      GetWindowText(PreviousViewer, WindowText, 255);
      ShowMessage('Previous clipboard viewer was: '+string(WindowText));
   end
   else
      ShowMessage('No previous clipboard viewer is installed.');

   {register this window as a clipboard viewer}
   hNextClipboardViewerWindow := SetClipboardViewer(Form1.Handle);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
   {remove the application window from the chain of clipboard viewers}
   ChangeClipboardChain(Form1.Handle, hNextClipboardViewerWindow);
end;
```


Clipboard Viewer Example - Data Pasted From A TEditControl Class Window

```
{This application demonstrates how to implement a simple clipboard viewer.  It
 can display clipboard contents for two formats: CF_BITMAP and CT_TEXT.  When
 an application copies data to the clipboard, the system sends the
 WM_DRAWCLIPBOARD message to all registered clipboard viewer windows.  Upon
 receiving this message, this application will come to the foreground and
 display the contents of the clipboard for either of the two supported formats.}

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Menus, ExtCtrls;

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Panel1: TPanel;
    Image1: TImage;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
    procedure DisplayClipboard(var Msg: TWMDrawClipBoard); message WM_DRAWCLIPBOARD;
  public
    { Public declarations }
  end;
```

Figure 7-4: The clipboard viewer

Chapter 8

System Information Functions

At some point, certain applications will need to query the system to determine various aspects about the operating system it is running on, the machine hardware, etc. This chapter covers a broad range of information query and modification functions, such as retrieving the computer name, the current local time, startup information, the Windows version, and environment variables.

Accessibility Features

Just about any system-wide parameter a user can modify through a control panel applet is available to a Windows application. Everything from the size of a window border to the wallpaper selected for the desktop can be queried or modified by the SystemParametersInfo function. This function also makes a number of accessibility features available to the application. These accessibility features provide alternative forms of user interface communication to users who are physically challenged in one way or another. For example, Windows has an accessibility feature called SoundSentry. This feature allows the system to display a visual indicator when a sound has been generated, thus alerting a hearing-impaired user that the application has emitted audible feedback. The following example demonstrates turning this accessibility feature on.

Note: You must have accessibility features installed before this example will work. If accessibility features are not installed, the call to SystemParametersInfo with SPI_SETSOUNDSENTRY (or SPI_GETSOUNDSENTRY) will simply fail and return FALSE.

Listing 8-1: Using the SoundSentry accessibility feature

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  {output a sound through the internal speaker}
  MessageBeep(0);
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  SoundInfo: TSoundSentry;       // holds the sound sentry options
begin
```

```
  {initialize the sound sentry options to turn it on}
  with SoundInfo do
  begin
    cbSize := SizeOf(TSoundSentry);
    dwFlags := SSF_SOUNDSENTRYON;
    iFSTextEffect := SSTF_DISPLAY;
    iFSTextEffectMSec := 500;
    iFSTextEffectColorBits := clRed;
    iFSGrafEffect := SSGF_DISPLAY;
    iFSGrafEffectMSec := 500;
    iFSGrafEffectColor := clRed;
    iWindowsEffect := SSWF_DISPLAY;
    iWindowsEffectMSec := 500;
    lpszWindowsEffectDLL := nil;
    iWindowsEffectOrdinal := 0
  end;

  {turn on the sound sentry for visual indication of sounds}
  if SystemParametersInfo(SPI_SETSOUNDSENTRY, SizeOf(TSoundSentry),
                          @SoundInfo, 0) then
    Label1.Caption := 'SoundSentry turned on'
  else
    Label1.Caption := 'SystemParametersInfo call failed, accessibility ' +
                      'features not installed.'
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
var
  SoundInfo: TSoundSentry;       // holds the sound sentry options
begin
  {initialize the sound sentry options to turn it off}
  with SoundInfo do
  begin
    cbSize := SizeOf(TSoundSentry);
    dwFlags := SSF_SOUNDSENTRYON;
    iFSTextEffect := SSTF_NONE;
    iFSTextEffectMSec := 500;
    iFSTextEffectColorBits := clRed;
    iFSGrafEffect := SSGF_NONE;
    iFSGrafEffectMSec := 500;
    iFSGrafEffectColor := clRed;
    iWindowsEffect := SSWF_NONE;
    iWindowsEffectMSec := 500;
    lpszWindowsEffectDLL := nil;
    iWindowsEffectOrdinal := 0
  end;

  {turn off sound sentry when the form closes}
  SystemParametersInfo(SPI_SETSOUNDSENTRY, SizeOf(TSoundSentry), @SoundInfo, 0);
end;
```

SystemParametersInfo could be used to create some very interesting utilities outside of the control panel. For example, the following application demonstrates how to change the desktop wallpaper.

Listing 8-2: Changing the desktop wallpaper

```
procedure TForm1.FileListBox1Click(Sender: TObject);
begin
  {display a preview of the selected image}
  Image1.Picture.LoadFromFile(FileListBox1.FileName);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  {set the selected image as the desktop wallpaper. the user profile will
   be updated with the new wallpaper setting}
  if FileListBox1.ItemIndex>-1 then
    SystemParametersInfo(SPI_SETDESKWALLPAPER, 0, PChar(FileListBox1.FileName),
                         SPIF_SENDCHANGE);
end;
```

Delphi vs. the Windows API

There is a plethora of information to be garnered from the system concerning all manner of settings, and while Delphi provides a modicum of support for some of this functionality, applications will need to use the Windows API to take full advantage of what the system has to offer. While some of these functions may be somewhat complex, most are relatively easy to implement, and even the complex functions are used in a rather straightforward manner. Fortunately, Delphi applications have access to a wide range of informational API calls, and using the functions in this chapter will allow a developer to query or set almost any system level setting available.

System Information Functions

The following system information functions are covered in this chapter.

Table 8-1: System information functions

Function	Description
ExpandEnvironmentStrings	Expands an environment variable string with its defined value.
FreeEnvironmentStrings	Frees an environment block returned from GetEnvironmentStrings.
GetCommandLine	Retrieves the command line used to launch the application.
GetComputerName	Retrieves the network computer name.
GetDiskFreeSpaceEx	Retrieves information on disk space.
GetDriveType	Retrieves a specified drive type, such as fixed or removable.
GetEnvironmentStrings	Retrieves a block of environment variable strings for the current process.
GetEnvironmentVariable	Retrieves the value of a single environment variable.
GetLocaleInfo	Retrieves information on the specified locale.

Function	Description
GetLocalTime	Retrieves the local time.
GetLogicalDrives	Retrieves the drives available to the machine.
GetLogicalDriveStrings	Retrieves the names of the drives available to the machine.
GetStartupInfo	Retrieves the startup information for the application's main window.
GetSystemDefaultLangID	Retrieves the system default language identifier.
GetSystemDefaultLCID	Retrieves the system default locale identifier.
GetSystemDirectory	Retrieves the Windows system directory.
GetSystemInfo	Retrieves system hardware information.
GetSystemTime	Retrieves the current system time.
GetSystemTimeAsFileTime	Retrieves the current system time in a file system time format.
GetTimeZoneInformation	Retrieves time zone information concerning standard and daylight savings time.
GetUserDefaultLangID	Retrieves the user-defined default language identifier.
GetUserDefaultLCID	Retrieves the user-defined default locale identifier.
GetUserName	Retrieves the logged-on network username.
GetVersionEx	Retrieves the Windows version.
GetVolumeInformation	Retrieves information on the specified volume.
GetWindowsDirectory	Retrieves the Windows directory.
IsProcessorFeaturePresent	Determines if certain processor features are available.
SetComputerName	Sets the network computer name.
SetEnvironmentVariable	Sets the value of a single environment variable.
SetLocaleInfo	Sets the specified locale information.
SetLocalTime	Sets the local time.
SetSystemTime	Sets the system time.
SetTimeZoneInformation	Sets time zone information concerning standard and daylight savings time.
SetVolumeLabel	Sets the specified volume's label.
SystemParametersInfo	Retrieves or modifies a number of system-wide parameters.
VerLanguageName	Retrieves the name of the specified language.

ExpandEnvironmentStrings Windows.pas

Syntax

```
ExpandEnvironmentStrings(
lpSrc: PChar;            {a string that contains the variable to expand}
lpDst: PChar;            {buffer to receive string}
nSize: DWORD             {the maximum size of buffer}
): DWORD;                {returns the number of bytes copied to the buffer}
```

Description

This function is used to expand an environment variable string. This function will also return the size of the new expanded string.

Parameters

lpSrc: A pointer to a null-terminated string containing the unexpanded environment variable. This string can contain one or more environment variable references. Each reference in the string is expanded and the resulting string is returned. These variable strings take the form of %variable name%, where the variable name is an environment variable.

lpDst: A pointer to a buffer that receives the new expanded string. If this parameter is set to NIL, the function returns the required size of the buffer to hold the expanded environment string.

nSize: The maximum size of the buffer. If the lpDst parameter is set to NIL, set this parameter to zero.

Return Value

If the function succeeds, it returns the number of characters copied to the buffer; otherwise, it returns zero. To get extended error information, call the GetLastError function.

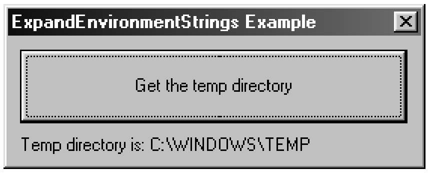
See Also

GetEnvironmentStrings, GetEnvironmentVariable

Example

Listing 8-3: Expanding an environment variable.

```
Procedure TForm1.Button1Click(Sender:Tobject);
var
  ExpandedStr: array[0..255] of char;  // holds the expanded environment string
begin
  {expand the %TEMP% environment string}
  ExpandEnvironmentStrings('Temp directory is: %TEMP%', ExpandedStr,
                           SizeOf(ExpandedStr));

  {display the expanded string}
  Label1.Caption := StrPas(ExpandedStr);
end;
```



Temp directory is: C:\WINDOWS\TEMP

Figure 8-1: The expanded environment variable

FreeEnvironmentStrings Windows.pas

Syntax

FreeEnvironmentStrings(	
p1: PChar	{a pointer to the environment strings block to free}
): BOOL;	{returns TRUE or FALSE}

Description

This function frees a block of environment variable strings as returned by the GetEnvironmentStrings function.

Parameters

p1: A pointer to a block of environment variable strings as returned by the GetEnvironmentStrings function.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetEnvironmentStrings

Example

Please see Listing 8-7 under GetEnvironmentStrings.

GetCommandLine Windows.pas

Syntax

GetCommandLine: PChar	{returns the command line string}

Description

The GetCommandLine function is used to get the command line that was used to start the program. This includes any command line parameters. The command line is returned in the form of a null-terminated string.

Return Value

If the function succeeds, it returns a pointer to the null-terminated command line string; otherwise, it returns NIL.

See Also

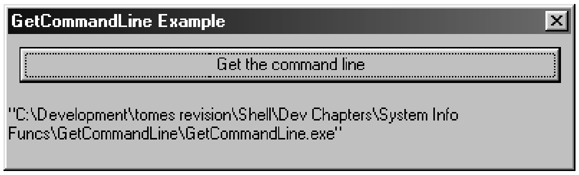
CreateProcess*

Example

Listing 8-4: Retrieving the command line

```
procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
  {retrieve the command line}
  Label1.Caption := StrPas(GetCommandLine);
end;
```


GetCommandLine Example

Get the command line

"C:\Development\tomes revision\Shell\Dev Chapters\System Info Funcs\GetCommandLine\GetCommandLine.exe"

Figure 8-2: The command line

GetComputerName Windows.pas

Syntax

```
GetComputerName(
lpBuffer: PChar;          {a pointer to a buffer that receives the computer name}
var nSize: DWORD          {the size of the lpBuffer buffer}
): BOOL;                  {returns TRUE or FALSE}
```

Description

This function retrieves the Windows networking computer name for the current system. The function also can retrieve the required size of the buffer to store the computer name. When the nSize parameter is set to zero and the lpBuffer parameter is set to NIL, the function will return the required size of the lpBuffer in the nSize parameter.

Parameters

lpBuffer: A pointer to a null-terminated string buffer receiving the computer name.

nSize: A variable that specifies the maximum length of the buffer pointed to by the lpBuffer parameter, in characters. This value should be at least MAX_COMPUTER-NAME_LENGTH in size.

Return Value

If the function succeeds, it returns TRUE and the variable pointed to by the nSize parameter receives the number of characters copied to the buffer. If the function fails, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetUserName, SetComputerName

Example

Please see Listing 8-22 under SetComputerName.

GetDiskFreeSpaceEx SysUtils.pas

Syntax

```
GetDiskFreeSpaceEx(
Directory: PChar;         {a pointer to the root path string}
var FreeAvailable: TLargeInteger;        {returns available free bytes}
var TotalSpace: TLargeInteger;           {returns total bytes}
TotalFree: PLargeInteger                 {returns total free bytes}
): BOOL                                  {returns TRUE or FALSE}
```

Description

The GetDiskFreeSpaceEx function retrieves the total amount of space, the total amount of free space, and the total amount of available free space to the calling thread on a specified partition.

Note: Under Windows 95 prior to Service Release 2, this function returns incorrect values for volumes larger than 2 gigabytes in size.

Parameters

Directory: A null-terminated string containing the root directory of the drive to query. If this parameter is NIL, the function returns information for the partition containing the current directory.

FreeAvailable: A variable receiving the total amount of free space on the partition, in bytes, that is available to the calling thread.

TotalSpace: A variable receiving the total amount of space on the partition, in bytes.

TotalFree: A variable receiving the total amount of free space on the partition, in bytes.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

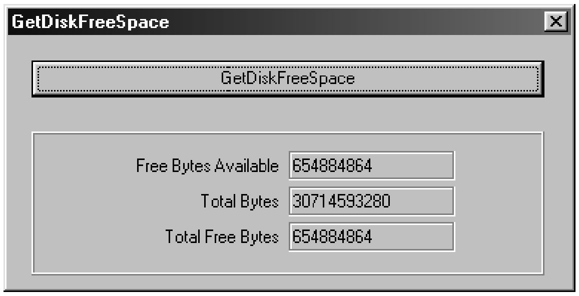
See Also

GetDriveType

Example

Listing 8-5: Retrieving the free disk space

```
procedure TForm1.Button1Click(Sender: TObject);
var
  FreeBytesAvailable,
  TotalBytes,
  TotalFreeBytes: TLargeInteger;
begin
  {retrieve the disk space information}
  if GetDiskFreeSpaceEx('C:\', FreeBytesAvailable, TotalBytes,
                        @TotalFreeBytes) then
  begin
    {display the disk space information}
    Panel2.Caption := IntToStr(FreeBytesAvailable);
    Panel3.Caption := IntToStr(TotalBytes);
    Panel4.Caption := IntToStr(TotalFreeBytes);
  end;
end;
```



```
Free Bytes Available  654884864
Total Bytes           30714593280
Total Free Bytes      654884864
```

Figure 8-3: Free disk space

GetDriveType   Windows.pas

Syntax

GetDriveType(
lpRootPathName: PChar   {a pointer to the root path string}
): UINT;   {returns a value based on the drive type}

Description

GetDriveType is used to determine the type of drive being accessed and will indicate fixed, removable, or remote (network) drives.

Parameters

lpRootPathName: A null-terminated string containing the root directory of the drive to be queried. If this parameter is NIL, the function uses the root of the current directory.

Return Value

If the function is successful, it returns one value from the following table; otherwise, it returns DRIVE_UNKNOWN.

See Also

GetDiskFreeSpaceEx

Example

Listing 8-6: Retrieving drive types

```
procedure TForm1.DriveComboBox1Change(Sender: TObject);
var
  DrivePath: array[0..3] of char;     // holds the root directory to query
begin
  {assemble the name of the root path of the drive to query}
  StrPCopy(DrivePath, DriveComboBox1.Drive);
  StrCat(DrivePath, ':\');

  {retrieve the drive type and display it}
  case GetDriveType(DrivePath) of
    DRIVE_UNKNOWN: Panel1.Caption := 'No Type Information';
    DRIVE_NO_ROOT_DIR: Label1.Caption := 'Root Directory does not exist';
    DRIVE_REMOVABLE: Panel1.Caption := 'Removable';
    DRIVE_FIXED: Panel1.Caption := 'Fixed';
    DRIVE_REMOTE: Panel1.Caption := 'Remote';
    DRIVE_CDROM: Panel1.Caption := 'CDROM';
    DRIVE_RAMDISK: Panel1.Caption := 'RamDisk';
  end;
end;
```

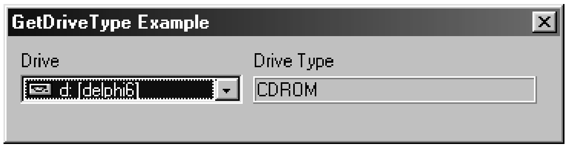


Figure 8-4: This drive is a CD-ROM

Table 8-2: GetDriveType return values

Value	Description
DRIVE_UNKNOWN	The drive type cannot be determined.
DRIVE_NO_ROOT_DIR	The root directory does not exist.
DRIVE_REMOVABLE	Indicates a removable disk drive.
DRIVE_FIXED	Indicates a non-removable disk drive (hard drive).
DRIVE_REMOTE	Indicates a remote (network) drive.
DRIVE_CDROM	Indicates a CD-ROM drive.
DRIVE_RAMDISK	Indicates a RAM disk.

GetEnvironmentStrings *Windows.pas*

Syntax

GetEnvironmentStrings: PChar {returns a pointer to the environment strings}

Description

The GetEnvironmentStrings function returns a pointer to the system environment variable strings. This includes environment variable settings such as the PATH, PROMPT and LASTDRIVE environment variables. The return value from this parameter can be used to specify the environment address in a call to the CreateProcess function. When the environment strings block is no longer needed, it should be freed by calling the FreeEnvironmentStrings function.

Return Value

If the function succeeds, it returns a pointer to a null-terminated string buffer. This buffer is composed of each environment variable string separated by a null terminating character. The buffer is ended with a double null terminating character. If the function fails, it returns NIL.

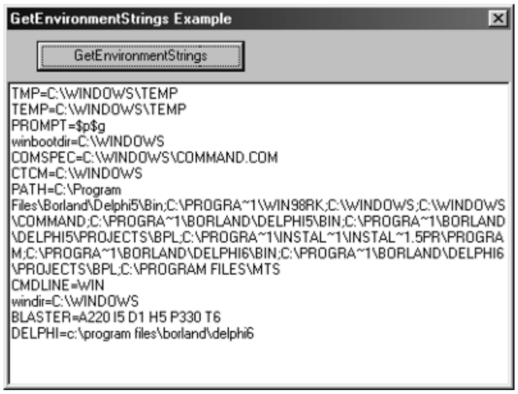
See Also

CreateProcess*, FreeEnvironmentStrings, GetEnvironmentVariable, SetEnvironmentVariable

Example

Listing 8-7: Retrieving the environment strings

```
procedure TForm1.Button1Click(Sender: Tobject);
var
  MyPointer: PChar;        // holds the returned environment strings
begin
  {clear the memo}
  Memo1.Lines.Clear;

  {retrieve the environment strings}
  MyPointer := GetEnvironmentStrings;

  {begin displaying environment strings}
  if MyPointer <> nil then
    while MyPointer <> nil do
    begin
      {display an environment string}
      Memo1.Lines.Add(StrPas(MyPointer));

      {environment strings are separated by null terminators, so if we
       increment the pointer past the null terminator of the last string
       displayed, it will be at the beginning of the next string}
      Inc(MyPointer,StrLen(MyPointer) + 1);

      {determine if we are at the end of the environment strings buffer}
      if (Byte(MyPointer[0]) = 0) then MyPointer := nil;
    end;

  {the environment strings are no longer needed, so free them}
  FreeEnvironmentStrings(MyPointer);
end;
```



```
TMP=C:\WINDOWS\TEMP
TEMP=C:\WINDOWS\TEMP
PROMPT=$p$g
winbootdir=C:\WINDOWS
COMSPEC=C:\WINDOWS\COMMAND.COM
CTCM=C:\WINDOWS
PATH=C:\Program
Files\Borland\Delphi5\Bin;C:\PROGRA~1\WIN98RK;C:\WINDOWS;C:\WINDOWS
\COMMAND;C:\PROGRA~1\BORLAND\DELPHI5\BIN;C:\PROGRA~1\BORLAND
\DELPHI5\PROJECTS\BPL;C:\PROGRA~1\INSTAL~1\INSTAL~1.5PR\PROGRA
M;C:\PROGRA~1\BORLAND\DELPHI6\BIN;C:\PROGRA~1\BORLAND\DELPHI6
\PROJECTS\BPL;C:\PROGRAM FILES\MTS
CMDLINE=WIN
windir=C:\WINDOWS
BLASTER=A220 I5 D1 H5 P330 T6
DELPHI=c:\program files\borland\delphi6
```

Figure 8-5: The returned environment strings

GetEnvironmentVariable   Windows.pas

Syntax

GetEnvironmentVariable(
lpName: PChar;   {a pointer to the variable name string}

```
lpBuffer: PChar;          {a pointer to a string to receive the value}
nSize: DWORD              {the size of the lpBuffer}
): DWORD;                 {returns the number of bytes written to the buffer}
```

Description

The GetEnvironmentVariable function retrieves a given environment variable value. The function can also return the required size of the buffer to hold the environment variable value.

Parameters

lpName: A null-terminated string containing the name of the environment variable to be retrieved.

lpBuffer: A pointer to a null-terminated string buffer that receives the environment variable's value. If this parameter is set to NIL, the function returns the size of the buffer required to hold the environment variable's value.

nSize: Specifies the maximum size of the buffer in characters.

Return Value

If the function succeeds, it returns the number of characters copied to the buffer pointed to by the lpBuffer parameter; otherwise, it returns zero.

See Also

GetEnvironmentStrings, SetEnvironmentVariable

Example

Please see Listing 8-23 under SetEnvironmentVariable.

GetLocaleInfo   Windows.pas

Syntax

```
GetLocaleInfo(
Locale: LCID;             {the locale identifier}
LCType: LCTYPE;           {information type flag}
lpLCData: PChar;          {a pointer to an output buffer}
cchData: Integer          {the length of the output buffer}
): Integer;               {returns the number of characters copied to the buffer}
```

Description

GetLocaleInfo retrieves specific information for a certain locale. A variety of information is available, according to the flag specified in the LCType parameter. The function returns the locale information in the form of a string, stored in the buffer pointed to by the lpLCData parameter.

Parameters

Locale: The locale identifier from which information is requested. This can be a specific locale identifier or one value from Table 8-3.

LCType: A flag indicating the type of information requested. The constant LOCALE_NOUSEROVERRIDE may be combined with any one of the items from Table 8-4, which means user overrides are not considered and the system default value for the locale is returned.

lpLCData: A pointer to a string buffer that receives the requested locale information.

cchData: Specifies the size, in bytes, of the buffer pointed to by the lpLCData parameter. If this parameter is zero, the function returns the size of the requested information, in bytes, and the lpLCData parameter is ignored.

Return Value

If the function succeeds, it returns the number of bytes copied to the lpLCData buffer, and lpLCData will point to a string containing the requested information. If the function fails, it returns zero. To get extended error information, call the GetLastError function.

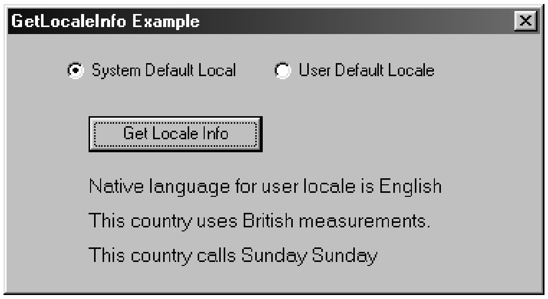
See Also

GetSystemDefaultLCID, GetUserDefaultLCID, SetLocaleInfo

Example

Listing 8-8: Retrieving locale information

```
procedure TForm1.Button1Click(Sender: TObject);
var
  OutputBuffer: PChar;  // holds local info
  SelectedLCID: LCID;   // holds the selected LCID
begin
  {allocate memory for the string}
  OutputBuffer := StrAlloc(255);

  {get the native language}
  if RadioButton1.Checked then
    SelectedLCID := GetSystemDefaultLCID
  else
    SelectedLCID := GetUserDefaultLCID;

  GetLocaleInfo(SelectedLCID, LOCALE_SNATIVELANGNAME,
              OutputBuffer, 255);
  Label1.Caption :='Native language for user locale is ' + OutputBuffer;

  {get the measurement system}
  GetLocaleInfo(SelectedLCID, LOCALE_IMEASURE, OutputBuffer, 255);
  if OutputBuffer = '0' then
    Label2.Caption := 'This country uses metric measurements.';
  if OutputBuffer = '1' then     // pounds, ounces, quarts, miles, etc.
    Label2.Caption := 'This country uses British measurements.';

  {get the name of Sunday}
  GetLocaleInfo(SelectedLCID, LOCALE_SDAYNAME7, OutputBuffer, 255);
  Label3.Caption := 'This country calls Sunday ' + OutputBuffer;

  {dispose of the string memory}
  StrDispose(OutputBuffer);
end;
```



Native language for user locale is English
This country uses British measurements.
This country calls Sunday Sunday

Figure 8-6: Default locale information

Table 8-3: GetLocaleInfo Locale values

Value	Description
LOCALE_SYSTEM_DEFAULT	The system's default locale.
LOCALE_USER_DEFAULT	The user's default locale.

Table 8-4: GetLocaleInfo LCType values

Value	Description
LOCALE_FONTSIGNATURE	A bit pattern identifying the relationship between characters needed to support the language of the locale and the contents of fonts.
LOCALE_ICALENDARTYPE	The current calendar type. Maximum output buffer size is two characters. The calendar type values are given in Table 8-5.
LOCALE_ICENTURY	Specifier for four-digit century. The maximum size of this string is two characters. The specifier can be one of the following values: 0 = abbreviated two-digit century; 1 = four-digit century.
LOCALE_ICOUNTRY	Country code, given as an IBM country code (international phone code) with a maximum length of six characters.
LOCALE_ICURRDIGITS	Number of fractional digits for local monetary displays, with the maximum size of the returned output buffer being three characters.
LOCALE_ICURRENCY	Positive currency mode. Maximum output buffer is two characters. The mode values are given in Table 8-6.
LOCALE_IDATE	Short date format. The maximum size of this string is two. The specifier can be any value from Table 8-7.
LOCALE_IDAYLZERO	Specifier for leading zeroes in day fields. The maximum size of this string is two. The specifier can be one of the following values: 0 = no leading zeroes for days; 1 = leading zeroes for days.
LOCALE_IDEFAULTANSICODEPAGE	ANSI code page for this locale. If the locale does not have an ANSI code page, this value is 0. This has a maximum size of six characters.
LOCALE_IDEFAULTCODEPAGE	OEM code page for this locale. This has a maximum size of six characters.

Value	Description
LOCALE_IDEFAULTCOUNTRY	Country code of the primary country of the locale. This has a maximum size of six characters.
LOCALE_IDEFAULTLANGUAGE	Language identifier of the primary language of the locale. This has a maximum size of five characters.
LOCALE_IDIGITS	This is the number of fractional digits. The maximum number of digits for this return value is three. That does not mean that the maximum number of fractional digits is three; it simply means the maximum number could possibly be 999, taking up three spaces in the returned output buffer.
LOCALE_IDIGITSUBSTITUTION	**Windows 2000/XP and later**: Determines the shape of digits. This can be any value from Table 8-8.
LOCALE_IFIRSTDAYOFWEEK	Specifier for the first day of the week for the locale. This has a maximum size of two characters and can be any value from Table 8-9.
LOCALE_IFIRSTWEEKOFYEAR	Specifier for the first week of the year for the locale. This has a maximum size of two characters and can be any value from Table 8-10.
LOCALE_IINTLCURRDIGITS	Number of fractional digits for international monetary displays, maximum size being three characters.
LOCALE_ILANGUAGE	Language of the locale. Maximum length is five.
LOCALE_ILDATE	Long date format. The maximum size of this string is two. The specifier can be any of the values from Table 8-7.
LOCALE_ILZERO	Specifies if leading zeroes exist for decimal fields. 0 = no leading zeroes. 1 = use leading zeroes. The maximum number of spaces returned for this field is two.
LOCALE_IMEASURE	System of measurement for the locale. 0 for metric (S.I.) and 1 for U.S. measurements system. The maximum size for this value is two characters.
LOCALE_IMONLZERO	Specifier for leading zeroes in month fields. The maximum size of this string is two. The specifier can be one of the following values: 0 = no leading zeroes for months; 1 = leading zeroes for months.
LOCALE_INEGCURR	Negative currency mode. Maximum size for this string is three. The mode can be any of the values in Table 8-11.
LOCALE_INEGNUMBER	Negative number formatting. Maximum size for this string is two. The mode can be any of the values in Table 8-12.
LOCALE_INEGSEPBYSPACE	Separation of monetary symbol in a negative monetary value. This value is 1 if the monetary symbol is separated by a space from the negative amount, 0 if it is not. The maximum size of this string is two.
LOCALE_INEGSIGNPOSN	Formatting index for negative values. The maximum size of this string is two. The index can be one of the values from Table 8-13.
LOCALE_INEGSYMPRECEDES	Position of monetary symbol in a negative monetary value. This value is 1 if the monetary symbol precedes the negative amount, 0 if it follows it. The maximum size of this string is two.

Value	Description
LOCALE_IOPTIONALCALENDAR	Specifies an available optional calendar. This can be one value from Table 8-5.
LOCALE_IPAPERSIZE	**Windows 2000/XP and later**: The default paper size used for the locale. This can be one value from Table 8-14.
LOCALE_IPOSSEPBYSPACE	Separation of monetary symbol in a positive monetary value. This value is 1 if the monetary symbol is separated by a space from a positive amount, 0 if it is not. The maximum size of this string is 2.
LOCALE_IPOSSIGNPOSN	Formatting index for positive values. The maximum size of this string is 2. The index can be one of the values from Table 8-13.
LOCALE_IPOSSYMPRECEDES	Position of monetary symbol in a positive monetary value. This value is 1 if the monetary symbol precedes the positive amount, 0 if it follows it. The maximum size of this string is two.
LOCALE_ITIME	Time format specifier. The maximum size of this string is 2. The specifier can be one of the following values: 0 = AM/PM 12-hour format; 1 = 24-hour format
LOCALE_ITIMEMARKPOSN	AM/PM position indicator. This can be one of the following values: 0 = used as a suffix; 1 = used as a prefix.
LOCALE_ITIMEMARKERUSE	Specifies if the AM/PM marker is used with 12-hour or 24-hour clocks. This can be one value from Table 8-15.
LOCALE_ITLZERO	Specifier for leading zeroes in time fields. The maximum size of this string is two. The specifier can be one of the following values: 0 = no leading zeroes for hours; 1 = leading zeroes for hours.
LOCALE_NOUSEROVERRIDE	Used in conjunction with other values from this table, this flag causes the function to return the system default for the indicated LCType instead of any user-defined values.
LOCALE_RETURN_NUMBER	**Windows 98/Me/NT 4.0 and later**: Used in conjunction with any other LOCALE_I* flag, this flag causes the function to return the value as a number instead of a string. The buffer pointed to by lpLCData must be at least the size of a DWORD.
LOCALE_S1159	String for the AM designator.
LOCALE_S2359	String for the PM designator.
LOCALE_SABBREVCTRYNAME	ISO 3166 abbreviated country name.
LOCALE_SABBREVDAYNAME1	Native abbreviated name for Monday.
LOCALE_SABBREVDAYNAME2	Native abbreviated name for Tuesday.
LOCALE_SABBREVDAYNAME3	Native abbreviated name for Wednesday.
LOCALE_SABBREVDAYNAME4	Native abbreviated name for Thursday.
LOCALE_SABBREVDAYNAME5	Native abbreviated name for Friday.
LOCALE_SABBREVDAYNAME6	Native abbreviated name for Saturday.
LOCALE_SABBREVDAYNAME7	Native abbreviated name for Sunday.
LOCALE_SABBREVLANGNAME	Name of language, in abbreviated ISO 639 format, using the two-character abbreviation, with a possible third character to indicate a sublanguage.
LOCALE_SABBREVMONTHNAME1	Native abbreviated name for January.

Value	Description
LOCALE_SABBREVMONTHNAME2	Native abbreviated name for February.
LOCALE_SABBREVMONTHNAME3	Native abbreviated name for March.
LOCALE_SABBREVMONTHNAME4	Native abbreviated name for April.
LOCALE_SABBREVMONTHNAME5	Native abbreviated name for May.
LOCALE_SABBREVMONTHNAME6	Native abbreviated name for June.
LOCALE_SABBREVMONTHNAME7	Native abbreviated name for July.
LOCALE_SABBREVMONTHNAME8	Native abbreviated name for August.
LOCALE_SABBREVMONTHNAME9	Native abbreviated name for September.
LOCALE_SABBREVMONTHNAME10	Native abbreviated name for October.
LOCALE_SABBREVMONTHNAME11	Native abbreviated name for November.
LOCALE_SABBREVMONTHNAME12	Native abbreviated name for December.
LOCALE_SABBREVMONTHNAME13	Native abbreviated name for the thirteenth month, if it exists.
LOCALE_SCOUNTRY	Unabbreviated localized name of the country.
LOCALE_SCURRENCY	Monetary symbol ($ for the U.S.).
LOCALE_SDATE	Characters for the date separator.
LOCALE_SDAYNAME1	Native long name for Monday.
LOCALE_SDAYNAME2	Native long name for Tuesday.
LOCALE_SDAYNAME3	Native long name for Wednesday.
LOCALE_SDAYNAME4	Native long name for Thursday.
LOCALE_SDAYNAME5	Native long name for Friday.
LOCALE_SDAYNAME6	Native long name for Saturday.
LOCALE_SDAYNAME7	Native long name for Sunday.
LOCALE_SDECIMAL	Character that is used as a decimal separator (such as a period for U.S. floating-point values).
LOCALE_SENGCOUNTRY	Unabbreviated English name of the country. This representation can always be shown in a 7-bit ASCII (127-character) character set.
LOCALE_SENGCURRNAME	**Windows 98/Me/2000/XP and later**: The English name for the locale currency.
LOCALE_SENGLANGUAGE	Full English name of the locale in ISO standard 639 format. This representation can always be shown in a 7-bit ASCII (127-character) character set.
LOCALE_SGROUPING	Number of decimal digits in each group to the left of the decimal character. This is a string with values separated by semicolons. The number in each group is given separately. If a number is common for all groups, specify the number followed by a zero group. In the U.S., this would be given by a string value 3;0, meaning that all the groups have three decimal digits.
LOCALE_SINTLSYMBOL	ISO 4217 international monetary symbol for the locale, given as three characters, followed by the character that separates the string from the amount display.

Value	Description
LOCALE_SISO3166CTRYNAME	**Windows 98/Me/NT 4.0 and later**: The ISO 3166 name for the country or region.
LOCALE_SISO639LANGNAME	**Windows 98/Me/NT 4.0 and later**: The abbreviated ISO 639 name for the locale language.
LOCALE_SLANGUAGE	Full unabbreviated localized language name for this locale.
LOCALE_SLIST	The character that is used as a list separator for the locale.
LOCALE_SLONGDATE	Long date formatting string for the current locale.
LOCALE_SMONDECIMALSEP	Characters used as the monetary decimal separator.
LOCALE_SMONGROUPING	Sizes for each group of decimal digits to the left of the decimal point. The number in each group is given separately. If a number is common for all groups, specify the number followed by a zero group. In the U.S., this would be given by a string value 3;0, meaning that all the groups have three decimal digits.
LOCALE_SMONTHNAME1	Native long name for January.
LOCALE_SMONTHNAME2	Native long name for February.
LOCALE_SMONTHNAME3	Native long name for March.
LOCALE_SMONTHNAME4	Native long name for April.
LOCALE_SMONTHNAME5	Native long name for May.
LOCALE_SMONTHNAME6	Native long name for June.
LOCALE_SMONTHNAME7	Native long name for July.
LOCALE_SMONTHNAME8	Native long name for August.
LOCALE_SMONTHNAME9	Native long name for September.
LOCALE_SMONTHNAME10	Native long name for October.
LOCALE_SMONTHNAME11	Native long name for November.
LOCALE_SMONTHNAME12	Native long name for December.
LOCALE_SMONTHNAME13	Native long name for the thirteenth month, if it exists.
LOCALE_SMONTHOUSANDSEP	Characters used as monetary separators for groups of digits to the left of the decimal point.
LOCALE_SNATIVECTRYNAME	Native name of the country or region.
LOCALE_SNATIVECURRNAME	**Windows 98/Me/2000/XP and later**: The native name of the locale currency.
LOCALE_SNATIVE DIGITS	The native digits for 0 through 9. This allows any characters in the locale character set to be used to represent numerical output regardless of their ASCII value mappings.
LOCALE_SNATIVELANGNAME	Name of the language in the native language.
LOCALE_SNEGATIVESIGN	String value for the negative sign.
LOCALE_SPOSITIVESIGN	String value for the positive sign.
LOCALE_SSHORTDATE	Short date formatting string for the current locale.
LOCALE_SSORTNAME	**Windows 98/Me/2000/XP and later**: Full native name of the sort for the specified locale.

Value	Description
LOCALE_STHOUSAND	Character or characters used to separate digit groups on the left side of the decimal character (decimal point). This would be the comma for U.S. locales.
LOCALE_STIME	Characters for the time separator.
LOCALE_STIMEFORMAT	Time formatting strings for the current locale.
LOCALE_SYEARMONTH	**Windows 98/Me/2000/XP and later**: Specifies the format string for dates containing only the year and month.
LOCALE_USE_CP_ACP	Uses the system ANSI code page for any string translation. This flag can be combined with any other flag in this table.

Table 8-5: GetLocaleInfo LCType LOCALE_ICALENDARTYPE and LOCALE_IOPTIONALCALENDAR type values

Value	Description
0	No calendar
1	Localized Gregorian
2	English string Gregorian
3	Japanese (Year of the Emperor)
4	Taiwan
5	Korean (Tangun Era)
6	Hijri (Arabic lunar)
7	Thai
8	Hebrew (lunar)
9	Middle East French Gregorian
10	Arabic Gregorian
11	Gregorian Transliterated English
12	Gregorian Transliterated French

Table 8-6: GetLocaleInfo LCType LOCALE_ICURRENCY mode values

Value	Description
0	Prefix with no separation
1	Suffix with no separation
2	Prefix with one character separation
3	Suffix with one character separation

Table 8-7: GetLocaleInfo LCType LOCALE_IDATE and LOCALE_ILDATE values

Value	Description
0	Month-Day-Year
1	Day-Month-Year
2	Year-Month-Day

Table 8-8: GetLocaleInfo LCType LOCALE_IDIGITSUBSTITUTION values

Value	Description
0	The shape depends on the previous text in the same output.
1	None, full Unicode compatibility.
2	Native shapes, as determined by LOCALE_SNATIVEDIGITS.

Table 8-9: GetLocaleInfo LCType LOCALE_IFIRSTDAYOFWEEK values

Value	Description
0	LOCALE_SDAYNAME1
1	LOCALE_SDAYNAME2
2	LOCALE_SDAYNAME3
3	LOCALE_SDAYNAME4
4	LOCALE_SDAYNAME5
5	LOCALE_SDAYNAME6
6	LOCALE_SDAYNAME7

Table 8-10: GetLocaleInfo LCType LOCALE_IFIRSTWEEKOFYEAR values

Value	Description
0	The week containing January 1 is the first week.
1	The first full week containing January 1 is the first week.
2	The first week containing at least four days is the first week.

Table 8-11: GetLocaleInfo LCType LOCALE_INEGCURR mode values

Value	Description
0	($1.1)
1	–$1.1
2	$–1.1
3	$1.1–
4	(1.1$)
5	–1.1$
6	1.1–$
7	1.1$–
8	–1.1 $ (space before $)
9	–$ 1.1 (space after $)
10	1.1 $– (space before $)
11	$ 1.1– (space after $)
12	$ –1.1 (space after $)
13	1.1– $ (space before $)
14	($ 1.1) (space after $)
15	(1.1 $) (space before $)

Table 8-12: GetLocaleInfo LCType LOCALE_INEGNUMBER mode values

Value	Description
0	(12345)
1	–12345
2	– 12345
3	12345–
4	12345–

Table 8-13: GetLocaleInfo LCType LOCALE_INEGSIGNPOSN and LOCALE_IPOSSIGNPOSN values

Value	Description
0	Parentheses surround the amount and the monetary symbol.
1	The sign string precedes the amount and the monetary symbol.
2	The sign string succeeds the amount and the monetary symbol.
3	The sign string immediately precedes the monetary symbol.
4	The sign string immediately succeeds the monetary symbol.

Table 8-14: GetLocaleInfo LCType LOCALE_IPAPERSIZE values

Value	Description
1	Letter size
5	Legal size
8	A3
9	A4

Table 8-15: GetLocaleInfo LCType LOCALE_ITIMEMARKERUSE values

Value	Description
0	Use with 12-hour clock
1	Use with 24-hour clock
2	Use with both
3	Never used

GetLocalTime *Windows.pas*

Syntax

```
GetLocalTime(
var lpSystemTime: TSystemTime   {a pointer to a TSystemTime structure}
);                              {this procedure does not return a value}
```

Description

The GetLocalTime function retrieves the current local date and time.

Parameters

lpSystemTime: A pointer to a TSystemTime structure that receives the current local date and time. The TSystemTime data structure is defined as:

```
TSystemTime = record
      wYear: Word;                {the current year}
      wMonth: Word;               {the month number}
      wDayOfWeek: Word;           {the day of the week number}
      wDay: Word;                 {the current day of the month}
      wHour: Word;                {the current hour}
      wMinute: Word;              {the current minute}
      wSecond: Word;              {the current second}
      wMilliseconds: Word;        {the current millisecond}
end;
```

Please see the FileTimeToSystemTime function for a description of this data structure.

See Also

GetSystemTime, SetLocalTime

Example

Please see Listing 8-25 under SetLocalTime.

GetLogicalDrives Windows.pas

Syntax

GetLogicalDrives: DWORD {returns a bitmask representing available drives}

Description

The GetLogicalDrives function retrieves a bitmask value, where each bit represents an available drive (bit 0 = drive A, bit 1 = drive B, etc.).

Return Value

If the function succeeds, it returns a bitmask value representing available drives; otherwise, it returns zero.

See Also

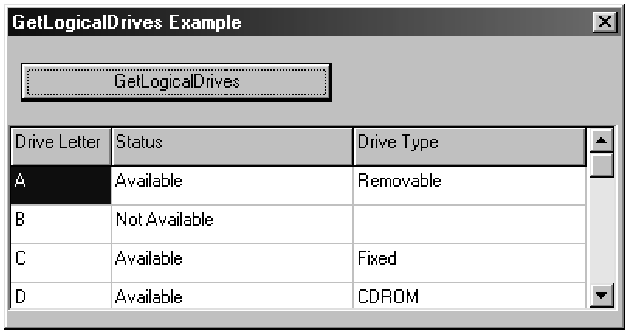
GetLogicalDriveStrings

Example

Listing 8-9: Retrieving a list of available drives

```
procedure TForm1.Button1Click(Sender: TObject);
var
  AvailableDrives: DWord;            // holds the bitmask of available drives
  Counter: Integer;                  // general loop counter
  DrivePath: array[0..3] of Char;    // holds the drive name
begin
```

```
  {display column headings}
  StringGrid1.Cells[0, 0] := 'Drive Letter';
  StringGrid1.Cells[1, 0] := 'Status';
  StringGrid1.Cells[2, 0] := 'Drive Type';

  {retrieve the available disk drives}
  AvailableDrives := GetLogicalDrives;

  {loop through all 26 possible drive letters}
  for Counter := 0 to 25 do
  begin
    {display the drive letter}
    StringGrid1.Cells[0, Counter + 1] := Char(Ord('A') + Counter);

    {if this drive is available...}
    if LongBool(AvailableDrives and ($0001 shl Counter)) = True then
    begin
      {indicate that the drive is available}
      StringGrid1.Cells[1, Counter + 1] := 'Available';

      {prepare drive path for GetDriveType function}
      StrpCopy(DrivePath,Char(Ord('A') + Counter));
      StrCat(DrivePath, ':\');

      {retrieve and display the drive type}
      case GetDriveType(DrivePath) of
        DRIVE_UNKNOWN: StringGrid1.Cells[2, Counter + 1] := 'No Type Information';
        DRIVE_NO_ROOT_DIR: StringGrid1.Cells[2, Counter + 1] := 'Root does not exist';
        DRIVE_REMOVABLE: StringGrid1.Cells[2, Counter + 1] := 'Removable';
        DRIVE_FIXED: StringGrid1.Cells[2, Counter + 1] := 'Fixed';
        DRIVE_REMOTE: StringGrid1.Cells[2, Counter + 1] := 'Remote';
        DRIVE_CDROM: StringGrid1.Cells[2, Counter + 1] := 'CDROM';
        DRIVE_RAMDISK: StringGrid1.Cells[2, Counter + 1] := 'RamDisk';
      end;
    end
    else
      {indicate that this drive is not available}
      StringGrid1.Cells[1, Counter + 1] := 'Not Available';
  end;
end;
```



Drive Letter	Status	Drive Type
A	Available	Removable
B	Not Available	
C	Available	Fixed
D	Available	CDROM

Figure 8-7: Available drives

GetLogicalDriveStrings Windows.pas

Syntax

GetLogicalDriveStrings(	
nBufferLength: DWORD;	{the size of the buffer}
lpBuffer: PAnsiChar	{a pointer to a buffer receiving drive name strings}
): DWORD;	{returns the number of characters copied to the buffer}

Description

This function retrieves the names of all logically defined drives, including mapped network drives, and stores them in the buffer pointed to by the lpBuffer parameter.

Parameters

nBufferLength: Specifies the size of the buffer pointed to by the lpBuffer parameter, excluding the null terminator.

lpBuffer: A pointer to a buffer that receives the names of the logical drives. Logical drive names are in the form of driveletter:\ (i.e., C:\). Each drive name in the string is separated by a null terminating character, and the string is ended with two null terminating characters. If this parameter is set to NIL, the function returns the size of the buffer required to hold the drive names.

Return Value

If the function succeeds, it returns the number of characters copied to the buffer, not counting the null terminator at the end. If the function fails, it returns zero. To get extended error information, call the GetLastError function.

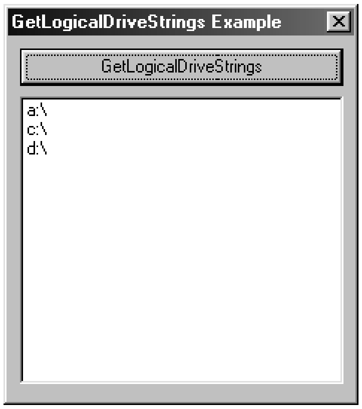
See Also

GetDiskFreeSpaceEx, GetDriveType, GetLogicalDrives

Example

Listing 8-10: Retrieving the names of the logical drives

```
procedure TForm1.Button1Click(Sender: TObject);
var
  TheDriveStrings: array[0..4*26+2] of Char; // holds the drive strings
  StringPtr: PChar;                          // a pointer to the drive strings
begin
  {retrieve the logical drive strings}
  GetLogicalDriveStrings(SizeOf(TheDriveStrings), TheDriveStrings);

  {initialize the pointer to the beginning of the drive strings buffer}
  StringPtr := TheDriveStrings;

  {begin looping through the drive strings}
  while StringPtr <> nil do
  begin
    {add this string to the list box}
    ListBox1.Items.Add(StringPtr);
    {logical drive strings are separated by null terminators, so if we
     increment the pointer past the null terminator of the last string
     displayed, it will be at the beginning of the next string}
    Inc(StringPtr, StrLen(StringPtr) + 1);

    {determine if we are at the end of the logical drive strings buffer}
    if (Byte(StringPtr[0]) = 0) then
      StringPtr := nil;
  end;
end;
```



```
a:\
c:\
d:\
```

Figure 8-8: The drive names

GetStartupInfo Windows.pas

Syntax

GetStartupInfo(	
var lpStartupInfo: TStartupInfo	{record to receive startup info}
);	{this procedure does not return a value}

Description

The GetStartUpInfo function retrieves information about the main window of the calling process when the process was created.

Parameters

lpStartupInfo: A pointer to a TStartupInfo structure that receives information about the main window of the calling process. The TStartupInfo structure is defined as:

TStartupInfo = record	
cb: DWORD;	{the size of the TStartupInfo record}
lpReserved: Pointer;	{reserved}
lpDesktop: Pointer;	{a pointer to the desktop}
lpTitle: Pointer;	{the title for console applications}
dwX: DWORD;	{the default column (left) position}
dwY: DWORD;	{the default row (top) position}
dwXSize: DWORD;	{the default width}
dwYSize: DWORD;	{the default height}

```
    dwXCountChars: DWORD;  {the screen width for a console app}
    dwYCountChars: DWORD;  {the screen height for a console app}
    dwFillAttribute: DWORD;  {color settings for a console app}
    dwFlags: DWORD;          {flags to determine significant fields}
    wShowWindow: Word;       {the default show window setting}
    cbReserved2: Word;       {reserved}
    lpReserved2: PByte;      {reserved}
    hStdInput: THandle;      {the standard handle for input}
    hStdOutput: THandle;     {the standard handle for output}
    hStdError: THandle;      {the standard handle for error output}
end;
```

The TStartupInfo structure is described under the CreateProcess function in *The Tomes of Delphi: Win32 Core API — Windows 2000 Edition*.

See Also

CreateProcess*

Example

Listing 8-11: Retrieving the startup information

```
const
  {define the names of the ShowWindow constants}
  ShowConst: array[0..10] of string = ('SW_HIDE','SW_SHOWNORMAL',
                                       'SW_SHOWMINIMIZED', 'SW_SHOWMAXIMIZED',
                                       'SW_SHOWNOACTIVATE','SW_SHOW',
                                       'SW_MINIMIZE', 'SW_SHOWMINNOACTIVE',
                                       'SW_SHOWNA','SW_RESTORE',
                                       'SW_SHOWDEFAULT');

procedure TForm1.Button1Click(Sender: TObject);
var
  MyStartupInfo: TStartupInfo;    // holds window startup properties
begin
  {retrieve the startup properties}
  GetStartupinfo(MyStartupInfo);

  {display the startup properties}
  Panel_State.Caption  := ShowConst[MyStartupInfo.wShowWindow];
  Panel_Left.Caption   := IntToStr(MyStartupInfo.dwX);
  Panel_Top.Caption    := IntToStr(MyStartupInfo.dwY);
  Panel_Width.Caption  := IntToStr(MyStartupInfo.dwXSize);
  Panel_Height.Caption := IntToStr(MyStartupInfo.dwYSize);
end;
```

GetSystemDefaultLangID Windows.pas

Syntax

GetSystemDefaultLangID: LANGID {returns the default system language identifier}

Description

This function returns the default language identifier for the system. Use the VerLanguageName function to get the name from the identifier.

Return Value

If the function succeeds, it returns the default numeric language identifier for the system; otherwise, it returns zero.

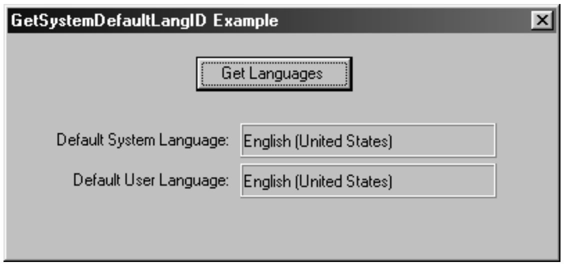
See Also

GetLocaleInfo, GetSystemDefaultLCID, GetUserDefaultLangID, VerLanguageName

Example

Listing 8-12: Retrieving the system default language identifier

```
procedure TForm1.Button1Click(Sender: TObject);
var
  SDLName:array[0..255] of char;  // holds the name of the system language
  UDLName:array[0..255] of char;  // holds the name of the user language
begin
  {retrieve the names of the system and user default languages}
  VerLanguageName(GetSystemDefaultLangID, SDLName, 255);
  VerLanguageName(GetUserDefaultLangID, UDLName, 255);

  {display the names of the languages}
  Label_SD.caption := SDLName;
  Label_Ud.caption := UDLName;
end;
```



Default System Language: English (United States)
Default User Language: English (United States)

Figure 8-9: Language names for the system and user default languages

GetSystemDefaultLCID   Windows.pas

Syntax

GetSystemDefaultLCID: LCID   {returns the system default locale identifier}

Description

This function is used to retrieve the default locale identifier for the system.

Return Value

If the function succeeds, it returns the system default locale identifier. If the function fails, it returns zero.

See Also

GetLocaleInfo, GetUserDefaultLCID

Example

Please see Listing 8-8 under GetLocaleInfo.

GetSystemDirectory Windows.pas

Syntax

GetSystemDirectory(	
lpBuffer: PChar;	{a pointer to a buffer receiving the directory string}
uSize: UINT	{the maximum size of the lpBuffer buffer}
): UINT;	{returns the number of bytes written to the buffer}

Description

This function retrieves a string containing the path of the Windows system directory. Applications should not create files in this directory.

Parameters

lpBuffer: A pointer to a null-terminated string buffer receiving the Windows system directory path. If this parameter is set to NIL, the function returns the required size of the buffer to hold the Windows system directory path string.

uSize: Specifies the maximum size of the buffer pointed to by the lpBuffer parameter, in characters.

Return Value

If the function is successful, it returns the number of bytes written to the buffer pointed to by the lpBuffer parameter; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

GetCurrentDirectory, GetWindowsDirectory, SetCurrentDirectory

Example

Listing 8-13: Retrieving the Windows system directory

```
procedure TForm1.Button1Click(Sender: TObject);
var
  SysDir: array[0..MAX_PATH] of Char;    // holds the system directory
begin

  {retrieve the system directory and display it}
  GetSystemDirectory(SysDir, MAX_PATH);
  Label1.Caption := StrPas(SysDir)
end;
```

GetSystemInfo Windows.pas

Syntax

```
GetSystemInfo(
var lpSystemInfo: TSystemInfo           {a pointer to a system information record}
);                                      {this procedure does not return a value}
```

Description

This function retrieves information about the type of system hardware in use.

Parameters

lpSystemInfo: A pointer to a TSystemInfo structure that receives information about the system and hardware upon which the process is running. The TSystemInfo data structure is defined as:

```
TSystemInfo = record
      case Integer of
      0: (
      dwOemId: DWORD)                         {obsolete}
      1:(
      wProcessorArchitecture: Word            {the type of systemprocessor}
      wReserved: Word                         {reserved}
      dwPageSize: DWORD                       {page size for virtual memory}
      lpMinimumApplicationAddress: Pointer    {lowest memory access address}
      lpMaximumApplicationAddress: Pointer    {highest memory access address}
      dwActiveProcessorMask: DWORD            {a bit array of active processors}
      dwNumberOfProcessors: DWORD             {the number of processors}
      dwProcessorType: DWORD                  {the type of processor present}
      dwAllocationGranularity: DWORD          {granularity of virtual memory}
      wProcessorLevel: Word                   {system required processor level}
      wProcessorRevision: Word)               {system required processor revision}
end;
```

dwOemId: This member is obsolete and is not used.

Note: Under Windows 95/98/Me, this member will always be set to PROCESSOR_ARCHITECTURE_INTEL.

wProcessorArchitecture: Indicates the type of system-specific processor architecture in the system and can be one value from Table 8-16.

wReserved: This member is reserved for future use and is currently ignored.

dwPageSize: Indicates the page size of virtual memory. This value is used by the LocalAlloc function to allocate additional blocks of memory.

lpMinimumApplicationAddress: A pointer to the lowest memory address that is accessible by applications and DLLs.

lpMaximumApplicationAddress: A pointer to the highest memory address that is accessible by applications and DLLs.

dwActiveProcessorMask: An array of bits indicating which processors are present and active. Bit 0 indicates processor 0, bit 1 indicates processor 1, etc.

dwNumberOfProcessors: Indicates the total number of processors in the system.

dwProcessorType: Under Windows 95 and Windows NT prior to version 4, this member indicates the type of processor in the system and can be one value from Table 8-17. However, on later versions of Windows, this member is obsolete and is ignored.

dwAllocationGranularity: Specifies the minimum amount of memory allocated each time virtual memory is used. In the past, this has been hard-coded to 64K, but it can vary for different processor architectures.

wProcessorLevel: **Windows NT/2000/XP and later**: This member indicates the level of the processor present and can be one value from Table 8-18.

wProcessorRevision: **Windows NT/2000/XP and later**: This member indicates the revision of the processor present and can be one value from Table 8-19.

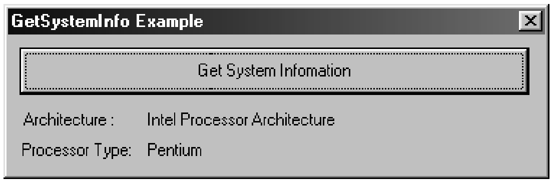
See Also

GetVersionEx

Example

Listing 8-14: Retrieving the system information

```
const
  {Whoops!  These constants are used by the GetSystemInfo function,
   but they are not defined in the Delphi source code}
  PROCESSOR_INTEL_386     = 386;
  PROCESSOR_INTEL_486     = 486;
  PROCESSOR_INTEL_PENTIUM = 586;
  PROCESSOR_MIPS_R4000    = 4000;
  PROCESSOR_ALPHA_21064   = 21064;

  PROCESSOR_ARCHITECTURE_INTEL    = 0;
  PROCESSOR_ARCHITECTURE_MIPS     = 1;
  PROCESSOR_ARCHITECTURE_ALPHA    = 2;
  PROCESSOR_ARCHITECTURE_PPC      = 3;
  PROCESSOR_ARCHITECTURE_UNKNOWN  = $FFFF;
  PROCESSOR_ARCHITECTURE_IA64     = 6;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var
  MySysInfo: TSystemInfo;   // holds the system information
begin
  {retrieve information about the system}
  GetSystemInfo(MySysInfo);
  {display the system's processor architecture}
  case MySysInfo.wProcessorArchitecture of
    PROCESSOR_ARCHITECTURE_INTEL: begin
      {display the processor architecture}
      Label4.Caption := 'Intel Processor Architecture';

      {display the processor type}
      case MySysInfo.dwProcessorType of
        PROCESSOR_INTEL_386: Label5.Caption := '80386';
        PROCESSOR_INTEL_486: Label5.Caption := '80486';
        PROCESSOR_INTEL_PENTIUM: Label5.Caption := 'Pentium';
      end;
    end;
    PROCESSOR_ARCHITECTURE_MIPS:
      Label4.Caption := 'MIPS Processor Architecture';
    PROCESSOR_ARCHITECTURE_ALPHA:
      Label4.Caption := 'DEC ALPHA Processor Architecture';
    PROCESSOR_ARCHITECTURE_PPC:
      Label4.Caption := 'PPC Processor Architecture';
    PROCESSOR_ARCHITECTURE_UNKNOWN:
      Label4.Caption := 'Unknown Processor Architecture';
  end;
end;
```



Architecture : Intel Processor Architecture
Processor Type: Pentium

Figure 8-10: The processor architecture

Table 8-16: GetSystemInfo lpSystemInfo.wProcessorArchitecture values

Name	Description
PROCESSOR_ARCHITECTURE_INTEL	Intel X86 processor architecture.
PROCESSOR_ARCHITECTURE_MIPS	Indicates Windows NT 3.51.
PROCESSOR_ARCHITECTURE_ALPHA	Indicates Windows NT 4.0 or earlier.
PROCESSOR_ARCHITECTURE_PPC	Indicates Windows NT 4.0 or earlier.
PROCESSOR_ARCHITECTURE_IA64	Indicates 64-bit Windows.
PROCESSOR_ARCHITECTURE_UNKNOWN	Unknown processor architecture

Table 8-17: GetSystemInfo lpSystemInfo.dwProcessorType values

Name	Description
PROCESSOR_INTEL_386	Intel 386 processor.
PROCESSOR_INTEL_486	Intel 486 processor.
PROCESSOR_INTEL_PENTIUM	Intel Pentium processor.

Table 8-18: GetSystemInfo lpSystemInfo.wProcessorLevel values

wProcessorArchitecture Value	Value	Description
PROCESSOR_ARCHITECTURE_INTEL	3	Intel 80386
	4	Intel 80486
	5	Pentium
	6	Pentium Pro
PROCESSOR_ARCHITECTURE_MIPS	4	MIPS R4000
PROCESSOR_ARCHITECTURE_ALPHA	21064	Alpha 21064
	21066	Alpha 21066
	21164	Alpha 21164
PROCESSOR_ARCHITECTURE_PPC	1	PPC 601
	3	PPC 603
	4	PPC 604
	6	PPC 603+
	9	PPC 604+
	20	PPC 620
PROCESSOR_ARCHITECTURE_IA64	1	64-bit Windows

Table 8-19: GetSystemInfo lpSystemInfo.wProcessorRevision values

Processor Type	Revision Breakdown
Intel 80386 or 80486	A value of the form xxyz.If xx = $FF, then y - $A is the model number and z is the stepping identifier. For example, an Intel 80486-D0 will return a value of $FFD0. If xx < $FF, then xx + 'A' is the stepping letter and yz is the minor stepping.

Intel Pentium, Cyrix, or NextGen 586	A value of the form xxyy.
	xx = model number
	yy = stepping
	For example, a value of $0201 indicates Model 2, Stepping 1.

MIPS	A value of the form 00xx. xx = eight-bit revision number of the processor (the low-order eight bits of the PRId register).

ALPHA	A value of the form xxyy.
	xx = model number
	yy = pass number
	For example, a value of $0201 indicates Model A02, Pass 01.

Processor Type	Revision Breakdown
PPC	A value of the form xxyy. xx.yy = processor version register For example, a value of $0201 indicates Version 2.01.

GetSystemTime Windows.pas

Syntax

```
GetSystemTime(
var SystemTime: TSystemTime         {a pointer to a system time structure}
)                                   {this procedure does not return a value}
```

Description

This function retrieves the current system date and time in coordinated universal time format (UTC).

Parameters

SystemTime: A pointer to a TSystemTime structure that receives the current system date and time. The TSystemTime data structure is defined as:

```
TSystemTime = record
      wYear: Word;                  {the current year}
      wMonth: Word;                 {the month number}
      wDayOfWeek: Word;             {the day of the week number}
      wDay: Word;                   {the current day of the month}
      wHour: Word;                  {the current hour}
      wMinute: Word;                {the current minute}
      wSecond: Word;                {the current second}
      wMilliseconds: Word;          {the current millisecond}
end;
```

Please see the FileTimeToSystemTime function for a description of this data structure.

See Also

GetLocalTime, SetSystemTime

Example

Please see Listing 8-26 under SetSystemTime.

GetSystemTimeAsFileTime Windows.pas

Syntax

```
GetSystemTimeAsFileTime(
var lpSystemTimeAsFileTime: TFileTime     {a pointer to a TFileTime structure}
);                                        {this procedure does not return a value}
```

Description

This procedure is used to retrieve the current system date and time in the form of a file time variable. This value is expressed in coordinated universal time format (UTC).

Parameters

lpSystemTimeAsFileTime: A pointer to a TFileTime data structure that receives the current system time in coordinated universal time format (UTC). Please see Chapter 4 for more information about coordinated universal time format (UTC) and the TFileTime data structure.

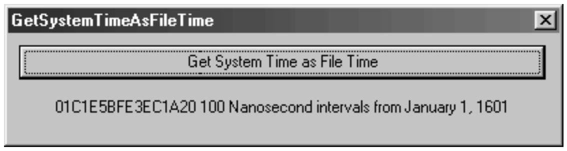
See Also

GetSystemTime, SystemTimeToFileTime

Example

Listing 8-15 Retrieving the system time in 100-nanosecond intervals

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyFileTime: TFileTime;    // holds the system file time as a file time value
begin
  {retrieve the system time as a file time}
  GetSystemTimeAsFileTime(MyFileTime);

  {display the system time as a file time value}
  Panel1.Caption := IntToHex(MyFileTime.dwHighDateTime ,8)+
                    IntToHex(MyFileTime.dwLowDateTime , 8) +
                    ' 100 Nanosecond intervals from January 1, 1601' ;
end;
```



01C1E5BFE3EC1A20 100 Nanosecond intervals from January 1, 1601

Figure 8-11: The current system time

GetTimeZoneInformation Windows.pas

Syntax

GetTimeZoneInformation(
var lpTimeZoneInformation: TTimeZoneInformation {a pointer to TTimeZoneInformation}
): DWORD; {returns a time zone code}

Description

This function is used to retrieve the time zone information for the local system. The time zone information controls the translation between coordinated universal time format (UTC) and local time.

Parameters

lpTimeZoneInformation: A pointer to a TTimeZoneInformation data structure that receives the time zone information for the system. The TTimeZoneInformation data structure is defined as:

```
TTimeZoneInformation = record
        Bias: Longint                          {Difference between times}
        StandardName: array[0..31] of WCHAR {Name of Time Zone in Standard}
        StandardDate: TSystemTime              {Date of change to Standard time}
        StandardBias: Longint                  {Standard time added to Bias}
        DaylightName: array[0..31] of WCHAR {Name of Time Zone in Daylight}
        DaylightDate: TSystemTime              {Date of change to Daylight time}
        DaylightBias: Longint                  {Daylight time added to Bias}
end
```

Bias: Specifies the difference in local time and coordinated universal time format (UTC), in minutes. To find the translation between a UTC time format and local time, use the following formula:

```
Coordinated Universal Time = Local Time + Bias.
```

StandardName: Contains a null-terminated string describing the name of the time zone during the Standard Daylight time state.

StandardDate: A TSystemTime structure that specifies the date that the system will change from Daylight Time to Standard Time.

StandardBias: Additional difference in UTC and local time during Standard Time. This value is added to the Bias member when determining the difference in time during the Standard Time state, but in most time zones this value is set to zero.

DaylightName: Contains a null-terminated string describing the name of the time zone during the Daylight Time state.

DaylightDate: A TSystemTime structure that specifies the date that the system will change from Standard Time to Daylight Time.

DaylightBias: Additional difference in UTC and local time during Daylight Time. This value is added to the Bias member when determining the difference in time during the Daylight Time state; in most time zones set this value to –60.

Return Value

If the function succeeds, it returns one time zone code from the following table; otherwise, it returns $FFFFFFFF. To get extended error information, call the GetLastError function.

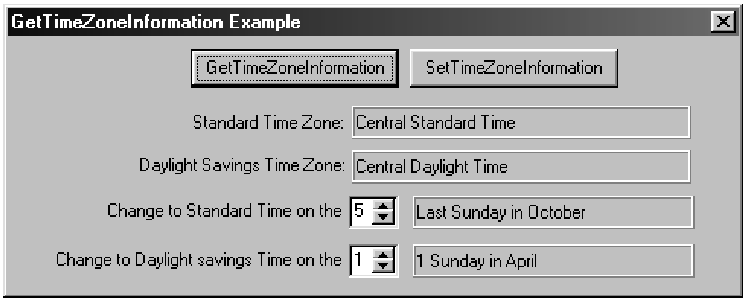
See Also

GetLocalTime, GetSystemTime, SetTimeZoneInformation

Example

Listing 8-16: Retrieving time zone information

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyTimeZoneInformation: TTimeZoneInformation;  // holds time zone information
begin
  {retrieve the time zone information}
  GetTimeZoneInformation(MyTimeZoneInformation);

  {display the time zone information}
  Panel1.Caption := MyTimeZoneInformation.StandardName;
  Panel2.Caption := MyTimeZoneInformation.DaylightName;

  {display standard time starting day}
  spnStandard.Value := MyTimeZoneInformation.StandardDate.wDay;
  if MyTimeZoneInformation.StandardDate.wDay = 5 then
    Label_Startstandard.Caption := 'Last Sunday in ' +
      LongMonthNames[MyTimeZoneInformation.StandardDate.wMonth]
  else
    Label_Startstandard.Caption :=
      IntToStr(MyTimeZoneInformation.StandardDate.wDay) +
      ' Sunday in ' + LongMonthNames[MyTimeZoneInformation.StandardDate.wMonth];

  {display daylight savings time starting day}
  spnDaylight.Value := MyTimeZoneInformation.DaylightDate.wDay;
  if MyTimeZoneInformation.DaylightDate.wDay = 5 then
    Label_Daylight.Caption := 'Last Sunday in ' +
      LongMonthNames[MyTimeZoneInformation.DaylightDate.wMonth]
  else
    Label_Daylight.Caption := IntToStr(MyTimeZoneInformation.DaylightDate.wDay)+
      ' Sunday in ' + LongMonthNames[MyTimeZoneInformation.DaylightDate.wMonth];
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  MyTimeZoneInformation: TTimeZoneInformation;  // holds time zone information
begin
  {retrieve the time zone information}
  GetTimeZoneInformation(MyTimeZoneInformation);

  {specify our changes}
  MyTimeZoneInformation.StandardDate.wDay := spnStandard.Value;
  MyTimeZoneInformation.DaylightDate.wDay := spnDaylight.Value;

  {set the new time zone information}
  SetTimeZoneInformation(MyTimeZoneInformation);
end;
```



GetTimeZoneInformation Example

GetTimeZoneInformation | SetTimeZoneInformation

Standard Time Zone: Central Standard Time

Daylight Savings Time Zone: Central Daylight Time

Change to Standard Time on the 5 Last Sunday in October

Change to Daylight savings Time on the 1 1 Sunday in April

Figure 8-12: The time zone information

Table 8-20: GetTimeZoneInformation return values

Value	Description
TIME_ZONE_ID_UNKNOWN	The system is in an unknown time zone. **Windows NT/2000/XP**: If daylight-saving time is not used in the current time zone, this value is returned.
TIME_ZONE_ID_STANDARD	The system is in Standard time state. **Windows 95/98/Me**: If daylight-saving time is not used in the current time zone, this value is returned.
TIME_ZONE_ID_DAYLIGHT	The system is in daylight-saving time state.

GetUserDefaultLangID Windows.pas

Syntax

GetUserDefaultLangID: LANGID {returns the default user language identifier}

Description

This function returns the default user identifier for the system. Use the VerLanguage-Name function to get the name from the identifier.

Return Value

If the function succeeds, it returns the default user numeric language identifier for the system; otherwise, it returns zero.

See Also

GetLocaleInfo, GetSystemDefaultLangID, GetUserDefaultLCID, VerLanguageName

Example

Please see Listing 8-12 under GetSystemDefaultLangID.

GetUserDefaultLCID Windows.pas

Syntax

GetUserDefaultLCID: LCID {returns the user default locale identifier}

Description

This function is used to retrieve the default user locale identifier for the system.

Return Value

If the function succeeds, it returns the default user locale identifier. If the function fails, it returns zero.

See Also

GetLocaleInfo, GetSystemDefaultLCID

Example

Please see Listing 8-8 under GetLocaleInfo.

GetUserName Windows.pas

Syntax

```
GetUserName(
lpBuffer: PChar;                {a pointer to a buffer to receive the username}
var nSize: DWORD                {the size of the buffer}
): BOOL;                        {returns TRUE or FALSE}
```

Description

This function is used to get the Windows networking username for the current user logged onto the system.

Parameters

lpBuffer: A pointer to a null-terminated string receiving the username. If this parameter is set to NIL, the variable identified by the nSize parameter will be set to the size of the buffer required to hold the username string.

nSize: A variable containing the size of the buffer pointed to by the lpBuffer parameter, in characters. When the function returns, this variable will contain the number of characters copied to the lpBuffer buffer.

Return Value

If the function succeeds, it returns TRUE and the variable identified by the nSize parameter will contain the number of characters copied to the lpBuffer buffer. If the function fails, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetComputerName

Example

Listing 8-17: Retrieving the username

```
procedure TForm1.Button1Click(Sender: Tobject);
var
  UserName: PChar;    // holds the user name
  Count: DWORD;       // holds the size of the user name
begin
  {retrieve the required size of the user name buffer}
  Count := 0;
  GetUserName(nil,Count);

  {allocate memory for the user name}
  Username := StrAlloc(Count);

  {retrieve the user name}
  if GetUserName(UserName,count) then
    Label1.Caption := StrPas(UserName)
  else ShowMessage('username Not Found');

  {dispose of allocated memory}
  StrDispose(UserName)
end;
```

GetVersionEx Windows.pas

Syntax

```
GetVersionEx(
var lpVersionInformation: TOSVersionInfo   {a pointer to a TOSVersionInfo structure}
): BOOL;                                   {returns TRUE or FALSE}
```

Description

This function retrieves information about the Windows version currently running on the system.

Parameters

lpVersionInformation: A pointer to a TOSVersionInfo data structure that receives information about the current version of Windows. The TOSVersionInfo data structure is defined as:

```
TOSVersionInfo = record
      dwOSVersionInfoSize: DWORD              {the size of TOSVersionInfo}
      dwMajorVersion: DWORD                   {the major version number}
      dwMinorVersion: DWORD                   {the minor version number}
      dwBuildNumber: DWORD                    {the build number}
      dwPlatformId: DWORD                     {operating system platform flags}
      szCSDVersion:array[0..127]of AnsiChar   {additional O/S information}
end;
```

dwOSVersionInfoSize: Specifies the size of the TOSVersionInfo structure, in bytes. This member must be set to SizeOf(TOSVersionInfo).

dwMajorVersion: Specifies the major version number of the operating system and can contain one value from Table 8-21.

dwMinorVersion: Specifies the minor version number of the operating system and can contain one value from Table 8-22.

dwBuildNumber: Specifies the build number of the operating system.

Note: Under Windows 95/98/Me, the high-order word of this value contains the major and minor version numbers.

dwPlatformId: A flag specifying the operating system platform. This member may contain one value from Table 8-23.

szCSDVersion: Contains a null-terminated string with additional information on the operating system. Under Windows 95/98/Me, this could contain one value from Table 8-24 (this table is not all inclusive).

Note: Under Windows NT/2000/XP and later, this string indicates the latest service pack installed.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetSystemInfo

Example

Listing 8-18: Retrieving information about the Windows version

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyVerInfo: TOSVersionInfo;    // holds version information
begin
  {set the size member of the TOSVersionInfo structure}
  MyVerInfo.dwOSVersionInfoSize := SizeOf(TOSVersionInfo);

  {retrieve the operating system version information}
  GetVersionEx(MyVerInfo);

  {display the operating system version information}
  Panel2.Caption := IntToStr(MyVerInfo.dwMajorVersion);
  Panel3.Caption := IntToStr(MyVerInfo.dwMinorVersion);
  Panel4.Caption := IntToStr(MyVerInfo.dwBuildNumber);
  case MyVerInfo.dwPlatformId of
    VER_PLATFORM_WIN32s: Panel5.Caption := 'Win 32s under Windows 3.1';
    VER_PLATFORM_WIN32_WINDOWS: Panel5.Caption := 'Windows 95/98/Me';
    VER_PLATFORM_WIN32_NT: Panel5.Caption := 'Windows NT/2000/XP/.NET server';
  end;
end;
```

Table 8-21: GetVersionEx lpVersionInformation.dwMajorVersion values

Value	Description
3	Windows NT 3.51
4	Windows 95/98/Me/NT 4.0
5	Windows 2000/XP/.NET server

Table 8-22: GetVersionEx lpVersionInformation.dwMinorVersion values

Value	Description
0	Windows 95/NT 4.0/2000
1	Windows XP/.NET server
10	Windows 98
51	Windows NT 3.51
90	Windows Me

Table 8-23: GetVersionEx lpVersionInformation.dwPlatformId values

Value	Description
VER_PLATFORM_WIN32s	Win32 on the 16-bit version of Windows
VER_PLATFORM_WIN32_WINDOWS	Windows 95/98/Me
VER_PLATFORM_WIN32_NT	Windows NT/2000/XP/.NET server

Table 8-24: GetVersionEx lpVersionInformation.szCSDVersion values

Value	Description
'C'	Windows 95 OSR 2
'A'	Windows 98 Second Edition

GetVolumeInformation Windows.pas

Syntax

```
GetVolumeInformation(
lpRootPathName: PChar;                  {the path to the root directory}
lpVolumeNameBuffer: PChar;              {the buffer receiving the volume name}
nVolumeNameSize: DWORD;                 {the maximum size of the buffer}
lpVolumeSerialNumber: PDWORD;           {a pointer to the volume serial number}
var lpMaximumComponentLength: DWORD;    {maximum file component name}
var lpFileSystemFlags: DWORD;           {file system flags}
lpFileSystemNameBuffer: PChar;          {the buffer receiving the file system name}
nFileSystemNameSize: DWORD              {the maximum size of the file system name}
): BOOL;                                {returns TRUE or FALSE}
```

Description

This function returns information about the file system and volume specified by the root directory path in the lpRootPathName parameter.

Parameters

lpRootPathName: A pointer to a null-terminated string containing the path of the root directory for the drive to query. If this parameter is set to NIL, the root directory of the current directory is used. For a UNC root directory path, add an additional backslash to the end (i.e., \\ServerName\ShareName\).

lpVolumeNameBuffer: A pointer to a null-terminated string buffer receiving the name of the volume.

nVolumeNameSize: Specifies the maximum size of the lpVolumeNameBuffer buffer.

lpVolumeSerialNumber: A variable that receives the serial number of the volume.

Note: Under Windows 95/98/Me, if the indicated drive is a network drive, the serial number is not returned.

lpMaximumComponentLength: A variable that receives the maximum size for file-names and directory names, in characters. Systems that support long file names, such as the FAT system, will return a value of 255.

lpFileSystemFlags: A variable that receives a value indicating the type of file system in use. This variable can be set to any combination of values from Table 8-25.

lpFileSystemNameBuffer: A pointer to a null-terminated string buffer receiving the name of the file system, such as FAT or NTFS.

nFileSystemNameSize: Specifies the maximum size of the lpFileSystemNameBuffer buffer.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

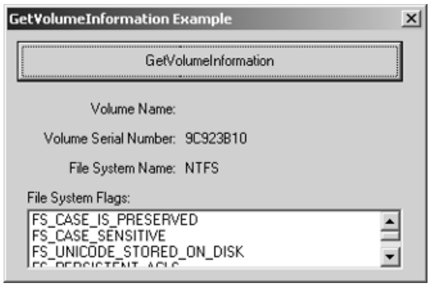
GetFileAttributes, SetVolumeLabel

Example

Listing 8-19: Retrieving volume information

```
const
  {these GetVolumeInformation constants are not defined in Delphi 6}
  FILE_SUPPORTS_OBJECT_IDS       = $00010000;
  FILE_SUPPORTS_ENCRYPTION       = $00020000;
  FILE_NAMED_STREAMS             = $00040000;
  FILE_READ_ONLY_VOLUME          = $00080000;
  FILE_VOLUME_QUOTAS             = $00000020;
  FILE_SUPPORTS_SPARSE_FILES     = $00000040;
  FILE_SUPPORTS_REPARSE_POINTS   = $00000080;
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  RootPath: array[0..20] of Char;     // holds the root directory name
  VolName: array[0..255] of Char;     // holds the volume name
  SerialNumber: DWORD;                // holds the serial number
  MaxCLength: DWORD;                  // holds the maximum file component length
  FileSysFlag: DWORD;                 // holds file system flags
  FileSysName: array[0..255] of Char; // holds the name of the file system
begin
  {indicate information is to be retrieved from the C drive}
  RootPath := 'C:\';

  {retrieve the volume information}
  GetVolumeInformation(RootPath, VolName, 255, @SerialNumber, MaxCLength,
     FileSysFlag, FileSysName, 255);

  {display the information}
  Label4.Caption := VolName;
  Label5.Caption := IntToHex(SerialNumber, 8);
  Label6.Caption := FileSysName;

  {determine the file system flags}
  if (FileSysFlag and FS_CASE_IS_PRESERVED) = FS_CASE_IS_PRESERVED then
    ListBox1.Items.Add('FS_CASE_IS_PRESERVED');
  if (FileSysFlag and FS_CASE_SENSITIVE) = FS_CASE_SENSITIVE then
    ListBox1.Items.Add('FS_CASE_SENSITIVE');
  if (FileSysFlag and FS_UNICODE_STORED_ON_DISK) = FS_UNICODE_STORED_ON_DISK then
    ListBox1.Items.Add('FS_UNICODE_STORED_ON_DISK');
  if (FileSysFlag and FS_PERSISTENT_ACLS) = FS_PERSISTENT_ACLS then
    ListBox1.Items.Add('FS_PERSISTENT_ACLS');
  if (FileSysFlag and FS_VOL_IS_COMPRESSED) = FS_VOL_IS_COMPRESSED then
    ListBox1.Items.Add('FS_VOL_IS_COMPRESSED');
  if (FileSysFlag and FS_FILE_COMPRESSION) = FS_FILE_COMPRESSION then
    ListBox1.Items.Add('FS_FILE_COMPRESSION');
  if (FileSysFlag and FILE_SUPPORTS_OBJECT_IDS) = FILE_SUPPORTS_OBJECT_IDS then
    ListBox1.Items.Add('FILE_SUPPORTS_OBJECT_IDS');
  if (FileSysFlag and FILE_SUPPORTS_ENCRYPTION) = FILE_SUPPORTS_ENCRYPTION then
    ListBox1.Items.Add('FILE_SUPPORTS_ENCRYPTION');
  if (FileSysFlag and FILE_NAMED_STREAMS) = FILE_NAMED_STREAMS then
    ListBox1.Items.Add('FILE_NAMED_STREAMS');
  if (FileSysFlag and FILE_READ_ONLY_VOLUME) = FILE_READ_ONLY_VOLUME then
    ListBox1.Items.Add('FILE_READ_ONLY_VOLUME');
  if (FileSysFlag and FILE_VOLUME_QUOTAS) = FILE_VOLUME_QUOTAS then
    ListBox1.Items.Add('FILE_VOLUME_QUOTAS');
  if (FileSysFlag and FILE_SUPPORTS_SPARSE_FILES) = FILE_SUPPORTS_SPARSE_FILES then
    ListBox1.Items.Add('FILE_SUPPORTS_SPARSE_FILES');
  if (FileSysFlag and FILE_SUPPORTS_REPARSE_POINTS) = FILE_SUPPORTS_REPARSE_POINTS then
    ListBox1.Items.Add('FILE_SUPPORTS_REPARSE_POINTS');
end;
```



Volume Name:

Volume Serial Number: 9C923B10

File System Name: NTFS

File System Flags:

FS_CASE_IS_PRESERVED
FS_CASE_SENSITIVE
FS_UNICODE_STORED_ON_DISK

Figure 8-13: The current volume information

Table 8-25: GetVolumeInformation lpFileSystemFlags values

Value	Description
FS_CASE_IS_PRESERVED	Indicates that the case of the file or directory name is retained when it is stored to disk.
FS_CASE_SENSITIVE	Indicates that the file system supports case-sensitive filenames.
FS_UNICODE_STORED_ON_DISK	Indicates that the file system supports Unicode characters in filenames as they appear on disk.
FS_PERSISTENT_ACLS	Indicates that the file system preserves and enforces ACLs.
FS_FILE_COMPRESSION	Indicates that the file system supports file-based compression. This flag cannot be used with FS_VOL_IS_COMPRESSED.
FS_VOL_IS_COMPRESSED	Indicates that the specified volume is compressed. This flag cannot be used with FS_FILE_COMPRESSION.
FILE_NAMED_STREAMS	Indicates that the file system supports named streams.
FILE_READ_ONLY_VOLUME	**Windows XP only**: The indicated volume is read only.
FILE_SUPPORTS_ENCRYPTION	Indicates that the file system supports the Encrypted File System (EFS).
FILE_SUPPORTS_OBJECT_IDS	Indicates that the file system supports object identifiers.
FILE_SUPPORTS_REPARSE_POINTS	Indicates that the file system supports reparse points.
FILE_SUPPORTS_SPARSE_FILES	Indicates that the file system supports sparse files.
FILE_VOLUME_QUOTAS	Indicates that the file system supports disk quotas.

GetWindowsDirectory Windows.pas

Syntax

```
GetWindowsDirectory(
lpBuffer: PChar;          {the buffer receiving Windows directory}
uSize: UINT             {the maximum size of the buffer}
): UINT;                {returns the number of bytes written to the buffer}
```

Description

This function retrieves the path for the Windows directory. Typically, this is the directory where applications should store initialization files and help files.

Parameters

lpBuffer: A pointer to a null-terminated string buffer receiving the Windows directory path. If this parameter is set to NIL, the function returns the required size of the buffer to hold the Windows directory path.

uSize: Specifies the maximum size of the buffer pointed to by the lpBuffer parameter and should indicate a minimum size of MAX_PATH characters.

Return Value

If the function succeeds, it returns the number of characters copied to the buffer pointed to by the lpBuffer parameter, not including the null terminator. If the function fails, it returns zero. To get extended error information, call the GetLastError function.

See Also

GetCurrentDirectory, GetSystemDirectory

Example

Listing 8-20: Retrieving the Windows directory

```
procedure TForm1.Button1Click(Sender: TObject);
var
  WinDir: array[0..MAX_PATH] of char;   // holds the Windows directory
begin
  {retrieve the Windows directory...}
  GetWindowsDirectory(WinDir, MAX_PATH);

  {...and display it}
  Label1.Caption := StrPas(WinDir)
end;
```

IsProcessorFeaturePresent   Windows.pas

Syntax

```
IsProcessorFeaturePresent(
ProcessorFeature: DWORD         {processor feature flag}
): BOOL;                        {returns TRUE or FALSE}
```

Description

This function determines if the processor feature indicated by the ProcessorFeature value is present on the system.

Note: This function is not supported under Windows 95/98/Me.

Parameters

ProcessorFeature: A flag indicating the feature to test. This parameter may be one value from the following table.

Return Value

If the feature is present, the function returns TRUE; otherwise, it returns FALSE

See Also

GetSystemMetrics*, SystemParametersInfo

Example

Listing 8-21: Checking for processor features

```
{******************************************************************************
  Note: The IsProcessorFeaturePresent function is not supported under Windows
  95/98/Me. This example should be run under Windows NT or later.
 ******************************************************************************}
const
  {these constants are not defined in Delphi 6}
  PF_FLOATING_POINT_PRECISION_ERRATA  = 0;
  PF_FLOATING_POINT_EMULATED          = 1;
  PF_COMPARE_EXCHANGE_DOUBLE          = 2;
  PF_MMX_INSTRUCTIONS_AVAILABLE       = 3;
  PF_XMMI_INSTRUCTIONS_AVAILABLE      = 6;
  PF_3DNOW_INSTRUCTIONS_AVAILABLE     = 7;
  PF_RDTSC_INSTRUCTION_AVAILABLE      = 8;
  PF_PAE_ENABLED                      = 9;


procedure TForm1.Button1Click(Sender: TObject);
begin
  {check for floating point precision error (early pentiums)}
  if IsProcessorFeaturePresent(PF_FLOATING_POINT_PRECISION_ERRATA) then
    Label5.Caption := 'TRUE';

  {check for software emulated floating point operations}
  if IsProcessorFeaturePresent(PF_FLOATING_POINT_EMULATED) then
    Label5.Caption := 'TRUE';

  {check for MMX instruction availability}
  if IsProcessorFeaturePresent(PF_MMX_INSTRUCTIONS_AVAILABLE) then
    Label5.Caption := 'TRUE';

  {check for 3D-Now instruction availability}
  if IsProcessorFeaturePresent(PF_3DNOW_INSTRUCTIONS_AVAILABLE) then
    Label5.Caption := 'TRUE';
end;
```

Table 8-26: IsProcessorFeaturePresent ProcessorFeature values

Value	Description
PF_FLOATING_POINT_PRECISION_ERRATA	Indicates the existence of the floating-point precision error.
PF_FLOATING_POINT_EMULATED	Indicates that floating-point operations are emulated in software.
PF_COMPARE_EXCHANGE_DOUBLE	Indicates that the compare and exchange double operation is available.
PF_MMX_INSTRUCTIONS_AVAILABLE	Indicates that MMX instructions are available.
PF_XMMI_INSTRUCTIONS_AVAILABLE	Indicates that XMMI instructions are available.
PF_3DNOW_INSTRUCTIONS_AVAILABLE	Indicates that 3D-Now instructions are available.
PF_RDTSC_INSTRUCTION_AVAILABLE	Indicates that the RDTSC instruction is available.
PF_PAE_ENABLED	Indicates that the processor is PAE enabled.

SetComputerName Windows.pas

Syntax

```
SetComputerName(
lpComputerName: PChar       {a pointer to the new computer name}
): BOOL;                    {returns TRUE or FALSE
```

Description

This function sets the computer name to the name specified by the lpComputerName parameter when the machine is rebooted.

Parameters

lpComputerName: A pointer to a null-terminated string containing the new name of the computer. This string cannot be longer than MAX_COMPUTERNAME_LENGTH characters.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetComputerName

Example

Listing 8-22: Setting and retrieving the computer name

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ComputerName: array[0..MAX_COMPUTERNAME_LENGTH + 1] of char; // holds the name
  Size: DWORD;                                                 // holds the size
begin
  {initialize the computer name size variable}
  Size := MAX_COMPUTERNAME_LENGTH + 1;

  {retrieve the computer name}
  if GetComputerName(ComputerName, Size) then
    Edit1.Text := StrPas(Computername)
  else
    Showmessage('Computer Name Not Found');
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  ComputerName: array[0..MAX_COMPUTERNAME_LENGTH + 1] of char; // holds the name
begin
  {copy the specified name to the ComputerName buffer}
  StrPCopy(ComputerName, Edit1.Text);

  {set the computer name}
  if SetComputerName(ComputerName) then
    ShowMessage('Computer name reset, setting will be used at next startup')
  else
    ShowMessage('Computer name not reset');
end;
```

SetEnvironmentVariable Windows.pas

Syntax

```
SetEnvironmentVariable(
lpName: PChar;          {the name of the environment variable to change}
lpValue: PChar          {the new environment variable value}
): BOOL;                {returns TRUE or FALSE}
```

Description

This function sets an environment variable for the current process. This function can also add or delete the environment variable for the current process.

Parameters

lpName: A pointer to a null-terminated string containing the name of the environment variable to change. If the environment variable does not exist, the system will create it if the lpValue parameter does not contain NIL. If the environment variable exists and the lpValue parameter contains NIL, the system deletes the specified environment variable.

lpValue: A pointer to a null-terminated string containing the new value for the environment variable.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

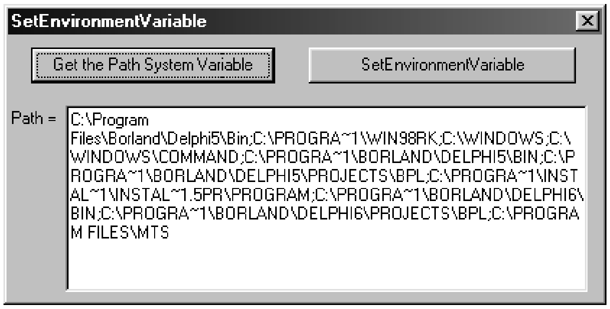
See Also

GetEnvironmentVariable

Example

Listing 8-23: Setting and retrieving environment variables

```
procedure TForm1.Button1Click(Sender: Tobject);
var
  ThePath: PChar;         // holds the Path environment variable
  Return: Integer;        // holds the function return value
begin
  {retrieve the size of the Path environment variable}
  Return := GetEnvironmentVariable('Path', nil, 0);

  {allocate a buffer}
  GetMem(ThePath, Return);

  {get the actual Path environment variable}
  Return := GetEnvironmentVariable('Path', ThePath, Return);

  {indicate any errors retrieving the path variable}
  if Return = 0 then
    ShowMessage('Path variable not found')
  {otherwise, display the path environment variable}
  else
    Edit1.Text := ThePath;

  {free the buffer}
  FreeMem(ThePath);
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  ThePath: array[0..MAX_PATH] of char;    // holds the Path environment variable
begin
  {copy the specified path setting to the buffer}
  StrPCopy(ThePath, Edit1.Text);

  {set the Path environment variable}
  if SetEnvironmentVariable('Path', ThePath) then
    ShowMessage('Variable has been reset')
  else
    ShowMessage('Variable has not been set')
end;
```



```
C:\Program
Files\Borland\Delphi5\Bin;C:\PROGRA~1\WIN98RK;C:\WINDOWS;C:\
WINDOWS\COMMAND;C:\PROGRA~1\BORLAND\DELPHI5\BIN;C:\P
ROGRA~1\BORLAND\DELPHI5\PROJECTS\BPL;C:\PROGRA~1\INST
AL~1\INSTAL~1.5PR\PROGRAM;C:\PROGRA~1\BORLAND\DELPHI6\
BIN;C:\PROGRA~1\BORLAND\DELPHI6\PROJECTS\BPL;C:\PROGRA
M FILES\MTS
```

Figure 8-14: The path environment variable

SetLocaleInfo Windows.pas

Syntax

```
SetLocaleInfo(
Locale: LCID;               {the locale identifier}
LCType: LCTYPE;             {locale data flag}
lpLCData: PChar             {a pointer to the new data}
): BOOL;                    {returns TRUE or FALSE}
```

Description

This function sets specific information for the locale identified by the Locale parameter. However, only certain locale information can be modified with this function.

Parameters

Locale: The locale identifier for which the information will be set.

LCType: A flag indicating the type of locale information to change. This parameter can be set to one value from Table 8-27. Note that this table of values is a subset of the locale information flags available under the GetLocaleInfo function.

lpLCData: A null-terminated string containing the new locale information. The LCType flag determines the size and format of this information.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetLocaleInfo

Example

Listing 8-24: Setting locale information

```
var
  Form1: TForm1;
  OriginalUnits: PChar;                   // holds original locale measurements
  OriginalUnitsBuff: array[0..1] of Char;
```

```
implementation

procedure TForm1.FormCreate(Sender: TObject);
begin
  {retrieve the original locale measurement units}
  OriginalUnits := @OriginalUnitsBuff;
  GetLocaleInfo(LOCALE_SYSTEM_DEFAULT, LOCALE_IMEASURE, OriginalUnits, 2);
end;

procedure TForm1.ButtonSetRadioClick(Sender: TObject);
var
  ChangeUnits: PChar;
  ChangeUnitsBuff: array[0..1] of Char;
begin
  {change the measurement units}
  ChangeUnits := @ChangeUnitsBuff;
  if RadioButtonBritish.Checked then
    ChangeUnits := '1'
  else
    ChangeUnits := '0';
  SetLocaleInfo(LOCALE_SYSTEM_DEFAULT, LOCALE_IMEASURE, ChangeUnits);

  {retrieve the set measurement units}
  GetLocaleInfo(LOCALE_SYSTEM_DEFAULT, LOCALE_IMEASURE, ChangeUnits, 2);
  if ChangeUnits = '0' then
    LabelMeasure.Caption := 'This country uses Metric measurements.';
  if ChangeUnits = '1' then     // pounds, ounces, quarts, miles, etc.
    LabelMeasure.Caption := 'This country uses British measurements.';
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  {restore original measurement units}
  SetLocaleInfo(LOCALE_SYSTEM_DEFAULT, LOCALE_IMEASURE, OriginalUnits);
end;
```

Table 8-27: SetLocaleInfo LCType values

Value	Description
LOCALE_ICALENDARTYPE	The current calendar type. Maximum input buffer size is two characters. The calendar type values are given in Table 8-28.
LOCALE_ICURRDIGITS	Number of fractional digits for local monetary displays, with the maximum size of the input buffer being three characters.
LOCALE_ICURRENCY	Positive currency mode. Maximum input buffer size is two characters. The mode values are given in Table 8-29.
LOCALE_IDIGITS	Indicates the number of fractional digits.
LOCALE_IFIRSTDAYOFWEEK	Specifier for the first day of the week for the locale. This has a maximum size of two characters and can be any value from Table 8-30.

Value	Description
LOCALE_IFIRSTWEEKOFYEAR	Specifier for the first week of the year for the locale. This has a maximum size of two characters, and can be any value from Table 8-31.
LOCALE_ILZERO	Specifies if leading zeroes exist for decimal fields. 0 = no leading zeroes; 1 = use leading zeroes.
LOCALE_IMEASURE	System of measurement for the locale. 0 for metric (S.I.) and 1 for U.S. measurements system.
LOCALE_INEGCURR	Negative currency mode. Maximum size for this string is three. The mode can be any of the values in Table 8-32.
LOCALE_INEGNUMBER	Negative number formatting. Maximum size for this string is two. The mode can be any of the values in Table 8-33.
LOCALE_IPAPERSIZE	**Windows 2000/XP and later**: The default paper size used for the locale. This can be one value from Table 8-34.
LOCALE_ITIME	Time format specifier. The maximum size of this string is two. The specifier can be one of the following values: 0 = AM/PM 12-hour format; 1 = 24-hour format.
LOCALE_S1159	String for the AM designator.
LOCALE_S2359	String for the PM designator.
LOCALE_SCURRENCY	Monetary symbol ($ for the U.S.).
LOCALE_SDATE	Characters for the date separator.
LOCALE_SDECIMAL	Character that is used as a decimal separator (such as a period for U.S. floating-point values).
LOCALE_SGROUPING	Number of decimal digits in each group to the left of the decimal character. This is a string with values separated by semicolons. The number in each group is given separately. If a number is common for all groups, specify the number followed by a zero group. In the U.S., this would be given by a string value 3;0, meaning that all the groups have three decimal digits.
LOCALE_SLIST	The character that is used as a list separator for the locale.
LOCALE_SLONGDATE	Long date formatting string for the current locale.
LOCALE_SMONDECIMALSEP	Characters used as the monetary decimal separator.
LOCALE_SMONGROUPING	Sizes for each group of decimal digits to the left of the decimal point. The number in each group is given separately. If a number is common for all groups, specify the number followed by a zero group. In the U.S., this would be given by a string value 3;0, meaning that all the groups have three decimal digits.
LOCALE_SMONTHOUSANDSEP	Characters used as monetary separators for groups of digits to the left of the decimal point.
LOCALE_SNEGATIVESIGN	String value for the negative sign.

Value	Description
LOCALE_SPOSITIVESIGN	String value for the positive sign.
LOCALE_SSHORTDATE	Short date formatting string for the current locale.
LOCALE_STHOUSAND	Character or characters used to separate digit groups on the left side of the decimal character (decimal point). This would be the comma for U.S. locales.
LOCALE_STIME	Characters for the time separator.
LOCALE_STIMEFORMAT	Time formatting strings for the current locale.
LOCALE_SYEARMONTH	**Windows 98/Me/2000/XP and later**: Specifies the format string for dates containing only the year and month.

Table 8-28: SetLocaleInfo LCType LOCALE_ICALENDARTYPE type values

Value	Description
0	No calendar
1	Localized Gregorian
2	English string Gregorian
3	Japanese (Year of the Emperor)
4	Taiwan
5	Korean (Tangun Era)
6	Hijri (Arabic lunar)
7	Thai
8	Hebrew (lunar)
9	Middle East French Gregorian
10	Arabic Gregorian
11	Gregorian Transliterated English
12	Gregorian Transliterated French

Table 8-29: SetLocaleInfo LCType LOCALE_ICURRENCY mode values

Value	Description
0	Prefix with no separation
1	Suffix with no separation
2	Prefix with one character separation
3	Suffix with one character separation

Table 8-30: SetLocaleInfo LCType LOCALE_IFIRSTDAYOFWEEK values

Value	Description
0	LOCALE_SDAYNAME1
1	LOCALE_SDAYNAME2
2	LOCALE_SDAYNAME3
3	LOCALE_SDAYNAME4

Value	Description
4	LOCALE_SDAYNAME5
5	LOCALE_SDAYNAME6
6	LOCALE_SDAYNAME7

Table 8-31: SetLocaleInfo LCType LOCALE_IFIRSTWEEKOFYEAR values

Value	Description
0	The week containing January 1 is the first week.
1	The first full week containing January 1 is the first week.
2	The first week containing at least four days is the first week.

Table 8-32: SetLocaleInfo LCType LOCALE_INEGCURR mode values

Value	Description
0	($1.1)
1	–$1.1
2	$–1.1
3	$1.1–
4	(1.1$)
5	–1.1$
6	1.1–$
7	1.1$–
8	–1.1 $ (space before $)
9	–$ 1.1 (space after $)
10	1.1 $– (space before $)
11	$ 1.1– (space after $)
12	$–1.1 (space after $)
13	1.1– $ (space before $)
14	($ 1.1) (space after $)
15	(1.1 $) (space before $)

Table 8-33: SetLocaleInfo LCType LOCALE_INEGNUMBER mode values

Value	Description
0	(12345)
1	–12345
2	– 12345
3	12345–
4	12345 –

Table 8-34: SetLocaleInfo LCType LOCALE_IPAPERSIZE values

Value	Description
1	Letter size
5	Legal size
8	A3
9	A4

SetLocalTime *Windows.pas*

Syntax

```
SetLocalTime(
const lpSystemTime: TSystemTime      {a pointer to a TSystemTime structure}
): BOOL;                             {returns TRUE or FALSE}
```

Description

This function sets the current local date and time.

Note: Under Windows NT/2000/XP, the system uses the current time zone information to perform any daylight-saving time conversion, if necessary. This conversion is based on the current time, not the new time being set. Also, calling this function enables the SE_SYSTEM-TIME_NAME security privilege, which is disabled by default.

Parameters

lpSystemTime: A pointer to a TSystemTime structure that contains the new current local date and time. The TSystemTime data structure is defined as:

```
TSystemTime = record
      wYear: Word;             {the current year}
      wMonth: Word;            {the month number}
      wDayOfWeek: Word;        {the day of the week number}
      wDay: Word;              {the current day of the month}
      wHour: Word;             {the current hour}
      wMinute: Word;           {the current minute}
      wSecond: Word;           {the current second}
      wMilliseconds: Word;     {the current millisecond}
end;
```

Please see the FileTimeToSystemTime function for a description of this data structure.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

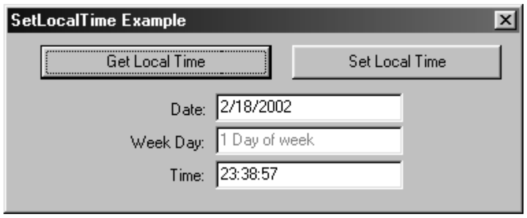
See Also

GetLocalTime, GetSystemTime, SetSystemTime

Example

Listing 8-25: Setting and retrieving the current local time

```
procedure TForm1.Button1Click(Sender: Tobject);
var
  CurrentTime : TSystemTime;                // holds time information
begin
  {retrieve the local time}
  GetLocalTime(CurrentTime);

  {display the local time elements}
  Edit1.Text := IntToStr(CurrentTime.wMonth) + '/' +
                IntToStr(CurrentTime.wDay) + '/' +
                IntToStr(CurrentTime.wYear);
  Edit2.Text := IntToStr(CurrentTime.wDayOfWeek) + ' Day of week';
  Edit3.Text := IntToStr(CurrentTime.whour) + ':' +
                IntToStr(CurrentTime.wMinute) + ':' +
                IntToStr(CurrentTime.wSecond);
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  CurrentTime : TSystemTime;                // holds time information
begin
  {retrieve the current time to initialize members of the CurrentTime
   structure that are not initialized from UI elements}
  GetLocalTime(CurrentTime);
  try
    {set the date from the user supplied date}
    Decodedate(StrToDateTime(Edit1.Text), CurrentTime.wYear, CurrentTime.wMonth,
      CurrentTime.wDay);

    {set the time from the user supplied time}
    DecodeTime(StrToTime(Edit3.text), CurrentTime.wHour, CurrentTime.wMinute,
      CurrentTime.wSecond, CurrentTime.wMilliseconds);

    {set the local time}
    SetLocalTime(CurrentTime)
  except
    {display a message if an error occurred}
    on E:Exception do ShowMessage('Error setting local time');
  end;
end;
```



Date: 2/18/2002
Week Day: 1 Day of week
Time: 23:38:57

Figure 8-15: The current local time

SetSystemTime Windows.pas

Syntax

```
SetSystemTime(
const lpSystemTime: TSystemTime       {a pointer to a TSystemTime structure}
): BOOL;                              {returns TRUE or FALSE}
```

Description

This function sets the current system date and time. The system time is in coordinated universal time format (UTC).

Note: Under Windows NT/2000/XP, calling this function enables the SE_SYSTEMTIME_NAME security privilege, which is disabled by default.

Parameters

lpSystemTime: A pointer to a TSystemTime structure that contains the new current system date and time. The TSystemTime data structure is defined as:

```
TSystemTime = record
      wYear: Word;              {the current year}
      wMonth: Word;             {the month number}
      wDayOfWeek: Word;         {the day of the week number}
      wDay: Word;               {the current day of the month}
      wHour: Word;              {the current hour}
      wMinute: Word;            {the current minute}
      wSecond: Word;            {the current second}
      wMilliseconds: Word;      {the current millisecond}
end;
```

Note that the value of the wDayOfWeek member is ignored. Please see the FileTimeToSystemTime function for a description of this data structure.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

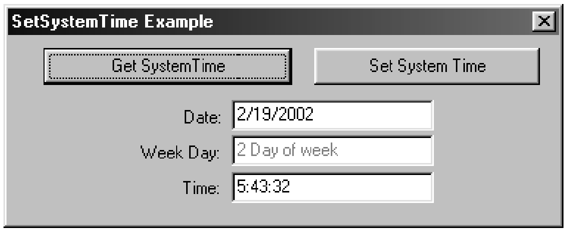
GetLocalTime, GetSystemTime, SetLocalTime

Example

Listing 8-26: Setting and retrieving the system time

```
procedure TForm1.Button1Click(Sender: Tobject);
var
  CurrentTime: TSystemTime;        // holds the system time
begin
```

```
  {retrieve the system time}
  GetSystemTime(CurrentTime);

  {display the system time}
  Edit1.Text := IntToStr(CurrentTime.wmonth) + '/' +
                IntToStr(CurrentTime.wDay) + '/' +
                IntToStr(CurrentTime.wYear);
  Edit2.Text := IntToStr(CurrentTime.wDayOfWeek) + ' Day of week';
  Edit3.Text := IntToStr(CurrentTime.whour) + ':' +
                IntToStr(CurrentTime.wMinute)+':'+IntToStr(CurrentTime.wSecond);
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  CurrentTime : TSystemTime;       // holds the system time
begin
  {retrieve the system time to initialize members of CurrentTime
   that are not supplied by the user}
  GetSystemTime(CurrentTime);
  try
    {set the date from the supplied date}
    DecodeDate(StrToDateTime(Edit1.Text), CurrentTime.wYear, CurrentTime.wMonth,
      CurrentTime.wDay);

    {set the time from the supplied time}
    DecodeTime(StrToTime(Edit3.Text), CurrentTime.wHour, CurrentTime.wMinute,
      CurrentTime.wSecond, CurrentTime.wMilliseconds);

    {set the system time}
    SetSystemTime(CurrentTime)
  except
    {indicate if there was an error}
    on E:Exception do ShowMessage('Error setting system time');
  end;
end;
```



Date: 2/19/2002
Week Day: 2 Day of week
Time: 5:43:32

Figure 8-16: The current system time

SetTimeZoneInformation ***Windows.pas***

Syntax

```
SetTimeZoneInformation(
const lpTimeZoneInformation:
  TTimeZoneInformation          {a pointer to TTimeZoneInformation}
): BOOL;                        {returns TRUE or FALSE}
```

Description

This function is used to set the time zone information for the local system. The time zone information controls the translation between coordinated universal time format (UTC) and local time.

Parameters

lpTimeZoneInformation: A pointer to a TTimeZoneInformation data structure that contains the new time zone information for the system. The TTimeZoneInformation data structure is defined as:

```
TTimeZoneInformation = record
      Bias: Longint                          {Difference between times}
      StandardName: array[0..31] of WCHAR    {Name of Time Zone in Standard}
      StandardDate: TSystemTime              {Date of change to Standard time}
      StandardBias: Longint                  {Standard time added to Bias}
      DaylightName: array[0..31] of WCHAR    {Name of Time Zone in Daylight}
      DaylightDate: TSystemTime              {Date of change to Daylight time}
      DaylightBias: Longint                  {Daylight time added to Bias}
end;
```

Please see GetTimeZoneInformation for more details.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetTimeZoneInformation, SetLocalTime, SetSystemTime

Example

Please see Listing 8-16 under GetTimeZoneInformation.

SetVolumeLabel Windows.pas

Syntax

```
SetVolumeLabel(
lpRootPathName: PChar;          {the root directory name of the volume to change}
lpVolumeName: PAnsiChar         {the new volume name}
): BOOL;                        {returns TRUE or FALSE}
```

Description

This function changes the volume name on the drive identified by the root path name contained in the lpRootPathName parameter.

Parameters

lpRootPathName: A pointer to a null-terminated string containing the path of the root directory on the drive whose volume name is to be changed. If this parameter is set to NIL, the volume containing the current directory is used.

lpVolumeName: A pointer to a null-terminated string containing the new name for the volume. If this parameter is set to NIL, the volume name is deleted.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetVolumeInformation

Example

Listing 8-27: Setting the volume name

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  {set the volume label for drive C}
  SetVolumeLabel('C:\', PChar(Edit1.Text));
end;
```

SystemParametersInfo Windows.pas

Syntax

```
SystemParametersInfo(
uiAction: UINT;          {the system-wide parameter to set or query}
uiParam: UINT;           {an integer dependent on uiAction}
pvParam: Pointer;        {a pointer to a structure dependent on uiAction}
fWinIni: UINT            {notification and save options}
): BOOL;                 {returns TRUE or FALSE}
```

Description

This function can query or set a system-wide parameter, such as mouse trails or desktop wallpaper. Most of these parameters are available from various applets under the Control Panel.

Parameters

uiAction: Specifies which system-wide parameter to set or query and can be one value from Table 8-36.

uiParam: An integer whose value is dependent on the value of the uiAction parameter. See Table 8-36 for a description of uiParam parameter values. Unless otherwise specified, this parameter should be set to zero.

pvParam: A pointer to a data structure. The type of data structure and its values are dependent on the value of the uiAction parameter. See Table 8-36 for a description of pvParam parameter structures. Unless otherwise specified, this parameter should be set to NIL.

fWinIni: Determines how the changes to the system-wide parameters are handled and can be one value from Table 8-35.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

GetSystemMetrics*

Example

Listing 8-28: Changing the size of non-client buttons

```
const
  {these constants are not defined in Delphi 6}
  SPI_GETDESKWALLPAPER          = $0073;
  SPI_GETDROPSHADOW             = $1024;
  SPI_SETDROPSHADOW             = $1025;
  SPI_GETFLATMENU               = $1022;
  SPI_SETFLATMENU               = $1023;
  SPI_GETFOCUSBORDERWIDTH       = $200E;
  SPI_SETFOCUSBORDERWIDTH       = $200F;
  SPI_GETFOCUSBORDERHEIGHT      = $2010;
  SPI_SETFOCUSBORDERHEIGHT      = $2011;
  SPI_GETFONTSMOOTHINGCONTRAST  = $200C;
  SPI_SETFONTSMOOTHINGCONTRAST  = $200D;
  SPI_GETFONTSMOOTHINGTYPE      = $200A;
  SPI_SETFONTSMOOTHINGTYPE      = $200B;
  SPI_GETMOUSECLICKLOCKTIME     = $2008;
  SPI_SETMOUSECLICKLOCKTIME     = $2009;
  SPI_GETMOUSECLICKLOCK         = $101E;
  SPI_SETMOUSECLICKLOCK         = $101F;
  SPI_GETMOUSESONAR             = $101C;
  SPI_SETMOUSESONAR             = $101D;
  SPI_GETMOUSEVANISH            = $1020;
  SPI_SETMOUSEVANISH            = $1021;

  FE_FONTSMOOTHINGSTANDARD      = $0001;
  FE_FONTSMOOTHINGCLEARTYPE     = $0002;

var
  MyNCM: TNonClientMetrics;                // holds non client metric info
  OriginalWidth, OriginalHeight: Integer;  // holds original button sizes

procedure TForm1.Button1Click(Sender: TObject);
begin
  {initialize the size of the data structure}
  MyNCM.cbSize := SizeOf(TNonClientMetrics);

  {retrieve the settings for the non client button sizes}
  SystemParametersInfo(SPI_GetNonClientMetrics, SizeOf(TNonClientMetrics),
                       @MyNCM, 0);

  {double the size of non client buttons}
  MyNCM.iCaptionWidth  := MyNCM.iCaptionWidth * 2;
  MyNCM.iCaptionHeight := MyNCM.iCaptionHeight * 2;

  {set the size of non client buttons to the new size}
  SystemParametersInfo(SPI_SetNonClientMetrics, SizeOf(TNonClientMetrics),
                       @MyNCM, SPIF_SENDWININICHANGE);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  {initialize the size of the data structure}
  MyNCM.cbSize := SizeOf(TNonClientMetrics);

  {retrieve the settings for the non client button sizes}
  SystemParametersInfo(SPI_GetNonClientMetrics, SizeOf(TNonClientMetrics),
                       @MyNCM, 0);

  {decrease the size of non client buttons}
  MyNCM.iCaptionWidth := MyNCM.iCaptionWidth div 2;
  MyNCM.iCaptionHeight := MyNCM.iCaptionHeight div 2;

  {set the size of non client buttons to the new size}
  SystemParametersInfo(SPI_SetNonClientMetrics, SizeOf(TNonClientMetrics),
                       @MyNCM, SPIF_SENDWININICHANGE);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  {initialize the size of the data structure}
  MyNCM.cbSize := SizeOf(TNonClientMetrics);

  {retrieve the settings for the non client button sizes}
  SystemParametersInfo(SPI_GetNonClientMetrics, SizeOf(TNonClientMetrics),
                       @MyNCM, 0);

  {store the original settings for restoration when the application ends}
  OriginalWidth  := MyNCM.iCaptionWidth;
  OriginalHeight := MyNCM.iCaptionHeight;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  {initialize the size of the data structure}
  MyNCM.cbSize := SizeOf(TNonClientMetrics);

  {set the size of the buttons to the original size}
  MyNCM.iCaptionWidth  := OriginalWidth;
  MyNCM.iCaptionHeight := OriginalHeight;

  {change the size of the non client buttons}
  SystemParametersInfo(SPI_SetNonClientMetrics, SizeOf(TNonClientMetrics),
                       @MyNCM, SPIF_SENDWININICHANGE);
end;
```

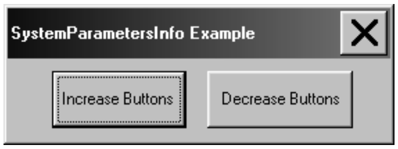


Figure 8-17: The modified non-client buttons

Table 8-35: SystemParametersInfo fWinIni values

Value	Description
SPIF_UPDATEINIFILE	Save the new settings to the user profile.
SPIF_SENDCHANGE	Update and broadcast the WM_SETTINGCHANGE message.

Table 8-36: SystemParametersInfo uiAction values

Value	Description
SPI_GETACCESSTIMEOUT	Retrieves information about the time-out period for accessibility features. uiParam: Specifies the size of the TAccessTimeout structure. pvParam: Points to a TAccessTimeout structure that receives the timeout information. **Note**: If accessibility features are not installed, this flag causes the function to fail.
SPI_GETACTIVEWINDOWTRACKING	**Windows 98/Me/2000/XP and later**: Indicates if active window tracking is enabled or disabled (active window tracking activates any window over which the mouse hovers). uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if enabled, FALSE if disabled.
SPI_GETACTIVEWNDTRKZORDER	**Windows 98/Me/2000/XP and later**: Indicates if windows activated via active window tracking are brought to the top. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if enabled, FALSE if disabled.
SPI_GETACTIVEWNDTRKTIMEOUT	**Windows 98/Me/2000/XP and later**: Indicates active window tracking window activation delay in milliseconds. uiParam: Not used. pvParam: Points to a DWORD value that receives the delay.
SPI_GETANIMATION	Retrieves information about animation effects, such as animated window minimizing/restoring. uiParam: Specifies the size of the TAnimationInfo structure. pvParam: Points to a TAnimationInfo structure that receives the animation effects information.
SPI_GETBEEP	Indicates whether the warning beeper is on or off. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if the beeper is on or FALSE if it is off.

Value	Description
SPI_GETBORDER	Retrieves the border multiplier that is used when determining the width of a window's sizing border. uiParam: Not used. pvParam: Points to an integer value that receives the border multiplier factor.
SPI_GETCARETWIDTH	**Windows 2000/XP and later**: Retrieves the width of the edit control caret, in pixels. uiParam: Not used. pvParam: Points to a DWORD value that receives the caret width.
SPI_GETCOMBOBOXANIMATION	**Windows 98/Me/2000/XP and later**: Indicates if the opening animation effect for combo boxes is enabled or disabled. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if enabled, FALSE if disabled.
SPI_GETCURSORSHADOW	**Windows 2000/XP and later**: Indicates if the cursor is displayed with a shadow. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if enabled, FALSE if disabled. **Note**: The video driver must be set to a color depth greater than 8-bit before this effect is displayed.
SPI_GETDESKWALLPAPER	**Windows 2000/XP and later**: Retrieves the path and filename of the desktop wallpaper bitmap. uiParam: Indicates the size of the buffer pointed to by the pvParam parameter, in characters. pvParam: Points to a null-terminated string buffer that receives the path and filename. The returned string will never be longer than MAX_PATH characters.
SPI_GETDEFAULTINPUTLANG	Retrieves the keyboard layout handle for the system default input language. uiParam: Not used. pvParam: Points to an integer value that receives the keyboard layout handle.
SPI_GETDRAGFULLWINDOWS	Indicates if full window dragging is enabled. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if full window dragging is enabled or FALSE if it is not. **Note**: This flag is supported under Windows 95 only if Windows Plus! is installed.

Value	Description
SPI_GETDROPSHADOW	**Windows XP only**: Indicates if the drop shadow effect is enabled or disabled. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if enabled, FALSE if disabled.
SPI_GETFILTERKEYS	Retrieves information about the FilterKeys accessibility feature. uiParam: Specifies the size of the TFilterKeys structure. pvParam: Points to a TFilterKeys structure that receives information about the FilterKeys feature. **Note**: If accessibility features are not installed, this flag causes the function to fail.
SPI_GETFLATMENU	**Windows XP only**: Indicates if native user menus have the flat menu appearance. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if the menus are flat, FALSE otherwise.
SPI_GETFOCUSBORDERHEIGHT	**Windows XP only**: Retrieves the height, in pixels, of the top or bottom edges of a focus rectangle drawn with the DrawFocusRect function. uiParam: Not used. pvParam: Points to a UINT value that receives the height of the focus rectangle edges.
SPI_GETFOCUSBORDERWIDTH	**Windows XP only**: Retrieves the width, in pixels, of the left or right edges of a focus rectangle drawn with the DrawFocusRect function. uiParam: Not used. pvParam: Points to a UINT value that receives the width of the focus rectangle edges.
SPI_GETFONTSMOOTHING	Indicates whether anti-aliasing is used to make font curves appear smoother (known as font smoothing). uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if font anti-aliasing is used or FALSE if it is not. **Note**: This flag is supported under Windows 95 only if the Windows Plus! Package is installed.
SPI_GETFONTSMOOTHINGCONTRAST	**Windows XP only**: Retrieves the contrast value used in ClearType font smoothing. uiParam: Not used. pvParam: Points to a UINT value that receives the contrast value. This value is in the range 1000-2200, with the default set at 1400.

Value	Description
SPI_GETFONTSMOOTHINGTYPE	**Windows XP only**: Retrieves the font smoothing type. uiParam: Not used. pvParam: Points to a UINT value that receives the font smoothing type information, either FE_FONTSMOOTHING-STANDARD for standard anti-aliasing or FE_FONT-SMOOTHINGCLEARTYPE for ClearType font smoothing.
SPI_GETFOREGROUNDFLASHCOUNT	**Windows 98/Me/2000/XP and later**: Retrieves the number of times an application's taskbar button flashes when rejecting a foreground switch request. uiParam: Not used. pvParam: Points to a DWORD value that receives the flash count.
SPI_GETFOREGROUNDLOCKTIMEOUT	**Windows 98/Me/2000/XP and later**: Retrieves the number of milliseconds following user input during which no applications are allowed to force themselves into the foreground. uiParam: Not used. pvParam: Points to a DWORD value that receives the lockout time.
SPI_GETGRADIENTCAPTIONS	**Windows 98/Me/2000/XP and later**: Indicates if the gradient effect in application title bars is enabled or disabled. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if enabled, FALSE if disabled.
SPI_GETGRIDGRANULARITY	Retrieves the granularity of the desktop sizing grid. uiParam: Not used. pvParam: Points to an integer value that receives the granularity value.
SPI_GETHIGHCONTRAST	**Windows 95/98/Me/2000/XP and later**: Retrieves information about the HighContrast accessibility feature, which sets the color scheme and appearance of the user interface to provide for maximum visibility for visually impaired users. uiParam: Specifies the size of the THighContrast structure. pvParam: Points to a THighContrast structure that receives information about the HighContrast feature. **Note**: If accessibility features are not installed, this flag causes the function to fail.
SPI_GETHOTTRACKING	**Windows 98/Me/2000/XP and later**: Indicates if hot tracking of user interface elements is enabled or disabled (i.e., flat buttons that display a border when the mouse enters, etc.). uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if enabled, FALSE if disabled.

Value	Description
SPI_GETICONMETRICS	Retrieves icon metric values, such as spacing and title wrap. uiParam: Specifies the size of the TIconMetrics structure. pvParam: Points to a TIconMetrics structure that receives information about icon metrics.
SPI_GETICONTITLELOGFONT	Retrieves the logical font for the current icon title font. uiParam: Specifies the size of the TLogFont structure. pvParam: Points to a TLogFont structure that receives the icon title logical font.
SPI_GETICONTITLEWRAP	Determines whether icon title wrapping is enabled. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if icon title wrapping is enabled or FALSE if it is not.
SPI_GETKEYBOARDCUES	**Windows 98/Me/2000/XP and later**: Indicates if menu access characters are always underlined. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if they are always underlined, FALSE otherwise.
SPI_GETKEYBOARDDELAY	Retrieves the keyboard repeat delay setting. uiParam: Not used. pvParam: Points to an integer value receiving the repeat delay setting.
SPI_GETKEYBOARDPREF	**Windows 95/98/Me/2000/XP and later**: Retrieves user keyboard preference, indicating whether the user prefers the keyboard over the mouse and wants applications to display keyboard interfaces that would otherwise be hidden. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if the user prefers the keyboard or FALSE if the user prefers the mouse.
SPI_GETKEYBOARDSPEED	Retrieves the keyboard repeat speed setting. uiParam: Not used. pvParam: Points to an integer value to receive the repeat speed setting.
SPI_GETLISTBOXSMOOTHSCROLLING	**Windows 98/Me/2000/XP and later**: Indicates if the smooth scrolling effect for list boxes is enabled or disabled. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if enabled, FALSE if disabled.

Value	Description
SPI_GETLOWPOWERACTIVE	Indicates if the screen saver low-power phase is enabled or disabled. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if enabled, FALSE if disabled. **Note**: Under Windows 95, this flag is supported for 16-bit applications only. Under Windows 98/Me, this is supported for 16- and 32-bit applications. Under Windows 2000/XP, only 32-bit applications are supported.
SPI_GETLOWPOWERTIMEOUT	Retrieves the time-out value, in seconds, for the screen save low-power phase. uiParam: Not used. pvParam: Points to an integer value that receives time-out value. **Note**: Under Windows 95, this flag is supported for 16-bit applications only. Under Windows 98/Me, this is supported for 16- and 32-bit applications. Under Windows 2000/XP, only 32-bit applications are supported.
SPI_GETMENUANIMATION	**Windows 98/Me/2000/XP and later**: Indicates if menu animation effects are enabled or disabled. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if enabled, FALSE if disabled.
SPI_GETMENUDROPALIGNMENT	Retrieves pop-up menu alignment, indicating whether pop-up menus are left or right aligned relative to the corresponding menu bar item. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if pop-up menus are left aligned or FALSE if they are right aligned.
SPI_GETMENUFADE	**Windows 2000/XP and later**: Indicates which menu animation effect is used, fading or sliding. If the return value is TRUE, the fade animation is used; otherwise, the slide animation is used. The SPI_GETMENUANIMATION flag determines if menu animation is enabled. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE for a fading effect, FALSE for a sliding effect.
SPI_GETMENUSHOWDELAY	**Windows 98/Me/NT/2000/XP and later**: Retrieves the amount of time, in milliseconds, before displaying a shortcut menu when the mouse is over a submenu item. uiParam: Not used. pvParam: Points to a DWORD value that receives the delay amount.
SPI_GETMENUUNDERLINES	See SPI_GETKEYBOARDCUES.

Value	Description
SPI_GETMINIMIZEDMETRICS	Retrieves the minimized window metrics, such as arrangement and width. uiParam: Specifies the size of the TMinimizedMetrics structure. pvParam: Pointer to a TMinimizedMetrics structure that receives the minimized window metric information.
SPI_GETMOUSE	Retrieves the two mouse speed threshold values and the mouse speed. uiParam: Not used. pvParam: Points to an array of the integer values to receive the mouse threshold values.
SPI_GETMOUSECLICKLOCK	**Windows Me/XP only**: Indicates if the mouse ClickLock feature is enabled or disabled. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if enabled, FALSE if disabled.
SPI_GETMOUSECLICKLOCKTIME	**Windows Me/XP only**: Retrieves the delay, in milliseconds, between the time the mouse button is pressed and the mouse ClickLock feature activates. The SPI_GETMOUSECLICKLOCK flag indicates if the ClickLock feature is enabled. uiParam: Not used. pvParam: Points to a DWORD value that receives the time delay.
SPI_GETMOUSEHOVERHEIGHT	**Windows 98/Me/NT/2000/XP and later**: Retrieves the height, in pixels, of the rectangle within which the mouse pointer has to stay for a WM_MOUSEHOVER message to be generated by the TrackMouseEvent function. uiParam: Not used. pvParam: Points to an integer value that receives the height.
SPI_GETMOUSEHOVERTIME	**Windows 98/Me/NT/2000/XP and later**: Retrieves the time, in milliseconds, that the mouse pointer has to stay in the hover rectangle for a WM_MOUSEHOVER message to be generated by the TrackMouseEvent function. uiParam: Not used. pvParam: Points to an integer value that receives the time.
SPI_GETMOUSEHOVERWIDTH	**Windows 98/Me/NT/2000/XP and later**: Retrieves the width, in pixels, of the rectangle within which the mouse pointer has to stay for a WM_MOUSEHOVER message to be generated by the TrackMouseEvent function. uiParam: Not used. pvParam: Points to an integer value that receives the width.

Value	Description
SPI_GETMOUSEKEYS	Retrieves information about the MouseKeys accessibility feature. MouseKeys allow the mouse cursor to be controlled by the numeric keypad. The Num Lock key toggles between mouse control and normal operation. uiParam: Specifies the size of the TMouseKeys structure. pvParam: Points to a TMouseKeys structure that receives information about the MouseKeys feature. **Note**: If accessibility features are not installed, this flag causes the function to fail.
SPI_GETMOUSESONAR	**Windows Me/XP only**: Indicates if the mouse Sonar feature is enabled or disabled. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if enabled, FALSE if disabled.
SPI_GETMOUSESPEED	**Windows 98/Me/2000/XP and later**: Retrieves the mouse speed, on a scale of 1-20 (20 being the fastest, 10 is the default). uiParam: Not used. pvParam: Points to an integer value that receives the mouse speed value.
SPI_GETMOUSETRAILS	**Windows 95/98/Me/XP only**: Indicates mouse trails are enabled. uiParam: Not used. pvParam: Pointer to an integer value. A value of 1 or 0 indicates mouse trails are disabled. A value greater than 1 indicates the number of mouse trails drawn to the screen.
SPI_GETMOUSEVANISH	**Windows Me/XP only**: Indicates if the mouse Vanish feature is enabled or disabled. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if enabled, FALSE if disabled.
SPI_GETNONCLIENTMETRICS	Retrieves metric values associated with the non-client area of a window. uiParam: Specifies the size of the TNonClientMetrics structure. pvParam: Points to a TNonClientMetrics structure that receives the non-client area metric values.
SPI_GETPOWEROFFACTIVE	Indicates if the screen saver power-off phase is enabled or disabled. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if enabled, FALSE if disabled. **Note**: Under Windows 95, this flag is supported for 16-bit applications only. Under Windows 98/Me, this is supported for 16- and 32-bit applications. Under Windows 2000/XP, only 32-bit applications are supported.

Value	Description
SPI_GETPOWEROFFTIMEOUT	Retrieves the time-out value, in seconds, of the screen saver power-off phase. uiParam: Not used. pvParam: Points to an integer value that receives the time-out value. **Note**: Under Windows 95, this flag is supported for 16-bit applications only. Under Windows 98/Me, this is supported for 16- and 32-bit applications. Under Windows 2000/XP, only 32-bit applications are supported.
SPI_GETSCREENREADER	**Windows 95/98/Me/2000/XP and later**: Indicates if a screen reviewer utility that directs textual information to an output device, such as a speech synthesizer or Braille display, is running. When this flag is set, an application should provide information in a textual format in situations where it would otherwise represent the information graphically. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if a screen reviewer utility is running or FALSE if not.
SPI_GETSCREENSAVEACTIVE	Indicates if screen savers can activate. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if screen savers can activate, FALSE if not.
SPI_GETSCREENSAVERRUNNING	**Windows 98/Me/2000/XP and later**: Indicates if a screen is currently running. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if a screen saver is running or FALSE if not.
SPI_GETSCREENSAVETIMEOUT	Retrieves the screen saver time-out value in seconds. uiParam: Not used. pvParam: Points to an integer value to receive the time-out value.
SPI_GETSELECTIONFADE	**Windows 2000/XP only**: Indicates if the selection fade effect is enabled or disabled. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if enabled, FALSE if disabled.

Value	Description
SPI_GETSERIALKEYS	**Windows 95/98/Me only**: Retrieves information about the SerialKeys accessibility feature, which interprets data from a communication device attached to a serial port as keyboard and mouse input. uiParam: Specifies the size of the TSerialKeys structure. pvParam: Points to a TSerialKeys structure that receives information about the SerialKeys feature. **Note**: If accessibility features are not installed, this flag causes the function to fail.
SPI_GETSHOWIMEUI	**Windows 98/Me/2000/XP and later**: Indicates if the IME status window is visible. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if the status window is visible, FALSE if it is not.
SPI_GETSHOWSOUNDS	Indicates if the ShowSounds accessibility feature is enabled. When this feature is enabled, applications should represent information visually when it would otherwise present it in an audible form. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if the feature is enabled, FALSE if it is not. **Note**: If accessibility features are not installed, this flag causes the function to fail.
SPI_GETSNAPTODEFBUTTON	**Windows 98/Me/NT/2000/XP and later**: Indicates if the snap-to-default-button feature is enabled. If enabled, the mouse cursor automatically moves to the default button of a dialog box, such as "OK" or "Apply." uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if the feature is enabled, FALSE if it is not.
SPI_GETSOUNDSENTRY	Retrieves information about the SoundSentry accessibility feature. When the SoundSentry feature is on, the system displays a visual indicator when a sound is generated. Under Windows 95/98/Me, the visual indicator is displayed only when a sound is generated through the internal PC speaker. Windows NT/2000/XP will display the visual indicator when a sound is generated through either the internal speaker or a multimedia sound card. uiParam: Specifies the size of the TSoundSentry structure. pvParam: Points to a TSoundSentry structure that receives information about the SoundSentry feature. **Note**: If accessibility features are not installed, this flag causes the function to fail.

Value	Description
SPI_GETSTICKYKEYS	Retrieves information about the StickyKeys accessibility feature. The StickyKeys feature allows a user to press a modifier key, such as Shift, Ctrl, or Alt, and then a second key one at a time instead of simultaneously to produce uppercase letters or other key combinations. uiParam: Specifies the size of the TStickyKeys structure. pvParam: Points to a TStickyKeys structure that receives information about the StickyKeys feature. **Note**: If accessibility features are not installed, this flag causes the function to fail.
SPI_GETTOGGLEKEYS	Retrieves information about the ToggleKeys accessibility feature. When the ToggleKeys feature is enabled, Windows outputs a high-pitched tone when the user turns on the Caps Lock, Num Lock, or Scroll Lock keys and a low-pitched tone when the user turns them off. uiParam: Specifies the size of the TToggleKeys structure. pvParam: Points to a TToggleKeys structure that receives information about the ToggleKeys feature. **Note**: If accessibility features are not installed, this flag causes the function to fail.
SPI_GETTOOLTIPANIMATION	**Windows 2000/XP and later**: Indicates if tooltip animation is enabled or disabled. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if the feature is enabled, FALSE if it is not.
SPI_GETTOOLTIPFADE	**Windows 2000/XP and later**: Indicates which menu tooltip animation effect is used, fading or sliding. If the return value is TRUE, the fade animation is used; otherwise, the slide animation is used. The SPI_GETTOOLTIPANIMATION flag determines if tooltip animation is enabled. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE for a fading effect, FALSE for a sliding effect.
SPI_GETUIEFFECTS	**Windows 2000/XP and later**: Indicates if all user interface animation effects are enabled or disabled. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if enabled, FALSE if disabled.
SPI_GETWHEELSCROLLLINES	**Windows 98/Me/NT/2000/XP and later**: Retrieves the number of lines scrolled when the mouse wheel is rotated for mice that come equipped with the mouse wheel. uiParam: Not used. pvParam: Points to an integer value that receives the number of lines scrolled.

Value	Description
SPI_GETWINDOWSEXTENSION	**Windows 95 only**: Indicates if Windows Plus! is installed. The function returns TRUE if Windows Plus! is installed, FALSE otherwise. uiParam: This parameter is always set to 1. pvParam: Not used.
SPI_GETWORKAREA	Retrieves the size of the desktop area not obscured by the taskbar. uiParam: Not used. pvParam: Points to a TRect that receives the dimensions of the work area.
SPI_ICONHORIZONTALSPACING	Sets the width of an icon cell for desktop spacing. uiParam: Specifies the width of the cell in pixels. pvParam: Not used.
SPI_ICONVERTICALSPACING	Sets the height of an icon cell for desktop spacing. uiParam: Specifies the height of the cell in pixels. pvParam: Not used.
SPI_SETACCESSTIMEOUT	Sets the time-out period associated with the accessibility features. uiParam: Specifies the size of the TAccessTimeout structure. pvParam: Points to a TAccessTimeout structure that contains the new time-out values. **Note**: If accessibility features are not installed, this flag causes the function to fail.
SPI_SETACTIVEWINDOWTRACKING	**Windows 98/Me/2000/XP and later**: Enables or disables active window tracking, which activates any window over which the mouse hovers. uiParam: Specifies zero to turn the option off or non-zero to turn it on. pvParam: Not used.
SPI_SETACTIVEWNDTRKZORDER	**Windows 98/Me/2000/XP and later**: Determines if windows activated via active window tracking are brought to the top. uiParam: Specifies non-zero to bring windows to the top or zero to turn this option off. pvParam: Not used.
SPI_SETACTIVEWNDTRKTIMEOUT	**Windows 98/Me/2000/XP and later**: Indicates active window tracking window activation delay in milliseconds. uiParam: Indicates the time delay in milliseconds. pvParam: Not used.

Value	Description
SPI_SETANIMATION	Sets the animation effect values, such as window minimizing/restoring animation. uiParam: Specifies the size of a TAnimationInfo structure. pvParam: Points to a TAnimationInfo structure that contains the new animation effect values.
SPI_SETBEEP	Turns the warning beeper on or off. uiParam: Specifies zero to turn the option off or non-zero to turn it on. pvParam: Not used.
SPI_SETBORDER	Sets the border multiplier that is used in determining the width of a window's sizing border. uiParam: Specifies the new border width multiplier. pvParam: Not used.
SPI_SETCARETWIDTH	**Windows 2000/XP and later**: Sets the width, in pixels, of the caret in edit controls. The minimum size is one pixel. uiParam: Specifies the caret width. pvParam: Not used.
SPI_SETCOMBOBOXANIMATION	**Windows 98/Me/2000/XP and later**: Enables or disables the opening animation effect for combo boxes. uiParam: Specifies zero to turn the option off or non-zero to turn it on. pvParam: Not used.
SPI_SETCURSORS	Instructs the system to reload all system cursors. uiParam: Must be set to zero. pvParam: Not used.
SPI_SETCURSORSHADOW	**Windows 2000/XP and later**: Enables or disables the shadow displayed with the cursor. uiParam: Specifies zero to turn the option off or non-zero to turn it on. pvParam: Not used. **Note**: The video driver must be set to a color depth greater than 8-bit before this effect is displayed.
SPI_SETDEFAULTINPUTLANG	Sets the default input language for the system. The specified language must be displayable using the current character set. uiParam: Not used. pvParam: Points to a keyboard layout handle for the new default language.
SPI_SETDESKPATTERN	Sets the current desktop pattern. Windows retrieves this pattern from the pattern setting in the WIN.INI file. uiParam: Not used. pvParam: Not used.

Value	Description
SPI_SETDESKWALLPAPER	Sets the desktop wallpaper. uiParam: Not used. pvParam: Points to a string that contains the name of the bitmap file to use for the wallpaper.
SPI_SETDOUBLECLICKTIME	Sets the maximum number of milliseconds that can occur between the first and second clicks of a double-click. uiParam: Specifies the new time in milliseconds. pvParam: Not used.
SPI_SETDOUBLECLKHEIGHT	Sets the height of the rectangle within which the mouse cursor must be located and the second click of a double-click must fall for it to be registered as a double-click. uiParam: Specifies the new height in pixels. pvParam: Not used.
SPI_SETDOUBLECLKWIDTH	Sets the width of the rectangle within which the mouse cursor must be located and the second click of a double-click must fall for it to be registered as a double-click. uiParam: Specifies new width in pixels. pvParam: Not used.
SPI_SETDRAGFULLWINDOWS	Sets full window dragging to on or off. uiParam: Specifies zero to disable full window dragging or non-zero to enable it. pvParam: Not used. **Note**: This flag is supported under Windows 95 only if Windows Plus! is installed.
SPI_SETDRAGHEIGHT	Sets the height, in pixels, of the rectangle used to detect the start of a mouse drag operation. uiParam: Specifies the new rectangle height value. pvParam: Not used.
SPI_SETDRAGWIDTH	Sets the width, in pixels, of the rectangle used to detect the start of a mouse drag operation. uiParam: Specifies the new rectangle width value. pvParam: Not used.
SPI_SETDROPSHADOW	**Windows XP only**: Enables or disables the drop shadow effect. uiParam: Specifies zero to turn the option off or non-zero to turn it on. pvParam: Not used.

Value	Description
SPI_SETFILTERKEYS	Sets the FilterKeys accessibility feature parameters. uiParam: Specifies the size of the TFilterKeys structure. pvParam: Points to a TFilterKeys structure that contains the new settings. **Note**: If accessibility features are not installed, this flag causes the function to fail.
SPI_SETFLATMENU	**Windows XP only**: Enables or disables flat appearance for native user menus. uiParam: Specifies zero to turn the option off or non-zero to turn it on. pvParam: Not used.
SPI_SETFOCUSBORDERHEIGHT	**Windows XP only**: Sets the height, in pixels, of the top or bottom edges of a focus rectangle drawn with the DrawFocusRect function. uiParam: Indicates the focus border height. pvParam: Not used.
SPI_SETFOCUSBORDERWIDTH	**Windows XP only**: Sets the width, in pixels, of the left or right edges of a focus rectangle drawn with the DrawFocusRect function. uiParam: Indicates the focus border width. pvParam: Not used.
SPI_SETFONTSMOOTHING	Enables or disables anti-aliased font drawing, making font curves appear smoother. uiParam: Specifies a value of zero to disable anti-aliased font curve drawing or a non-zero value to enable it. pvParam: Not used. **Note**: This flag is supported under Windows 95 only if Windows Plus! is installed.
SPI_SETFONTSMOOTHINGCONTRAST	**Windows XP only**: Sets the contrast value used in ClearType font smoothing. uiParam: Indicates the contrast value and must be in the range 1000-2200 (the default value is 1400). pvParam: Not used. **Note**: This flag is only valid when SPI_SETFONTSMOOTHINGTYPE is set to FE_FONTSMOOTHINGCLEARTYPE.
SPI_SETFONTSMOOTHINGTYPE	**Windows XP only**: Sets the font smoothing type. uiParam: Must be set to either FE_FONTSMOOTHINGSTANDARD, for standard anti-aliasing, or FE_FONTSMOOTHINGCLEARTYPE to use ClearType font smoothing. pvParam: Not used.

Value	Description
SPI_SETFOREGROUNDFLASHCOUNT	**Windows 98/Me/2000/XP and later**: Sets the number of times an application's taskbar button flashes when rejecting a foreground switch request. uiParam: Indicates the flash count. pvParam: Not used.
SPI_SETFOREGROUNDLOCKTIMEOUT	**Windows 98/Me/2000/XP and later**: Sets the number of milliseconds following user input during which no applications are allowed to force themselves into the foreground. uiParam: Indicates the time-out value in milliseconds. pvParam: Not used.
SPI_SETGRADIENTCAPTIONS	**Windows 98/Me/2000/XP and later**: Indicates if the gradient effect in application title bars is enabled or disabled. uiParam: Not used. pvParam: Points to a Boolean value that receives TRUE if enabled, FALSE if disabled.
SPI_SETGRIDGRANULARITY	Sets the granularity of the desktop sizing grid. uiParam: Specifies the new granularity value. pvParam: Not used.
SPI_SETHIGHCONTRAST	**Windows 95/98/Me/2000/XP and later**: Sets the HighContrast accessibility feature parameters. uiParam: Specifies the size of the THighContrast structure. pvParam: Points to the THighContrast structure that contains the new values. **Note**: If accessibility features are not installed, this flag causes the function to fail.
SPI_SETHOTTRACKING	**Windows 98/Me/2000/XP and later**: Enables or disables hot tracking of user interface elements (i.e., flat buttons that display a border when the mouse enters, etc.). uiParam: Specifies zero to turn the option off or non-zero to turn it on. pvParam: Not used.
SPI_SETICONMETRICS	Sets icon metrics, such as spacing and title wrap. uiParam: Specifies the size of the TIconMetrics structure. pvParam: Points to a TIconMetrics structure that contains the new values.
SPI_SETICONS	Instructs the system to reload all system icons. uiParam: Must be set to zero. pvParam: Not used.
SPI_SETICONTITLELOGFONT	Sets the logical font used for icon titles. uiParam: Specifies the size of the TLogFont structure. pvParam: Points to a TLogFont structure that contains the new values.

Value	Description
SPI_SETICONTITLEWRAP	Turns icon title wrapping on or off. uiParam: Specifies zero to disable icon title wrapping or non-zero to enable. pvParam: Not used.
SPI_SETKEYBOARDCUES	**Windows 98/Me/2000/XP and later**: Enables or disables the option to always underline menu access characters. uiParam: Specifies zero to disable the option or non-zero to enable. pvParam: Not used.
SPI_SETKEYBOARDDELAY	Sets the keyboard repeat delay setting. uiParam: Specifies the new repeat delay value in milliseconds. pvParam: Not used.
SPI_SETKEYBOARDPREF	**Windows 95/98/Me/2000/XP and later**: Sets user keyboard preference, indicating whether the user prefers the keyboard over the mouse and wants applications to display keyboard interfaces that would otherwise be hidden. uiParam: Specifies zero to indicate a keyboard preference or non-zero to indicate a mouse preference. pvParam: Not used.
SPI_SETKEYBOARDSPEED	Sets the keyboard repeat speed setting. uiParam: Specifies the new repeat speed value in milliseconds. pvParam: Not used.
SPI_SETLANGTOGGLE	Forces the system to read the hot key set from the registry for switching between input languages. uiParam: Not used. pvParam: Not used.
SPI_SETLISTBOXSMOOTHSCROLLING	**Windows 98/Me/2000/XP and later**: Enables or disables the smooth scrolling effect for list boxes. uiParam: Specifies 0 for disabled or non-zero for enabled. pvParam: Not used.
SPI_SETLOWPOWERACTIVE	Activates or deactivates the screen saver low-power phase. uiParam: Specifies zero to deactivate the low-power phase or 1 to activate it. pvParam: Not used. **Note**: Under Windows 95, this flag is supported for 16-bit applications only. Under Windows 98/Me, this is supported for 16- and 32-bit applications. Under Windows 2000/XP, only 32-bit applications are supported.
SPI_SETLOWPOWERTIMEOUT	Sets the time-out value, in seconds, for the screen saver low-power phase. uiParam: Specifies the new value in seconds. pvParam: Not used.

Value	Description
SPI_SETLOWPOWERTIMEOUT (cont.)	**Note**: Under Windows 95, this flag is supported for 16-bit applications only. Under Windows 98/Me, this is supported for 16- and 32-bit applications. Under Windows 2000/XP, only 32-bit applications are supported.
SPI_SETMENUANIMATION	**Windows 98/Me/2000/XP and later**: Enables or disables menu animation effects. uiParam: Specifies zero for disabled or non-zero for enabled. pvParam: Not used.
SPI_SETMENUDROPALIGNMENT	Sets the alignment value of drop-down menus. uiParam: Specifies zero for left alignment or non-zero for right alignment. pvParam: Not used.
SPI_SETMENUFADE	**Windows 2000/XP and later**: Indicates which menu animation effect is used, fading or sliding. A value of TRUE indicates the fade animation is used; otherwise, the slide animation is used. The SPI_SETMENUANIMATION flag determines if menu animation is enabled. uiParam: Specifies non-zero for a fading effect or zero for a sliding effect. pvParam: Not used.
SPI_SETMENUSHOWDELAY	**Windows 98/Me/NT/2000/XP and later**: Sets the amount of time, in milliseconds, before displaying a shortcut menu when the mouse is over a submenu item. uiParam: Indicates the delay amount in milliseconds. pvParam: Not used.
SPI_SETMENUUNDERLINES	See SPI_SETKEYBOARDCUES.
SPI_SETMINIMIZEDMETRICS	Sets minimized windows metrics. uiParam: Specifies the size of the TMinimizedMetrics structure. pvParam: Points to a TMinimizedMetrics structure that contains the new values.
SPI_SETMOUSE	Sets the two mouse speed threshold values and the mouse speed. uiParam: Not used. pvParam: Points to an array of three integers that contain the new values.
SPI_SETMOUSEBUTTONSWAP	Swaps or restores the left and right mouse buttons. uiParam: Specifies zero for standard mouse functionality or non-zero to swap the mouse buttons. pvParam: Not used.
SPI_SETMOUSECLICKLOCK	**Windows Me/XP only**: Enables or disables the mouse ClickLock feature. uiParam: Specifies zero to disable or non-zero to enable. pvParam: Not used.

Value	Description
SPI_SETMOUSECLICKLOCKTIME	**Windows Me/XP only**: Sets the delay, in milliseconds, between the time the mouse button is pressed and the mouse ClickLock feature activates. The SPI_SETMOUSECLICKLOCK flag indicates if the ClickLock feature is enabled. uiParam: Indicates the delay, in milliseconds. pvParam: Not used.
SPI_SETMOUSEHOVERHEIGHT	**Windows 98/Me/NT/2000/XP and later**: Sets the height, in pixels, of the rectangle within which the mouse pointer has to stay for a WM_MOUSEHOVER message to be generated by the TrackMouseEvent function. uiParam: Specifies the new height in pixels. pvParam: Not used.
SPI_SETMOUSEHOVERTIME	**Windows 98/Me/NT/2000/XP and later**: Sets the time, in milliseconds, that the mouse pointer has to stay in the hover rectangle for a WM_MOUSEHOVER message to be generated by the TrackMouseEvent function. uiParam: Specifies the new time interval in milliseconds. pvParam: Not used.
SPI_SETMOUSEHOVERWIDTH	**Windows 98/Me/NT/2000/XP and later**: Sets the width, in pixels, of the rectangle within which the mouse pointer has to stay for TrackMouseEvent to generate a WM_MOUSEHOVER message. uiParam: Specifies new width. pvParam: Not used.
SPI_SETMOUSEKEYS	Sets the MouseKeys accessibility feature parameters. uiParam: Specifies the size of a TMouseKeys structure. pvParam: Points to a TMouseKeys structure that contains the new values. **Note**: If accessibility features are not installed, this flag causes the function to fail.
SPI_SETMOUSESONAR	**Windows Me/XP only**: Enables or disables the mouse Sonar feature. uiParam: Specifies zero to disable or non-zero to enable. pvParam: Not used.
SPI_SETMOUSESPEED	**Windows 98/Me/2000/XP and later**: Sets the mouse speed on a scale of 1-20 (20 being the fastest, 10 is the default). uiParam: Specifies the mouse speed. pvParam: Not used.
SPI_SETMOUSETRAILS	**Windows 95/98/Me/XP only**: Enables or disables mouse trails. uiParam: Specifies 0 or 1 to disable mouse trails; a value greater than one indicates the number of mouse trails to draw. pvParam: Not used.n

Value	Description
SPI_SETMOUSEVANISH	**Windows Me/XP only**: Enables or disables the mouse Vanish feature. uiParam: Specifies zero to disable or non-zero to enable. pvParam: Not used.
SPI_SETNONCLIENTMETRICS	Sets the metrics values associated with the non-client area of a window. uiParam: Specifies the size of the TNonClientMetrics structure. pvParam: Points to a TNonClientMetrics structure that contains the new values.
SPI_SETPENWINDOWS	**Windows 95/98/Me only**: Specifies that Window's pen extensions are being loaded or unloaded. uiParam: Specifies zero to unload the pen extensions or non-zero to load pen extensions. pvParam: Not used.
SPI_SETPOWEROFFACTIVE	Activates or deactivates the screen saver power-off phase. uiParam: Specifies zero to deactivate or non-zero to activate. pvParam: Not used. **Note**: Under Windows 95, this flag is supported for 16-bit applications only. Under Windows 98/Me, this is supported for 16- and 32-bit applications. Under Windows 2000/XP, only 32-bit applications are supported.
SPI_SETPOWEROFFTIMEOUT	Retrieves the time-out value, in seconds, for the screen saver power-off phase. uiParam: Specifies the new value in seconds. pvParam: Not used. **Note**: Under Windows 95, this flag is supported for 16-bit applications only. Under Windows 98/Me, this is supported for 16- and 32-bit applications. Under Windows 2000/XP, only 32-bit applications are supported.
SPI_SETSCREENREADER	**Windows 95/98/Mw/2000/XP and later**: Indicates if a screen reviewer utility is running. uiParam: Specifies zero to indicate a screen reader is not present or non-zero to indicate a screen reader is present. pvParam: Not used.
SPI_SETSCREENSAVEACTIVE	Enables or disables the screen saver. uiParam: Specifies zero to disable the screen saver or non-zero to enable it. When enabling the screen saver, the last screen saver selected will be used. pvParam: Not used.
SPI_SETSCREENSAVETIMEOUT	Sets the amount of time, in seconds, that the system must be idle before the screen saver activates.

Value	Description
SPI_SETSCREENSAVETIMEOUT (cont.)	uiParam: Specifies the number of seconds to wait for the screen saver to activate. pvParam: Not used.
SPI_SETSELECTIONFADE	**Windows 2000/XP only**: Enables or disables the selection fade effect. uiParam: Specifies zero to disable or non-zero to enable. pvParam: Not used.
SPI_SETSERIALKEYS	**Windows 95/98/Me only**: Sets the SerialKeys accessibility feature parameters. uiParam: Specifies the size of the TSerialKeys structure. pvParam: Points to a TSerialKeys structure that contains the new values. **Note**: If accessibility features are not installed, this flag causes the function to fail.
SPI_SETSHOWIMEUI	**Windows 98/Me/2000/XP and later**: Enables or disables the IME status window. uiParam: Specifies zero to disable or non-zero to enable. pvParam: Not used.
SPI_SETSHOWSOUNDS	Enables or disables the ShowSounds accessibility feature. uiParam: Specifies 0 to disable the ShowSounds feature or 1 to enable it. pvParam: Not Used. **Note**: If accessibility features are not installed, this flag causes the function to fail.
SPI_SETSNAPTODEFBUTTON	**Windows 98/Me/NT/2000/XP and later**: Enables or disables the snap-to-default-button feature. uiParam: Specifies zero to disable the snap-to-default-button feature or non-zero to enable. pvParam: Not used.
SPI_SETSOUNDSENTRY	Sets the SoundSentry accessibility feature parameters. uiParam: Specifies the size of the TSoundSentry structure. pvParam: Points to a TSoundSentry structure that contains the new values. **Note**: If accessibility features are not installed, this flag causes the function to fail.
SPI_SETSTICKYKEYS	Sets the StickyKeys accessibility feature parameters. uiParam: Specifies the size of the TStickyKeys structure. pvParam: Points to a TStickyKeys structure that contains the new values. **Note**: If accessibility features are not installed, this flag causes the function to fail.

Value	Description
SPI_SETTOGGLEKEYS	Sets the ToggleKeys accessibility feature parameters. uiParam: Specifies the size of the TToggleKeys structure. pvParam: Points to a TToggleKeys structure that contains the new values. **Note**: If accessibility features are not installed, this flag causes the function to fail.
SPI_SETTOOLTIPANIMATION	**Windows 2000/XP and later**: Enables or disables tooltip animation. uiParam: Specifies zero if disabled or non-zero if enabled. pvParam: Not used.
SPI_SETTOOLTIPFADE	**Windows 2000/XP and later**: Sets which menu tooltip animation effect is used, fading or sliding. Use a non-zero value for the fade animation; otherwise, the slide animation is used. The SPI_GETTOOLTIPANIMATION flag determines if tooltip animation is enabled. uiParam: Specifies zero for a slide animation or non-zero for a fade animation. pvParam: Not used.
SPI_SETUIEFFECTS	**Windows 2000/XP and later**: Enables or disables all user interface animation effects. uiParam: Specifies zero for disabled or non-zero for enabled. pvParam: Not used.
SPI_SETWHEELSCROLLLINES	**Windows 98/Me/NT/2000/XP and later**: Sets the number of lines scrolled when the mouse wheel is rotated. uiParam: Specifies the number of lines to scroll. pvParam: Not used.
SPI_SETWORKAREA	Sets the size of the desktop area not obscured by the taskbar. uiParam: Not used. pvParam: Points to a TRect structure that contains the new work area size.

```
TAccessTimeout = packed record
    cbSize: UINT;              {the size of the TAccessTimeout structure}
    dwFlags: DWORD;            {time-out behavior properties}
    iTimeOutMSec: DWORD        {the time-out value in milliseconds}
end;
```

cbSize: Indicates the size of the TAccessTimeout record. Set this member to SizeOf(TAccessTimeout).

dwFlags: Flags indicating the behavior of the time-out options. This member can contain one or more values from the following table.

iTimeOutMSec: Indicates the length in milliseconds that must elapse without a keyboard or mouse action before the system will turn off the accessibility features.

Table 8-37: TAccessTimeout dwFlags values

Value	Description
ATF_ONOFFFEEDBACK	The system will play a sound before the accessibility features are turned off.
ATF_TIMEOUTON	The time-out interval has been set and the system will time out at that interval. If this flag is not set, the system will not time out no matter what time interval is set.

```
TAnimationInfo = packed record
    cbSize: UINT;                   {the size of the TAnimationInfo structure}
    iMinAnimate: Integer;           {enables or disables animation}
end;
```

cbSize: Specifies the size of the TAnimationInfo structure. This member should be set to SizeOf(TAnimationInfo).

iMinAnimate: Specifies if animation is enabled or disabled. A value of zero indicates that the animation is disabled; a non-zero value indicates that animation is enabled.

```
TFilterKeys = packed record
    cbSize: UINT;                   {the size of the TFilterkeys structure}
    dwFlags: DWORD;                 {sets behavior of filter keys}
    iWaitMSec: DWORD;               {acceptance delay}
    iDelayMSec: DWORD;              {repeat delay}
    iRepeatMSec: DWORD;             {repeat rate}
    iBounceMSec: DWORD;             {bounce time}
end;
```

cbSize: Specifies the size of the TFilterKeys record. This member should be set to SizeOf(TFilterKeys).

dwFlags: Indicates the behavior state of the filter keys options. This member can contain one or more values from the following table.

iWaitMSec: Specifies the time in milliseconds that the user must hold down a key before the system will accept it. This is also referred to as slow keys.

iDelayMSec: Specifies the delay interval for the repeat rate. This is the amount of time in milliseconds the user must hold down a key before the system will start to repeat that key.

iRepeatMSec: Specifies the repeat rate. This is the amount of time in milliseconds the system will wait before it repeats the keystroke again.

iBounceMSec: Specifies the bounce time for keystrokes. This is the amount of time that must pass before the system will accept another input from that key.

Table 8-38: TFilterKeys dwFlags values

Value	Description
FKF_AVAILABLE	The FilterKeys feature is available.
FKF_CLICKON	The system will click when keys are pressed. When slow keys are used, another click will sound when the key is accepted.
FKF_CONFIRMHOTKEY	**Windows 95/98/2000 only**: A dialog box will appear when enabling or disabling the FilterKeys options.
FKF_FILTERKEYSON	The FilterKeys feature is turned on.
FKF_HOTKEYACTIVE	The hot key is enabled for turning filter keys on or off. The hot key is the Shift key held down for eight seconds.
FKF_HOTKEYSOUND	The system will play a sound when the FilterKeys option is enabled or disabled.
FKF_INDICATOR	**Windows 95/98/2000 only**: Windows will display an indicator when the hot keys option is turned on.

```
THighContrast = packed record
     cbSize: UINT;                     {the size of the THighContrast structure}
     dwFlags: DWORD;                   {sets the behavior of high contrast options}
     lpszDefaultScheme: PAnsiChar;     {the name of the standard scheme}
end;
```

cbSize: Specifies the size of the THighContrast structure. Set this member to SizeOf(THighContrast).

dwFlags: Indicates the behavior state of the high contrast options. This member can contain one or more values from the following table.

lpszDefaultScheme: A pointer to a null-terminated string that contains the name of the standard color scheme for the system.

Table 8-39: THighContrast dwFlags values

Value	Description
HCF_AVAILABLE	The high contrast option is available.
HCF_CONFIRMHOTKEY	A dialog box will appear when enabling or disabling the high contrast options.
HCF_HIGHCONTRASTON	The high contrast mode is currently on.
HCF_HOTKEYACTIVE	The hot key for the high contrast mode is enabled. The hot key for turning the high contrast mode on and off is simultaneously pressing the left Alt, left Shift, and Print Screen keys.
HCF_HOTKEYAVAILABLE	Indicates if the hot key option is available on the system.
HCF_HOTKEYSOUND	The system will play a sound when the hot key is pressed to indicate that the high contrast option is enabled or disabled.
HCF_INDICATOR	Windows will display an indicator that the high contrast option is available.

```
TIconMetrics = packed record
    cbSize: UINT;               {the size of the TIconMetrics structure}
    iHorzSpacing: Integer;      {horizontal spacing for icons}
    iVertSpacing: Integer;      {vertical spacing for icons}
    iTitleWrap: Integer;        {word wrap titles}
    lfFont: TLogFont;           {the font for desktop icons}
end;
```

cbSize: Specifies the size of the TIconMetrics structure. This member should be set to SizeOf(TIconMetrics).

iHorzSpacing: Specifies the horizontal spacing for icons on the desktop.

iVertSpacing: Specifies the vertical spacing for icons on the desktop.

iTitleWrap: Indicates if icon titles are word wrapped. A zero value indicates that icon titles will not be word wrapped; a non-zero value indicates that they will be wrapped.

lfFont: Indicates the font to be used when displaying the icon title.

```
TMinimizedMetrics = packed record
    cbSize: UINT;               {the size of the TMinimizedMetrics structure}
    iWidth: Integer;            {the width of minimized windows}
    iHorzGap: Integer;          {the horizontal gap between minimized windows}
    iVertGap: Integer;          {the vertical gap between minimized windows}
    iArrange: Integer;          {minimized window arrangement}
end;
```

cbSize: Specifies the size of the TMinimizedMetrics structure. This member should be set to SizeOf(TMinimizedMetrics).

iWidth: Specifies the width of the minimized window.

iHorzGap: Specifies the horizontal space between each minimized window.

iVertGap: Specifies the vertical space between each minimized window.

iArrange: Specifies how the minimized windows are to be arranged. This member contains one value from the following table of starting positions and one value from the following table of directions.

Table 8-40: TMinimizedMetrics iArrange starting position values

Value	Description
ARW_BOTTOMLEFT	Start at the bottom-left corner of the work area.
ARW_BOTTOMRIGHT	Start at the bottom-right corner of the work area.
ARW_TOPLEFT	Start at the top-left corner of the work area.
ARW_TOPRIGHT	Start at the top-right corner of the work area.

Table 8-41: TMinimizedMetrics iArrange direction values

Value	Description
ARW_LEFT	Fill going to the left. Only valid with ARW_BOTTOMRIGHT and ARW_TOPRIGHT.
ARW_RIGHT	Fill going to the right. Only valid with ARW_BOTTOMLEFT and ARW_TOPLEFT.
ARW_UP	Fill going to the top. Only valid with ARW_BOTTOMRIGHT and ARW_BOTTOMLEFT.
ARW_DOWN	Fill going to the bottom. Only valid with ARW_TOPRIGHT and ARW_TOPLEFT.
ARW_HIDE	Tells the system to hide minimized windows. This is the default action for Windows 95.

```
TMouseKeys = packed record
     cbSize: UINT;                    {the size of the TMouseKeys structure}
     dwFlags: DWORD;                  {sets behavior of mouse key options}
     iMaxSpeed: DWORD;                {maximum mouse speed}
     iTimeToMaxSpeed: DWORD;          {time delay to maximum speed}
     iCtrlSpeed: DWORD;               {control key multiplier}
     dwReserved1: DWORD;              {reserved for future use}
     dwReserved2: DWORD;              {reserved for future use}
end;
```

cbSize: Specifies the size of the TMouseKeys structure. This member should be set to SizeOf(TMouseKeys).

dwFlags: Indicates the behavior of the mouse keys options. This member can contain one or more values from the following table.

iMaxSpeed: Specifies the maximum speed in pixels for the mouse. The value of this member may be in the range of 10 to 360.

iTimeToMaxSpeed: Specifies the time delay in milliseconds before the maximum speed is achieved. The value of this member may be in the range of 1000 and 5000.

iCtrlSpeed: Indicates the multiplier to add to the speed if the Ctrl key is held down. This is only available if the dwFlags member contains the MKF_MODIFIERS flag.

dwReserved1: Reserved for future use.

dwReserved2: Reserved for future use.

Table 8-42: TMouseKeys dwFlags values

Value	Description
MKF_AVAILABLE	The mouse key option is available.
MKF_CONFIRMHOTKEY	**Windows 95/98/2000 only**: A dialog box will appear when enabling or disabling the mouse keys options.

Value	Description
MKF_HOTKEYACTIVE	The hot key for the mouse keys mode is enabled. The hot key for turning mouse keys on and off is Left Alt+Left Shift+Num Lock.
MKF_HOTKEYSOUND	The system will play a sound when the mouse keys are enabled or disabled by using the hot keys.
MKF_INDICATOR	**Windows 95/98/2000 only**: Windows will display an indicator when mouse keys are turned on.
MKF_MOUSEKEYSON	The mouse keys are currently on.
MKF_MODIFIERS	**Windows 95/98/2000 only**: Indicates if the Ctrl and Alt keys will effect the mouse movement.
MKF_MOUSEMODE	**Windows 98/2000 only**: Numeric keypad input is processed as mouse movement.
MKF_REPLACENUMBERS	**Windows 95/98/2000 only**: Indicates if the mouse will be moved if the Num Lock key is on or off. If this flag is not specified, the numeric keypad will move the mouse cursor when the Num Lock key is off.
MKF_LEFTBUTTONSEL	**Windows 98/2000 only**: The left mouse button is used for mouse button actions.
MKF_RIGHTBUTTONSEL	**Windows 98/2000 only**: The right mouse button is used for mouse button actions.
MKF_LEFTBUTTONDOWN	**Windows 98/2000 only**: Indicates that the left button is down.
MKF_RIGHTBUTTONDOWN	**Windows 98/2000 only**: Indicates that the right button is down.

```
TNonClientMetrics = packed record
    cbSize: UINT;                        {the size of TNonClientMetrics structure}
    iBorderWidth: Integer;               {sizing border width}
    iScrollWidth: Integer;               {standard scroll bar width}
    iScrollHeight: Integer;              {standard scroll bar height}
    iCaptionWidth: Integer;              {width of caption buttons}
    iCaptionHeight: Integer;             {height of caption buttons}
    lfCaptionFont: TLogFont;             {font to use in the caption bar}
    iSmCaptionWidth: Integer;            {width for toolbar buttons}
    iSmCaptionHeight: Integer;           {height for toolbar buttons}
    lfSmCaptionFont: TLogFont;           {font to use in the toolbar}
    iMenuWidth: Integer;                 {menu bar button width}
    iMenuHeight: Integer;                {menu bar button height}
    lfMenuFont: TLogFont;                {font to use in menu bar}
    lfStatusFont: TLogFont;              {status bar font}
    lfMessageFont: TLogFont;             {message box font}
end;
```

cbSize: Specifies the size of the TNonClientMetrics structure. This member should be set to SizeOf(TNonClientMetrics).

iBorderWidth: Width of the window border for a sizable window.

iScrollWidth: Width of a standard vertical scroll bar.

iScrollHeight: Height of a standard horizontal scroll bar.

iCaptionWidth: Width of the caption bar buttons.

iCaptionHeight: Height of the caption bar buttons.

lfCaptionFont: Font to use in the caption bar.

iSmCaptionWidth: Width of the buttons in a toolbar windows caption.

iSmCaptionHeight: Height of the buttons in a toolbar windows caption.

lfSmCaptionFont: Font to use in a toolbar caption.

iMenuWidth: Width of the buttons that appear in a menu bar.

iMenuHeight: Height of the buttons that appear in a menu bar.

lfMenuFont: Font to use in a menu bar.

lfStatusFont: Font to use in a status bar.

lfMessageFont: Font to use in a message dialog box.

```
TSoundSentry = packed record
    cbSize: UINT;                       {the size of the TSoundSentry structure}
    dwFlags: DWORD;                     {sets behavior of sound sentry option}
    iFSTextEffect: DWORD;               {text app sound effect}
    iFSTextEffectMSec: DWORD;           {length of text app sound effect}
    iFSTextEffectColorBits: DWORD;      {color of text app sound effect}
    iFSGrafEffect: DWORD;               {graphic app sound effect}
    iFSGrafEffectMSec: DWORD;           {length of graphic app sound effect}
    iFSGrafEffectColor: DWORD;          {color of graphic app sound effect}
    iWindowsEffect: DWORD;              {Windows app sound effect}
    iWindowsEffectMSec: DWORD;          {length of Windows app sound effect}
    lpszWindowsEffectDLL: PAnsiChar;    {DLL that contains special sound
                                         effect}
    iWindowsEffectOrdinal: DWORD;       {reserved for future use}
end;
```

cbSize: Specifies the size of the TSoundSentry structure. Set this member to SizeOf(TSoundSentry).

dwFlags: Indicates the behavior of the sound sentry options. This member can contain one value from Table 8-43.

iFSTextEffect: Indicates the behavior of the sound sentry options when a text-based application is running in a full screen window. This member may contain one value from Table 8-44. This member is not available under Windows NT/2000 and must be set to zero.

iFSTextEffectMSec: Specifies how long the text effect change will last in milliseconds. This member is not available under Windows NT/2000 and must be set to zero.

iFSTextEffectColorBits: Specifies the color that will be used for the text change effect. This member is not available under Windows NT/2000 and must be set to zero.

iFSGrafEffect: Indicates the behavior of the sound sentry options when a graphic-based application is running in a full screen window. This member can contain one value from Table 8-45. This member is not available under Windows NT/2000 and must be set to zero.

iFSGrafEffectMSec: Specifies how long the graphic effect change will last in milliseconds. This member is not available under Windows NT/2000 and must be set to zero.

iFSGrafEffectColor: Specifies the color that will be used for the graphic change effect. This member is not available under Windows NT/2000 and must be set to zero.

iWindowsEffect: Indicates the behavior of the sound sentry options when a Windows-based application is running. This member can contain one value from Table 8-46.

iWindowsEffectMSec: Specifies how long the Windows effect change will last in milliseconds.

lpszWindowsEffectDLL: Specifies the name of the DLL that contains an exported SoundSentryProc callback function. This function will be called when a sound is generated. This member can be set to NIL if a DLL is not used.

iWindowsEffectOrdinal: Reserved for future use. This member must be set to zero.

Table 8-43: TSoundSentry dwFlags values

Value	Description
SSF_AVAILABLE	Indicates that the SoundSentry feature is available.
SSF_SOUNDSENTRYON	Indicates that the SoundSentry feature is currently on.

Table 8-44: TSoundSentry iFSTextEffect values

Value	Description
SSTF_BORDER	Flashes the screen border. This option is not available on all displays.
SSTF_CHARS	Flashes a character in the upper corner of the screen.
SSTF_DISPLAY	Flashes the entire display.
SSTF_NONE	No visual sound indicator.

Table 8-45: TSoundSentry iFSGrafEffect values

Value	Description
SSGF_DISPLAY	Flashes the entire display.
SSGF_NONE	No visual sound signal.

Table 8-46: TSoundSentry iWindowsEffect values

Value	Description
SSWF_CUSTOM	Call the SoundSentryProc function exported by the DLL specified by the lpszWindowsEffectDLL member.
SSWF_DISPLAY	Flashes the entire display.
SSWF_NONE	No visual sound signal.
SSWF_TITLE	Flashes the title bar of the active window.
SSWF_WINDOW	Flashes the active window.

```
TStickyKeys = packed record
    cbSize: UINT;          {the size of the TStickyKeys structure}
    dwFlags: DWORD;        {sets the behavior of sticky keys options}
end;
```

cbSize: Specifies the size of the TStickyKeys structure. Set this member to SizeOf(TStickyKeys).

dwFlags: Indicates the behavior of the sticky keys options. This member can contain one or more values from the following table.

Table 8-47: TStickyKeys dwFLags values

Value	Description
SKF_AUDIBLEFEEDBACK	The system will play a sound any time the Ctrl, Alt, or Shift key is turned on.
SKF_AVAILABLE	Indicates that the StickyKeys feature is available.
SKF_CONFIRMHOTKEY	**Windows 95/98/2000 only**: A dialog box will appear when enabling or disabling the sticky keys options.
SKF_HOTKEYACTIVE	Enables or disables the StickyKeys feature hot key. The hot key is pressing the Shift key five times.
SKF_HOTKEYSOUND	The system will play a sound when the hot key is used to enable or disable sticky keys.
SKF_INDICATOR	**Windows 95/98/2000 only**: Windows will display an indicator if sticky keys are on.
SKF_STICKYKEYSON	The StickyKeys feature is turned on.
SKF_TRISTATE	Pressing a modifier key twice in a row locks that key until it is pressed a third time.
SKF_TWOKEYSOFF	Turns sticky keys off when releasing a modifier key that has been pressed in combination with any other key.
SKF_LALTLATCHED	**Windows 98/2000 only**: Indicates the left ALT key is latched.
SKF_LCTLLATCHED	**Windows 98/2000 only**: Indicates the left CTRL key is latched.
SKF_LSHIFTLATCHED	**Windows 98/2000 only**: Indicates the left Shift key is latched.
SKF_RALTLATCHED	**Windows 98/2000 only**: Indicates the right ALT key is latched.

Value	Description
SKF_RCTLLATCHED	**Windows 98/2000 only**: Indicates the right CTRL key is latched.
SKF_RSHIFTLATCHED	**Windows 98/2000 only**: Indicates the right Shift key is latched.
SKF_LALTLOCKED	**Windows 98/2000 only**: Indicates the left ALT key is locked.
SKF_LCTLLOCKED	**Windows 98/2000 only**: Indicates the left CTRL key is locked.
SKF_LSHIFTLOCKED	**Windows 98/2000 only**: Indicates the left Shift key is locked.
SKF_RALTLOCKED	**Windows 98/2000 only**: Indicates the right ALT key is locked.
SKF_RCTLLOCKED	**Windows 98/2000 only**: Indicates the right CTRL key is locked.
SKF_RSHIFTLOCKED	**Windows 98/2000 only**: Indicates the right Shift key is locked.
SKF_LWINLATCHED	**Windows 98/2000 only**: Indicates the left Windows key is latched.
SKF_RWINLATCHED	**Windows 98/2000 only**: Indicates the right Windows key is latched.
SKF_LWINLOCKED	**Windows 98/2000 only**: Indicates the left Windows key is locked.
SKF_RWINLOCKED	**Windows 98/2000 only**: Indicates the right Windows key is locked.

```
TToggleKeys = packed record
     cbSize: UINT;           {the size of the TToggleKeys structure}
     dwFlags: DWORD;         {sets the behavior of toggle keys options}
end;
```

cbSize: Specifies the size of the TToggleKeys structure. Set this member to SizeOf(TToggleKeys).

dwFlags: Indicates the behavior of the toggle keys options. This member can contain one or more values from the following table.

Table 8-48: TToggleKeys dwFlags values

Value	Description
TKF_AVAILABLE	Indicates that the ToggleKeys feature is available.
TKF_CONFIRMHOTKEY	**Windows 95/98/2000 only**: A dialog box will appear when enabling or disabling the toggle keys options.
TKF_HOTKEYACTIVE	Enables or disables the ToggleKeys option hot key. The hot key is pressing the Num Lock key for eight seconds.
TKF_HOTKEYSOUND	The system will play a sound when the hot key is used to enable or disable the ToggleKeys option.
TKF_TOGGLEKEYSON	Indicates that the ToggleKeys feature is on.

```
TSerialKeys = packed record
     cbSize: UINT;           {the size of the TSerialKeys structure}
     dwFlags: DWORD;         {sets behavior of serial keys option}
```

```
    lpszActivePort: PAnsiChar;  {name of active port}
    lpszPort: PAnsiChar;        {reserved}
    iBaudRate: UINT;            {port baud rate}
    iPortState: UINT;           {reaction state of port}
    iActive: UINT;              {reserved}
end;
```

cbSize: Specifies the size of the TSerialKeys structure. This member should be set to SizeOf(TSerialKeys).

dwFlags: Indicates the behavior of the serial keys options. This member can contain one or more values from Table 8-49.

lpszActivePort: Indicates the name of the serial port to receive user input. This member can be set to Auto to instruct the system to monitor all unused serial ports.

lpszPort: This member is reserved and must be set to NIL.

iBaudRate: Specifies the current baud rate of the serial port identified by the lpszActivePort parameter. This member can contain one value from Table 8-50.

iPortState: Specifies the state of the serial port identified by the lpszActivePort parameter. This member can contain one value from Table 8-51.

iActive: Reserved for future use.

Table 8-49: TSerialKeys dwFlags values

Value	Description
SERKF_ACTIVE	The SerialKeys option is currently receiving input on the serial port specified by lpszActivePort.
SERKF_AVAILABLE	Indicates that the SerialKeys feature is available.
SERKF_SERIALKEYSON	Indicates that the SerialKeys feature is on.

Table 8-50: TSerialKeys iBaudRate values

Value	Description
CBR_110	110 baud
CBR_300	300 baud
CBR_600	600 baud
CBR_1200	1200 baud
CBR_2400	2400 baud
CBR_4800	4800 baud
CBR_9600	9600 baud
CBR_14400	14,400 baud
CBR_19200	19,200 baud
CBR_38400	38,400 baud
CBR_56000	56,000 baud
CBR_57600	57,600 baud
CBR_115200	115,200 baud

Value	Description
CBR_128000	128,000 baud
CBR_256000	256,000 baud

Table 8-51: TSerialKeys iPortState values

Value	Description
0	This port is ignored.
1	This port is watched for SerialKeys activation sequences when no other application has the port open.
2	All input on this port is treated as SerialKeys commands.

VerLanguageName Windows.pas

Syntax

```
VerLanguageName(
wLang: DWORD;            {the language identifier}
szLang: PChar;           {the buffer receiving the language name}
nSize: DWORD             {the maximum size of the buffer}
): DWORD;                {returns the number of bytes written to the buffer}
```

Description

This function retrieves a string describing the name of the language identified by the wLang parameter.

Parameters

wLang: Specifies the language identifier from which to retrieve the language name. This parameter can be set to the return value of GetSystemDefaultLangID, GetUser-DefaultLangID, or one value from the following table.

szLang: A pointer to a null-terminated string buffer receiving the name of the language. If this parameter is set to NIL, the function returns the required size of the buffer to hold the name of the language.

nSize: Specifies the maximum size of the szLang buffer.

Return Value

If the function succeeds, it returns the number of characters copied to the szLang buffer, not including the null terminator; otherwise, the function returns zero.

See Also

GetSystemDefaultLangID, GetUserDefaultLangID, VerQueryValue

Example

Please see Listing 8-12 under GetSystemDefaultLangID.

Table 8-52: VerLanguageName wLang values

Value	Description
$0000	Language neutral
$0400	The default process language
$0401	Arabic (Saudi Arabia)
$0801	Arabic (Iraq)
$0C01	Arabic (Egypt)
$1001	Arabic (Libya)
$1401	Arabic (Algeria)
$1801	Arabic (Morocco)
$1C01	Arabic (Tunisia)
$2001	Arabic (Oman)
$2401	Arabic (Yemen)
$2801	Arabic (Syria)
$2C01	Arabic (Jordan)
$3001	Arabic (Lebanon)
$3401	Arabic (Kuwait)
$3801	Arabic (U.A.E.)
$3C01	Arabic (Bahrain)
$4001	Arabic (Qatar)
$0402	Bulgarian
$0403	Catalan
$0404	Chinese (Taiwan)
$0804	Chinese (PRC)
$0C04	Chinese (Hong Kong)
$1004	Chinese (Singapore)
$0405	Czech
$0406	Danish
$0407	German (Standard)
$0807	German (Swiss)
$0C07	German (Austrian)
$1007	German (Luxembourg)
$1407	German (Liechtenstein)
$0408	Greek
$0409	English (United States)
$0809	English (United Kingdom)
$0C09	English (Australian)
$1009	English (Canadian)
$1409	English (New Zealand)
$1809	English (Ireland)

Value	Description
$1C09	English (South Africa)
$2009	English (Jamaica)
$2409	English (Caribbean)
$2809	English (Belize)
$2C09	English (Trinidad)
$040A	Spanish (Traditional sort)
$080A	Spanish (Mexican)
$0C0A	Spanish (Modern sort)
$100A	Spanish (Guatemala)
$140A	Spanish (Costa Rica)
$180A	Spanish (Panama)
$1C0A	Spanish (Dominican Republic)
$200A	Spanish (Venezuela)
$240A	Spanish (Colombia)
$280A	Spanish (Peru)
$2C0A	Spanish (Argentina)
$300A	Spanish (Ecuador)
$340A	Spanish (Chile)
$380A	Spanish (Uruguay)
$3C0A	Spanish (Paraguay)
$400A	Spanish (Bolivia)
$440A	Spanish (El Salvador)
$480A	Spanish (Honduras)
$4C0A	Spanish (Nicaragua)
$500A	Spanish (Puerto Rico)
$040B	Finnish
$040C	French (Standard)
$080C	French (Belgian)
$0C0C	French (Canadian)
$100C	French (Swiss)
$140C	French (Luxembourg)
$040D	Hebrew
$040E	Hungarian
$040F	Icelandic
$0410	Italian (Standard)
$0810	Italian (Swiss)
$0411	Japanese
$0412	Korean
$0812	Korean (Johab)

Value	Description
$0413	Dutch (Standard)
$0813	Dutch (Belgian)
$0414	Norwegian (Bokmal)
$0814	Norwegian (Nynorsk)
$0415	Polish
$0416	Portuguese (Brazilian)
$0816	Portuguese (Standard)
$0418	Romanian
$0419	Russian
$041A	Croatian
$0C1A	Serbian
$041B	Slovak
$041C	Albanian
$041D	Swedish
$081D	Swedish (Finland)
$041E	Thai
$041F	Turkish
$0421	Indonesian
$0422	Ukrainian
$0423	Belorussian
$0424	Slovenian
$0425	Estonian
$0426	Latvian
$0427	Lithuanian
$081A	Serbian
$0429	Farsi
$042D	Basque
$0436	Afrikaans
$0438	Faeroese

Chapter 9

Icon, Cursor, and Caret Functions

Windows, being the graphical environment that it is, displays information in a variety of ways, most obviously by means of simple graphics. Various images are used to portray the type of file being viewed in the Explorer or what type of action is available to the user, depending on the current position of the mouse cursor. The Windows functions concerned with the creation and manipulation of icon, cursor, and caret images give the developer a variety of means by which to communicate specific information or available actions.

Carets

The caret is a small, flashing image used to indicate which window currently has the keyboard focus and can accept text input. Since only one window at a time can have the keyboard focus, there is only one caret in the system. In Delphi, the default caret used by components that accept text is a thin, vertical line, and Delphi encapsulates the caret functions so completely that the developer will likely never have to be concerned with them. However, if a new caret shape is desired, the Windows caret functions allow the developer to specify a caret shape in terms of a desired width and height or based on a bitmap. See the CreateCaret function for an example of creating a new caret shape based on a bitmap. The following example demonstrates how to create a solid black box caret.

Listing 9-1: Creating a solid black box caret

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   {we must set focus to the window we want to type in. when
    this window receives focus manually (i.e., by using the TAB
    key), Delphi automatically reassigns the appropriate caret}
   Memo1.SetFocus;

   {hide the current caret}
   HideCaret(0);

   {destroy the current caret}
   DestroyCaret;
```

```
    {create the new caret shape (a solid black box)}
    CreateCaret(Memo1.Handle, 0, 10, 12);

    {display the new caret image}
    ShowCaret(0);
end;
```

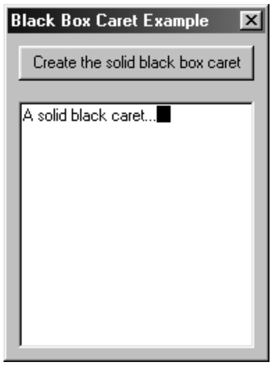


Figure 9-1: The black box caret shape

Icon and Cursor Masks

Windows does not have a native API function that copies a bitmap to a destination while interpreting some of the pixels as "transparent." However, icons and cursors have irregular shapes that, when drawn to the screen, allow the background to show through. This is accomplished by means of Boolean raster operations and masks.

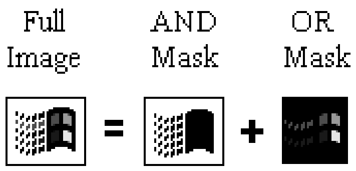


Figure 9-2: The icon and cursor image is a composite of an AND mask and an OR mask

Each icon and cursor is composed of two bitmap images known as masks: an AND mask and an OR mask. These images are combined using Boolean raster operations with the background image of the destination in two steps to create a final image exhibiting "transparent" pixels.

First, the AND mask is combined with the background image on the destination device context using the Boolean AND operator. The white pixels of the AND mask will preserve those pixels in the destination, while the black pixels of the AND mask will change the pixels in the destination to black, thereby carving out a space for the final image:

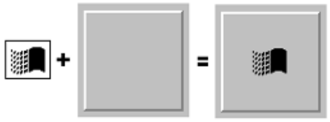


Figure 9-3: First step — combine the AND mask with the destination

Once the AND mask is combined with the destination, the OR mask is combined with the background image on the destination device context using the Boolean OR operator. The black pixels of the OR mask will preserve those pixels in the destination, while the colored pixels, which should fall within the black pixels created by the first step, should show up as they appear in the OR mask, thereby creating the final image with the illusion of transparent pixels:

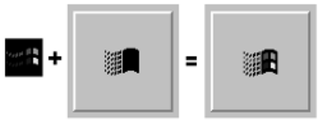


Figure 9-4: Last step — combine the OR mask with the destination

It is this method of bitmap merging that allows icons and cursors to have irregular shapes with the background showing through the "transparent" areas. The same technique can be used to display bitmaps transparently by using a combination of masks with the BitBlt function and the SRCAND and SRCPAINT raster operation codes. The GetIconInfo function can be used to retrieve the AND and OR masks for both icons and cursors.

Icon to Bitmap Conversion

The complementary nature of the Windows bitmap, cursor, and icon functions provides the developer a means by which a bitmap can be converted into an icon or cursor, or vice versa. With a few simple API calls, combined with Delphi's power and ease of use, the developer can create applications that could potentially use bitmaps, cursors, and icons interchangeably. The following example demonstrates a method by which any bitmap can be converted into an icon or any icon converted into a bitmap.

Listing 9-2: Converting icons to bitmaps and back

```
var
  Form1: TForm1;
  CurIcon: TIcon;        // holds an icon
  CurBitmap: TBitmap;    // holds a bitmap

implementation

{$R *.DFM}
```

```
procedure TForm1.FileListBox1DblClick(Sender: TObject);
begin
  {open the selected file as an icon (automatically converts bitmaps to icons}
  CurIcon.Handle := ExtractIcon(hInstance, PChar(FileListBox1.FileName), 0);

  {enable the save image button}
  Button1.Enabled := TRUE;

  {erase the paintbox}
  PaintBox1.Canvas.Brush.Color := clBtnFace;
  PaintBox1.Canvas.FillRect(PaintBox1.ClientRect);

  {if the user wants to convert icons to bitmaps...}
  if RadioButton1.Checked then
  begin
    {erase the current bitmap image}
    CurBitmap.Canvas.Brush.Color := clBtnFace;
    CurBitmap.Canvas.FillRect(PaintBox1.ClientRect);

    {draw the icon onto the bitmap}
    DrawIcon(CurBitmap.Canvas.Handle, 0, 0, CurIcon.Handle);

    {display the bitmap image}
    PaintBox1.Canvas.Draw((PaintBox1.Width div 2)-16,
                          (PaintBox1.Height div 2)-16, CurBitmap);
  end
  else
    {display the icon}
    DrawIcon(PaintBox1.Canvas.Handle, (PaintBox1.Width div 2)-16,
             (PaintBox1.Height div 2)-16, CurIcon.Handle);
  end;

procedure TForm1.RadioButton1Click(Sender: TObject);
begin
  {if the user wants to convert icons to bitmaps...}
  if Sender=RadioButton1 then
  begin
    {filter files by icons only}
    FileListBox1.Mask := '*.ico';

    {initialize the save picture dialog accordingly}
    SavePictureDialog1.Filter := 'Bitmaps (*.bmp)|*.bmp';
    SavePictureDialog1.DefaultExt := '*.bmp';
  end
  else
  begin
    {otherwise, filter files by bitmaps only}
    FileListBox1.Mask := '*.bmp';

    {initialize the save picture dialog accordingly}
    SavePictureDialog1.Filter := 'Icons (*.ico)|*.ico';
    SavePictureDialog1.DefaultExt := '*.ico';
  end;

  {erase the current paintbox image}
  PaintBox1.Canvas.Brush.Color := clBtnFace;
  PaintBox1.Canvas.FillRect(PaintBox1.ClientRect);

  {disable the save image button until the user selects a file to convert}
  Button1.Enabled := FALSE;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  {create the icon to hold converted bitmaps}
  CurIcon := TIcon.Create;

  {create and initialize the bitmap to hold converted icons}
  CurBitmap := TBitmap.Create;
  CurBitmap.Width := GetSystemMetrics(SM_CXICON);
  CurBitmap.Height := GetSystemMetrics(SM_CYICON);

  {initialize the save picture dialog for saving bitmaps}
  SavePictureDialog1.Filter := 'Bitmaps (*.bmp)|*.bmp';
  SavePictureDialog1.DefaultExt := '*.bmp';
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  {delete the last specified filename}
  SavePictureDialog1.FileName := '';

  {show the dialog box}
  if SavePictureDialog1.Execute then
    if RadioButton1.Checked then
      {as indicated, save the file as a bitmap...}
      CurBitmap.SaveToFile(SavePictureDialog1.FileName)
    else
      {...or icon}
      CurIcon.SaveToFile(SavePictureDialog1.FileName);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  {free the resources}
  CurIcon.Free;
  CurBitmap.Free;
end;
```

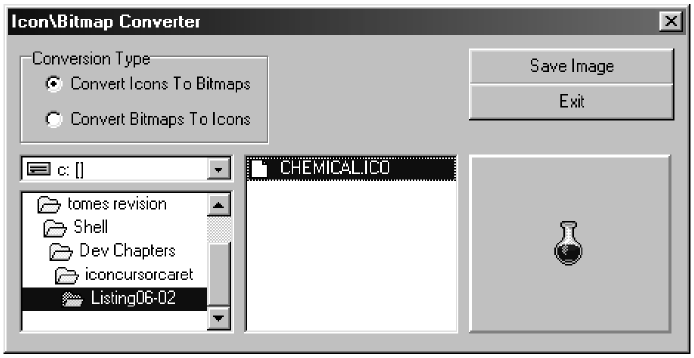


Figure 9-5: Converting an icon

Delphi vs. the Windows API

While Delphi does supply a TIcon object, there is very little support for cursors and almost no support for carets. The TIcon object will be the most useful for standard functions, such as loading and displaying an icon. However, the TIcon object doesn't have support for loading from a resource, and there is a lot more functionality provided by the Windows API than what has been encapsulated by this object. Cursors are similarly easy to use in Delphi thanks to the Cursor property. However, as with icons, the developer has a wider range of manipulation by using the Windows API. If you want a custom caret, the Windows API provides the only functionality for this type of feature.

Icon, Cursor, and Caret Functions

The following icon, cursor, and caret functions are covered in this chapter.

Table 9-1: Icon, cursor, and caret functions

Function	Description
CopyIcon	Creates a copy of an existing icon.
CreateCaret	Creates a new caret.
CreateCursor	Creates a new cursor.
CreateIcon	Creates a new icon.
CreateIconFromResource	Creates a new icon or cursor from resource information.
CreateIconFromResourceEx	Creates a new icon or cursor from resource information with a specified width and height.
CreateIconIndirect	Creates an icon or cursor from a data structure.
DestroyCaret	Destroys a caret.
DestroyCursor	Destroys a cursor.
DestroyIcon	Destroys an icon.
DrawIcon	Draws an icon at a specified location.
DrawIconEx	Draws an icon or cursor at a specified location.
ExtractAssociatedIcon	Retrieves a handle to an icon for the executable file associated with a specified file.
ExtractIcon	Retrieves a handle to an icon from a specified file.
ExtractIconEx	Retrieves a handle to the large and small icons from a specified file.
GetCursor	Retrieves a handle to the current cursor.
GetIconInfo	Retrieves information about an icon or cursor.
HideCaret	Hides the caret.
LoadCursor	Loads a cursor from the executable's resources.
LoadCursorFromFile	Loads a cursor from a file.
LoadIcon	Loads an icon from the executable's resources.
LookupIconIdFromDirectory	Searches through resource data for a compatible icon or cursor.

Function	Description
LookupIconIdFromDirectoryEx	Searches through resource data for a compatible icon or cursor with the specified width and height.
SetCursor	Sets the cursor to the specified cursor.
SetSystemCursor	Sets a system cursor to the specified cursor.
ShowCaret	Displays the caret if it was hidden.
ShowCursor	Displays or hides the cursor.

CopyIcon Windows.pas

Syntax

```
CopyIcon(
hIcon: HICON             {a handle to the icon to copy}
): HICON;                {returns a handle to an icon}
```

Description

This function makes an exact duplicate of the specified icon, returning its handle. This can be used to copy icons belonging to other modules.

Parameters

hIcon: A handle to the icon being copied.

Return Value

If the function succeeds, it returns a handle to an exact copy of the specified icon; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

DrawIcon, DrawIconEx

Example

Listing 9-3: Copying the application icon

```
procedure TForm1.Button1Click(Sender: TObject);
var
  IconCopy: HICON;       // holds a handle to the duplicated icon
begin
  {make a copy of the application icon...}
  IconCopy := CopyIcon(Application.Icon.Handle);

  {...and display it}
  DrawIcon(PaintBox1.Canvas.Handle, (PaintBox1.Width div 2)-16,
           (PaintBox1.Height div 2)-16, IconCopy);
end;
```

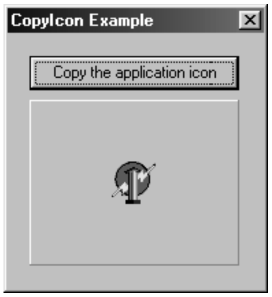


Figure 9-6: The duplicated application icon

CreateCaret Windows.ps

Syntax

```
CreateCaret(
hWnd: HWND;                 {a handle to the owner windows}
hBitmap: HBITMAP;           {a handle to a bitmap}
nWidth: Integer;            {the width of the caret}
nHeight: Integer            {the height of the caret}
): BOOL;                    {returns TRUE or FALSE}
```

Description

This function creates a new shape for the caret. The caret is assigned to a window and can be either a line, a block, or a bitmap. If a bitmap is specified, the bitmap determines the width and height of the caret shape. Otherwise, the width and height are in terms of logical units, and the exact dimensions are dependent upon the current mapping mode. The developer can retrieve the default width and height values used for a caret by calling the GetSystemMetrics function using the SM_CXBORDER and SM_CYBORDER flags. The CreateCaret function automatically destroys the previous caret shape, and the caret will not be visible until the ShowCaret function is called.

Parameters

hWnd: A handle to the window that will own the caret.

hBitmap: A handle to the bitmap used as the caret shape. If this parameter is zero, the caret will be a solid rectangle. If this parameter is set to (hBitmap) 1, the caret will be gray.

nWidth: The width of the caret in logical units. If this parameter is set to zero, the system-defined window border width is used as the default width of the cursor. If the hBitmap parameter is set to the handle of a bitmap, the bitmap determines the width, and this parameter is ignored.

nHeight: The height of the caret in logical units. If this parameter is set to zero, the system-defined window border height is used as the default height of the cursor. If the hBitmap parameter is set to the handle of a bitmap, the bitmap determines the height, and this parameter is ignored.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CreateBitmap*, CreateDIBitmap*, DestroyCaret, GetSystemMetrics*, HideCaret, LoadBitmap*, LoadImage*, ShowCaret

Example

Listing 9-4: Creating a new caret shape

```
procedure TForm1.Button1Click(Sender: TObject);
var
   TheCaretBitmap: HBitmap;     // a handle to the new caret bitmap
begin
   {load the caret bitmap from an external file}
   TheCaretBitmap := LoadImage(0,'NewCaret.bmp',IMAGE_BITMAP,0,0,
                              LR_DEFAULTSIZE OR LR_LOADFROMFILE);

   {we must set focus to the window we want to type in. when
    this window receives focus manually (i.e., by using the TAB
    key), Delphi automatically reassigns the appropriate caret}
   Memo1.SetFocus;

   {hide the current caret}
   HideCaret(0);

   {destroy the current caret}
   DestroyCaret;

   {create the new caret shape from the loaded bitmap}
   CreateCaret(Memo1.Handle,TheCaretBitmap,0,0);

   {display the new caret image}
   ShowCaret(0);
end;
```

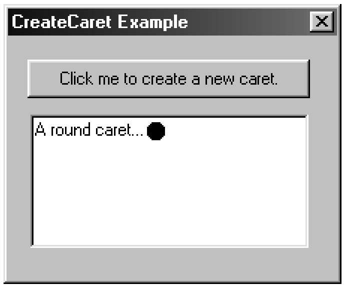


Figure 9-7: The new caret

CreateCursor Windows.pas

Syntax

```
CreateCursor(
hInst: HINST;               {a handle to the current application instance}
xHotSpot: Integer;          {the horizontal position of the cursor hot spot}
yHotSpot: Integer;          {the vertical position of the cursor hot spot}
nWidth: Integer;            {the width of the cursor in pixels}
nHeight: Integer;           {the height of the cursor in pixels}
pvANDPlaneter: Pointer;     {a pointer to the AND image data}
pvXORPlane: Pointer         {a pointer to the XOR image data}
): HCURSOR;                 {returns a handle to the cursor}
```

Description

This function creates a new cursor with the specified dimensions, image, and hot spot. This cursor can be added to Delphi’s screen cursors array to make it persistent.

Parameters

hInst: A handle to the current application instance.

xHotSpot: The horizontal coordinate of the cursor’s hot spot.

yHotSpot: The vertical coordinate of the cursor’s hot spot.

nWidth: The width of the cursor in pixels. Use GetSystemMetrics(SM_CXCURSOR) to determine the display driver-supported cursor width.

nHeight: The height of the cursor in pixels. Use GetSystemMetrics(SM_CYCURSOR) to determine the display driver-supported cursor height.

pvANDPlaneter: A pointer to an array of bytes containing the bit values for the AND mask of the cursor. This array contains information in the format of a device-dependent monochrome bitmap.

pvXORPlane: A pointer to an array of bytes containing the bit values for the XOR mask of the cursor. This array contains information in the format of a device-dependent monochrome bitmap.

Return Value

If the function succeeds, it returns a handle to the new cursor; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

CreateIcon, DestroyCursor, GetCursor, GetSystemMetrics*, SetCursor

Example

Listing 9-5: Creating a new cursor

```
var
  Form1: TForm1;
  OldCursor: HCURSOR;   // preserves the old cursor
  NewCursor: HCURSOR;   // a handle to the new cursor

implementation

procedure TForm1.Button1Click(Sender: TObject);
var
  MaskSize: Integer;    // holds the computed size of the cursor
  AndMask,              // cursor bit arrays
  XorMask: ^Byte;
  AndImage,             // intermediate bitmaps used to define the cursor shape
  XorImage: TBitmap;
begin
  {compute the size of the cursor bit arrays}
  MaskSize := (GetSystemMetrics(SM_CXICON) div 8)*GetSystemMetrics(SM_CYICON);

  {create the bitmap used to define the AND mask shape}
  AndImage := TBitmap.Create;
  with AndImage do
  begin
    {we are creating a black and white cursor}
    Monochrome := TRUE;

    {set the dimensions to those reported by the system}
    Width := GetSystemMetrics(SM_CXICON);
    Height := GetSystemMetrics(SM_CYICON);

    {create the shape of an X}
    Canvas.Brush.Color := clWhite;
    Canvas.Pen.Color := clBlack;
    Canvas.FillRect(Canvas.ClipRect);
    Canvas.MoveTo(0,0);
    Canvas.LineTo(Width,Height);
    Canvas.MoveTo(Width,0);
    Canvas.LineTo(0,Height);
  end;

  {create the bitmap used to define the XOR mask shape}
  XorImage := TBitmap.Create;
  with XorImage do
  begin
    {we are creating a black and white cursor}
    Monochrome := TRUE;

    {set the dimensions to those reported by the system}
    Width := GetSystemMetrics(SM_CXICON);
    Height := GetSystemMetrics(SM_CYICON);

    {fill the bitmap with black}
    Canvas.Brush.Color := clBlack;
    Canvas.Pen.Color := clBlack;
    Canvas.FillRect(Canvas.ClipRect);
  end;

  {allocate the memory for the bit arrays}
  GetMem(AndMask,MaskSize);
  GetMem(XorMask,MaskSize);

  {transfer the images in the bitmaps to the bit arrays}
  GetBitmapBits(AndImage.Handle, MaskSize, AndMask);
  GetBitmapBits(XorImage.Handle, MaskSize, XorMask);

  {create a new cursor based on the images transferred into
   the bit arrays}
  NewCursor := CreateCursor(hInstance, 0, 0, GetSystemMetrics(SM_CXICON),
                            GetSystemMetrics(SM_CYICON), AndMask, XorMask);

  {if the cursor for the window class is not set to zero, SetCursor will succeed but the
   cursor will be reset to the class cursor as soon as the mouse is moved. therefore, we
   must set the class cursor for the button and the form to zero}
  SetClassLong(Form1.Handle, GCL_HCURSOR, 0);
  SetClassLong(Button1.Handle, GCL_HCURSOR, 0);

  {now that the class cursor has been deleted, set the new cursor shape}
  SetCursor(NewCursor);

  {the temporary bitmaps are no longer needed, so dispose of them}
  AndImage.Free;
  XorImage.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  {retrieve and save a handle to the original cursor}
  OldCursor := GetCursor;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  {set the cursor back to the original cursor shape}
  SetCursor(OldCursor);

  {delete the new cursor}
  DestroyCursor(NewCursor);
end;
```

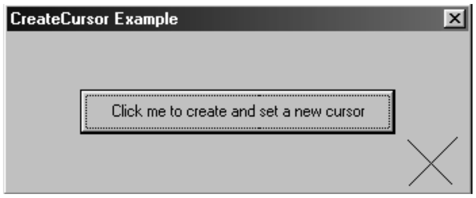


Figure 9-8: The new cursor

CreateIcon Windows.pas

Syntax

CreateIcon(
hInstance: HINST; {a handle to the application instance}

```
nWidth: Integer;          {the width of the icon}
nHeight: Integer;         {the height of the icon}
cPlanes: Byte;            {the number of color planes}
cBitsPixel: Byte;         {the number of bits describing an XOR mask pixel}
lpbANDbits: Pointer;      {a pointer to the AND mask data}
lpbXORbits: Pointer       {a pointer to the XOR mask data}
): HICON;                 {returns a handle to an icon}
```

Description

This function dynamically creates a new icon with the specified dimensions and image.

Parameters

hInstance: A handle to the instance of the application creating the icon.

nWidth: The width of the icon in pixels. This parameter must be set to the value returned by GetSystemMetrics(SM_CXICON).

nHeight: The height of the icon in pixels. This parameter must be set to the value returned by GetSystemMetrics(SM_CYICON).

cPlanes: The number of color planes used in the XOR icon mask.

cBitsPixel: The number of bits required to describe the color of one pixel (i.e., 8 bits for 256 color images, 24 bits for 16.7 million color images, etc.).

lpbANDbits: A pointer to an array of bytes containing the image for the AND mask of the icon. The image information contained in this array must describe a monochrome bitmap.

lpbXORbits: A pointer to an array of bytes containing the image for the XOR mask of the icon. This image information can describe either a monochrome bitmap or a device dependent color bitmap.

Return Value

If the function succeeds, it returns the handle to a new icon; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

CreateIconFromResource, CreateIconFromResourceEx, CreateIconIndirect, DrawIcon, DrawIconEx, GetSystemMetrics*, LoadIcon

Example

Listing 9-6: Creating an icon at run time

```
procedure TForm1.Button1Click(Sender: TObject);
var
  AndMaskSize,          // holds the computed size of the Icon
  XorMaskSize: Integer;

  AndMask,              // Icon bit arrays
  XorMask: ^Byte;
  AndImage,             // intermediate bitmaps used to define the Icon shape
  XorImage: TBitmap;
begin
  {compute the size of the Icon bit arrays}
  XorMaskSize := GetSystemMetrics(SM_CXICON)*GetSystemMetrics(SM_CYICON);
  {the AND mask is a monochrome bitmap. thus, each bit represents a
   pixel, so divide the width by 8 to get the correct number of bytes}
  AndMaskSize :=(GetSystemMetrics(SM_CXICON) div 8)*GetSystemMetrics(SM_CYICON);

  {create the bitmap used to define the XOR mask shape}
  XorImage := TBitmap.Create;
  XorImage.PixelFormat := pf8bit;
  with XorImage do
  begin
    {set the dimensions to those reported by the system}
    Width := GetSystemMetrics(SM_CXICON);
    Height := GetSystemMetrics(SM_CYICON);

    {fill the background with black}
    Canvas.Brush.Color := clBlack;
    Canvas.FillRect(Canvas.ClipRect);

    {draw a red box}
    Canvas.Brush.Color := clRed;
    Canvas.Pen.Color := clBlack;
    Canvas.FillRect(Rect(5,5,GetSystemMetrics(SM_CXICON)-5,
                         GetSystemMetrics(SM_CYICON)-5));
  end;

  {create the bitmap used to define the AND mask shape}
  AndImage := TBitmap.Create;
  with AndImage do
  begin
    {the AND mask is always black and white}
    Monochrome := TRUE;

    {set the dimensions to those reported by the system}
    Width := GetSystemMetrics(SM_CXICON);
    Height := GetSystemMetrics(SM_CYICON);

    {fill the background with white}
    Canvas.Brush.Color := clWhite;
    Canvas.FillRect(Canvas.ClipRect);

    {draw a black box the same size as the red box in the XOR bitmask}
    Canvas.Brush.Color := clBlack;
    Canvas.Pen.Color := clBlack;
    Canvas.FillRect(Rect(5,5,GetSystemMetrics(SM_CXICON)-5,
                         GetSystemMetrics(SM_CYICON)-5));
  end;

  {allocate the memory for the bit arrays}
  GetMem(AndMask,AndMaskSize);
  GetMem(XorMask,XorMaskSize);
  {transfer the images in the bitmaps to the bit arrays}
  GetBitmapBits(AndImage.Handle, AndMaskSize, AndMask);
  GetBitmapBits(XorImage.Handle, XorMaskSize, XorMask);

  {create a new Icon based on the images transferred into the bit arrays}
  NewIcon := CreateIcon(hInstance, GetSystemMetrics(SM_CXICON),
                        GetSystemMetrics(SM_CYICON), 1, 8, AndMask, XorMask);

  {point the application's icon to this new icon}
  Application.Icon.Handle := NewIcon;

  {display the icon on the form}
  DrawIcon(PaintBox1.Canvas.Handle,
           PaintBox1.Width div 2-(GetSystemMetrics(SM_CXICON) div 2),
           PaintBox1.Height div 2-(GetSystemMetrics(SM_CYICON) div 2), NewIcon);

  {the temporary bitmaps are no longer needed, so dispose of them}
  AndImage.Free;
  XorImage.Free;
end;
```

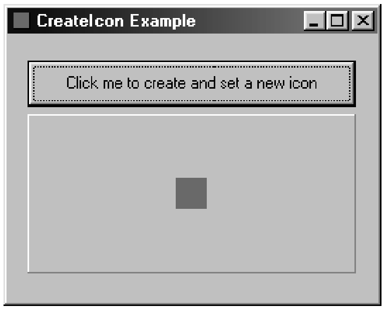


Figure 9-9: The new icon

CreateIconFromResource Windows.pas

Syntax

CreateIconFromResource(	
presbits: PByte;	{a pointer to icon or cursor bits}
dwResSize: DWORD;	{the number of bytes pointed to by presbits}
fIcon: BOOL;	{indicates an icon or cursor}
dwVer: DWORD	{the format version number}
): HICON;	{returns a handle to an icon or cursor}

Description

This function creates a new icon or cursor from the specified resource bits defining the icon or cursor image.

Parameters

presbits: A pointer to a buffer containing the icon or cursor resource bits. The return value from the LoadResource or LookupIconIdFromDirectory functions can be used as the input for this parameter.

dwResSize: The size of the buffer pointed to by the presbits parameter, in bytes.

fIcon: A flag indicating whether an icon or cursor is created. A value of TRUE causes the function to create an icon, and FALSE creates a cursor.

dwVer: Specifies the icon and cursor format version number. Win32 applications should set this value to $30000.

Return Value

If the function succeeds, it returns a handle to an icon or cursor; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

CreateIcon, CreateIconFromResourceEx, CreateIconIndirect, LookupIconId-FromDirectory, LookupIconIdFromDirectoryEx

Example

Listing 9-7: Creating an icon from resource information

```
procedure TForm1.Button1Click(Sender: TObject);
var
  IconBits: HGLOBAL;        // a handle to the icon image
  IconBitsPtr: Pointer;     // a pointer to the icon image
  ResHandle: HRSRC;         // a handle to the icon resource information
  ResId: Integer;           // holds the resource id for the icon
  TheIcon: HICON;           // a handle to the created icon
begin
  {retrieve a handle to the icon resource}
  ResHandle := FindResource(0, 'TARGET', RT_GROUP_ICON);

  {retrieve a handle to the icon resource image}
  IconBits := LoadResource(0, ResHandle);

  {retrieve a pointer to the icon image}
  IconBitsPtr := LockResource(IconBits);

  {find the icon that fits the current display device}
  ResId := LookupIconIDFromDirectory(IconBitsPtr, TRUE);

  {retrieve a handle to this icon}
  ResHandle := FindResource(0, MakeIntResource(ResId), RT_ICON);

  {load the icon resource image}
  IconBits := LoadResource(0, ResHandle);

  {retrieve a pointer to the icon image}
  IconBitsPtr := LockResource(IconBits);

  {create a new icon from the correct icon resource information}
  TheIcon := CreateIconFromResource(IconBitsPtr, SizeOfResource(0, ResHandle),
                                    TRUE, $30000);

  {display the icon}
  DrawIcon(PaintBox1.Canvas.Handle,
           PaintBox1.Width div 2-(GetSystemMetrics(SM_CXICON) div 2),
           PaintBox1.Height div 2-(GetSystemMetrics(SM_CYICON) div 2), TheIcon);
end;
```

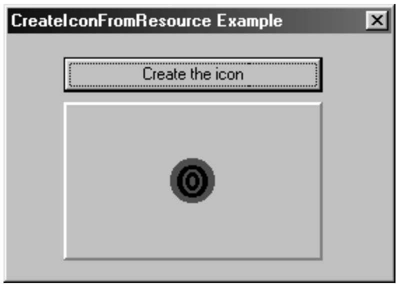


Figure 9-10: The new icon

CreateIconFromResourceEx   Windows.pas

Syntax

CreateIconFromResourceEx(	
presbits: PByte;	{a pointer to icon or cursor bits}
dwResSize: DWORD;	{the number of bytes pointed to by presbits}
fIcon: BOOL;	{indicates an icon or cursor}
dwVer: DWORD;	{the format version number}
cxDesired: Integer;	{the preferred width of the icon or cursor}
cyDesired: Integer;	{the preferred height of the icon or cursor}
Flags: UINT	{color flags}
): HICON;	{returns a handle to an icon or cursor}

Description

This function creates a new icon or cursor from the specified resource bits defining the icon or cursor image. Unlike the CreateIconFromResource function, this function allows the developer to determine the dimensions and color format of the icon or cursor.

Parameters

presbits: A pointer to a buffer containing the icon or cursor resource bits. The return value from the LoadResource or LookupIconIdFromDirectory functions can be used as the input for this parameter.

dwResSize: The size of the buffer pointed to by the presbits parameter, in bytes.

fIcon: A flag indicating whether an icon or cursor is created. A value of TRUE causes the function to create an icon, and FALSE creates a cursor.

dwVer: Specifies the icon and cursor format version number. Win32 applications should set this value to $30000.

cxDesired: Specifies the preferred width of the icon or cursor in pixels. If this parameter is zero, the function uses the value returned from GetSystemMetrics(SM_CX-ICON).

cyDesired: Specifies the preferred height of the icon or cursor in pixels. If this parameter is zero, the function uses the value returned from GetSystemMetrics(SM_CYICON).

Flags: A value indicating the color format for the icon or cursor. This parameter can be one value from the following table.

Return Value

If the function succeeds, it returns a handle to an icon or cursor; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

CreateIcon, CreateIconFromResource, CreateIconIndirect, LookupIconIdFromDirectory, LookupIconIdFromDirectoryEx

Example

Listing 9-8: More options for creating an icon from resource information

```
procedure TForm1.Button1Click(Sender: TObject);
var
  IconBits: HGLOBAL;        // a handle to the icon image
  IconBitsPtr: Pointer;     // a pointer to the icon image
  ResHandle: HRSRC;         // a handle to the icon resource information
  ResId: Integer;           // holds the resource id for the icon
  TheIcon: HICON;           // a handle to the created icon
begin
  {retrieve a handle to the icon resource}
  ResHandle := FindResource(0, 'TARGET', RT_GROUP_ICON);

  {retrieve a handle to the icon resource image}
  IconBits := LoadResource(0, ResHandle);

  {retrieve a pointer to the icon image}
  IconBitsPtr := LockResource(IconBits);

  {find the icon that fits the current display device}
  ResId := LookupIconIDFromDirectoryEx(IconBitsPtr, TRUE,
                                       0, 0, LR_DEFAULTCOLOR);

  {retrieve a handle to this icon}
  ResHandle := FindResource(0, MakeIntResource(ResId), RT_ICON);

  {load the icon resource image}
  IconBits := LoadResource(0, ResHandle);

  {retrieve a pointer to the icon image}
  IconBitsPtr := LockResource(IconBits);

  {create a new icon from the correct icon resource information}
  TheIcon := CreateIconFromResourceEx(IconBitsPtr, SizeOfResource(0, ResHandle),
                                      TRUE, $30000, 0, 0, LR_DEFAULTCOLOR);
  {display the icon}
  DrawIcon(PaintBox1.Canvas.Handle,
           PaintBox1.Width div 2-(GetSystemMetrics(SM_CXICON) div 2),
           PaintBox1.Height div 2-(GetSystemMetrics(SM_CYICON) div 2), TheIcon);
end;
```

Table 9-2: CreateIconFromResourceEx Flags values

Value	Description
LR_DEFAULTCOLOR	Create a color cursor or icon using the default system colors.
LR_MONOCHROME	Create a monochrome cursor or icon.

CreateIconIndirect Windows.pas

Syntax

```
CreateIconIndirect(
var piconinfo: TIconInfo      {a pointer to an icon information data structure}
): HICON;                     {returns a handle to an icon}
```

Description

This function dynamically creates a new icon from the dimensions and images defined in the piconinfo variable. After the icon is created, the application must manage the bitmaps used in the icon definition and delete them when they are no longer used. Icons created with this function must be destroyed by using the DestroyIcon function.

Parameters

piconinfo: A pointer to a TIconInfo data structure that describes the icon image. The TIconInfo structure is defined as:

```
TIconInfo = packed record
      fIcon: BOOL;              {indicates icon or cursor information}
      xHotspot: DWORD;          {the hot spot horizontal coordinate}
      yHotspot: DWORD;          {the hot spot vertical coordinate}
      hbmMask: HBITMAP;         {a bitmap handle}
      hbmColor: HBITMAP;        {a bitmap handle}
end;
```

See GetIconInfo for a description of the data structure members.

Return Value

If the function succeeds, it returns a handle to the new icon; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

CreateIcon, CreateIconFromResource, CreateIconFromResourceEx, DestroyIcon, DrawIcon, DrawIconEx, LoadIcon

Example

Listing 9-9: Creating an icon indirectly

```
var
  Form1: TForm1;
  NewIcon: HICON;  // holds the new icon

implementation

procedure TForm1.Button1Click(Sender: TObject);
var
  AndImage,             // bitmaps used to define the Icon shape
  XorImage: TBitmap;
  IconInfo: TIconInfo;  // the icon information data structure
begin
  {create the bitmap used to define the XOR mask shape}
  XorImage := TBitmap.Create;
  with XorImage do
  begin
    {set the dimensions to those reported by the system}
    Width := GetSystemMetrics(SM_CXICON);
    Height := GetSystemMetrics(SM_CYICON);

    {fill the background with black}
    Canvas.Brush.Color := clBlack;
    Canvas.FillRect(Canvas.ClipRect);

    {draw a red box}
    Canvas.Brush.Color := clRed;
    Canvas.Pen.Color := clBlack;
    Canvas.FillRect(Rect(5,5,GetSystemMetrics(SM_CXICON)-5,
                         GetSystemMetrics(SM_CYICON)-5));
  end;

  {create the bitmap used to define the AND mask shape}
  AndImage := TBitmap.Create;
  with AndImage do
  begin
    {the AND mask is always black and white}
    Monochrome := TRUE;

    {set the dimensions to those reported by the system}
    Width := GetSystemMetrics(SM_CXICON);
    Height := GetSystemMetrics(SM_CYICON);

    {fill the background with white}
    Canvas.Brush.Color := clWhite;
    Canvas.FillRect(Canvas.ClipRect);

    {draw a black box the same size as the red box in the XOR bitmask}
    Canvas.Brush.Color := clBlack;
    Canvas.Pen.Color := clBlack;
    Canvas.FillRect(Rect(5,5,GetSystemMetrics(SM_CXICON)-5,
                         GetSystemMetrics(SM_CYICON)-5));
  end;
  {initialize the icon information structure to define the new icon}
  IconInfo.fIcon := TRUE;
  IconInfo.xHotspot := 0;
  IconInfo.yHotspot := 0;
  IconInfo.hbmMask:=AndImage.Handle;
  IconInfo.hbmColor:=XorImage.Handle;

  {create a new Icon based on the icon information data structure}
  NewIcon := CreateIconIndirect(IconInfo);

  {point the application's icon to this new icon}
  Application.Icon.Handle := NewIcon;

  {display the icon on the form}
  DrawIcon(PaintBox1.Canvas.Handle,
          PaintBox1.Width div 2-(GetSystemMetrics(SM_CXICON) div 2),
          PaintBox1.Height div 2-(GetSystemMetrics(SM_CYICON) div 2), NewIcon);

  {the temporary bitmaps are no longer needed, so dispose of them}
  AndImage.Free;
  XorImage.Free;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  {delete the new Icon}
  DestroyIcon(NewIcon);
end;
```

DestroyCaret Windows.pas

Syntax

DestroyCaret: BOOL; {returns a TRUE or FALSE}

Description

This function deletes the caret's current shape, frees it from the window, and removes it from the screen. If a bitmap was used to define the caret's shape, the bitmap is not freed. DestroyCaret fails if the window that owns the caret is not in the current task.

Return Value

If the function succeeds, it returns TRUE; otherwise it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CreateCaret, HideCaret, ShowCaret

Example

Please see Listing 9-4 under CreateCaret.

DestroyCursor Windows.pas

Syntax

```
DestroyCursor(
hCursor: HICON                {a handle to the cursor being destroyed}
): BOOL;                      {returns TRUE or FALSE}
```

Description

This function destroys the cursor identified by the given cursor handle and frees its memory. This function should only be used to destroy cursors created with the CreateCursor function.

Parameters

hCursor: A handle to the cursor to be destroyed. This cursor handle must not be in use at the time this function is called.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CreateCursor

Example

Please see Listing 9-5 under CreateCursor.

DestroyIcon Windows.pas

Syntax

```
DestroyIcon(
hIcon: HICON            {a handle to the icon being destroyed}
): BOOL;                {returns TRUE or FALSE}
```

Description

This function destroys the icon identified by the given icon handle and frees its memory. This function should only be used to destroy icons created with the CreateIcon-Indirect function.

Parameters

hIcon: A handle to the icon to be destroyed. This icon handle must not be in use at the time this function is called.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CreateIconIndirect

Example

Please see Listing 9-9 under CreateIconIndirect.

DrawIcon   Windows.pas

Syntax

```
DrawIcon(
hDC: HDC;              {a handle to a device context}
X: Integer;            {the horizontal coordinate of the icon or cursor}
Y: Integer;            {the vertical coordinate of the icon or cursor}
hIcon: HICON           {a handle to the icon or cursor to draw}
): BOOL;               {returns TRUE or FALSE}
```

Description

This function draws an icon or cursor, including animated cursors, onto the specified device context.

Parameters

hDC: A handle to the device context upon which the icon or cursor will be drawn.

X: Indicates the horizontal position of the upper-left corner of the icon or cursor within the specified device context, subject to the current mapping mode.

Y: Indicates the vertical position of the upper-left corner of the icon or cursor within the specified device context, subject to the current mapping mode.

hIcon: A handle to the icon or cursor to be drawn.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

DrawIconEx, LoadCursor, LoadIcon, LoadImage*

Example

Please see Listing 9-6 under CreateIcon.

DrawIconEx   Windows.pas

Syntax

```
DrawIconEx(
hdc: HDC;              {a handle to a device context}
xLeft: Integer;        {the horizontal coordinate for displaying the icon}
yTop: Integer;         {the vertical coordinate for displaying the icon}
hIcon: HICON;                    {a handle to the icon to display}
cxWidth: Integer;                {the width of the icon}
cyHeight: Integer;               {the height of the icon}
istepIfAniCur: UINT;             {the frame index of an animated cursor}
hbrFlickerFreeDraw: HBRUSH;      {a handle to a brush}
diFlags: UINT                    {icon display flags}
): BOOL;                         {returns TRUE or FALSE}
```

Description

This function draws an icon or cursor, including animated cursors, onto the specified device context. The icon or cursor can be stretched or compressed as desired.

Parameters

hdc: A handle to the device context upon which the icon or cursor will be drawn.

xLeft: Indicates the horizontal position of the upper left corner of the icon or cursor within the specified device context, subject to the current mapping mode.

yTop: Indicates the vertical position of the upper left corner of the icon or cursor within the specified device context, subject to the current mapping mode.

hIcon: A handle to the icon or cursor to be drawn.

cxWidth: Specifies how wide to draw the icon or cursor, in logical units. If this parameter is zero and the diFlags parameter is set to DI_DEFAULTSIZE, the function uses the SM_CXICON or SM_CXCURSOR system metric values for the width. If this parameter is zero and the diFlags parameter is not set to DI_DEFAULTSIZE, the function uses the actual width of the icon or cursor resource.

cyHeight: Specifies how tall to draw the icon or cursor, in logical units. If this parameter is zero and the diFlags parameter is set to DI_DEFAULTSIZE, the function uses the SM_CYICON or SM_CYCURSOR system metric values for the height. If this parameter is zero and the diFlags parameter is not set to DI_DEFAULTSIZE, the function uses the actual height of the icon or cursor resource.

istepIfAniCur: Specifies which frame of an animated cursor to draw. If the hIcon parameter does not specify a handle to an animated icon, this parameter is ignored.

hbrFlickerFreeDraw: A handle to a brush. If the brush handle is valid, the function creates an offscreen bitmap using the brush for the background color, draws the icon or cursor into this offscreen bitmap, and then copies it onto the device context specified by the hdc parameter. This eliminates any flicker when displaying the icon or cursor. If this parameter is zero, the icon or cursor is drawn directly into the specified device context.

diFlags: A flag controlling drawing behavior. This parameter can be one or more values from the following table.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

DrawIcon, LoadCursor, LoadIcon, LoadImage*

Example

Please see Listing 9-14 under LoadCursorFromFile.

Table 9-3: DrawIconEx diFlags values

Value	Description
DI_COMPAT	Draws the default cursor image for the specified standard cursor, even if the cursor has been replaced by a call to the SetSystemCursor function. **Note**: This flag is ignored under Windows NT 4.0 and later.
DI_DEFAULTSIZE	Draws the icon or cursor with the system metric-defined width and height for icons and cursors.
DI_IMAGE	Draws only the OR mask of the icon or cursor.
DI_MASK	Draws only the AND mask of the icon or cursor.
DI_NOMIRROR	**Windows XP and later**: Draws the icon as an unmirrored icon.
DI_NORMAL	Combines the DI_IMAGE and DI_MASK values to draw the cursor or icon as it is normally displayed.

ExtractAssociatedIcon ShellAPI.pas

Syntax

```
ExtractAssociatedIcon(
hInst: HINST;                  {a handle to the application instance}
lpIconPath: PChar;             {a pointer to a filename string}
var lpiIcon: Word              {the icon index}
): HICON;                      {returns a handle to an icon}
```

Description

This function returns the handle to an icon extracted from the file referenced by the lpIconPath parameter. If this parameter does not point to an executable file, the icon is extracted from the executable file associated with the specified file. If this file does not have an associated executable file, this function returns the handle to a default icon assigned by Windows. In addition, if the filename specifies a bitmap or cursor file, this function will create an icon from the image and return its handle.

Parameters

hInst: A handle to the application instance.

lpIconPath: A pointer to a null-terminated string containing the filename from which to extract the icon.

lpiIcon: A pointer to a variable containing the index of the icon to extract. The index is zero-based, so a value of 0 will retrieve the first icon in the file.

Return Value

If the function succeeds, it returns the handle to an icon; otherwise, it returns zero.

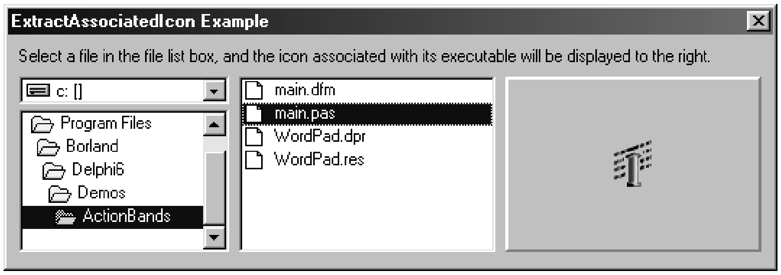
See Also

DrawIcon, DrawIconEx, ExtractIcon, LoadIcon

Example

Listing 9-10: Extracting icons associated with a file

```
procedure TForm1.FileListBox1Click(Sender: TObject);
var
   IconIndex: Word;
   TheIcon: TIcon;
begin
   {extract the first icon found}
   IconIndex:=0;

   {create the temporary icon object}
   TheIcon:=TIcon.Create;

   {extract a handle to the icon of the executable
    associated with the selected file}
   TheIcon.Handle:=ExtractAssociatedIcon(hInstance,PChar(FileListBox1.FileName),
                                         IconIndex);

   {copy the icon into the image object}
   Image1.Picture.Assign(TheIcon);

   {free the temporary icon object}
   TheIcon.Free;
end;
```


ExtractAssociatedIcon Example

Select a file in the file list box, and the icon associated with its executable will be displayed to the right.

Figure 9-11: An icon associated with a file

ExtractIcon ShellAPI.pas

Syntax

ExtractIcon(	
hInst: HINST;	{a handle to the application instance}
lpszExeFileName: PChar;	{a pointer to a filename string}
nIconIndex: UINT	{the icon index}
): HICON;	{returns a handle to an icon}

Description

This function returns a handle to an icon extracted from an executable file, DLL, or icon file. In addition, if the filename specifies a bitmap or cursor file, this function will create an icon from the image and return its handle. This function can be used to convert bitmaps or cursors into icons.

Parameters

hInst: A handle to the application instance.

lpszExeFileName: A pointer to a null-terminated string containing the filename from which to extract the icon.

nIconIndex: The index of the icon to retrieve. The index is zero-based, so a value of 0 will retrieve the first icon in the file, if any exist. If this value is –1, the return value will be the total number of icons stored in the specified file.

Return Value

If the function succeeds, it returns a handle to an icon. If the function fails, or there are no icons stored in the specified file, it returns zero.

See Also

DrawIcon, DrawIconEx, ExtractAssociatedIcon, LoadIcon

Example

Listing 9-11: Extracting the icon from a file

```
  {the ExtractIcon function is imported incorrectly in Delphi 6}
  function ExtractIcon(hInst: HINST; lpszExeFileName: PChar;
                       nIconIndex: Integer): HICON; stdcall;
var
  Form1: TForm1;

implementation

{$R *.DFM}

{import the function}
function ExtractIcon; external 'shell32.dll' name 'ExtractIconA';

procedure TForm1.FileListBox1DblClick(Sender: TObject);
var
   TheIcon: TIcon;        // this will hold the returned icon
   NumIcons: Integer;     // holds the icon count
begin
   {determine the number of icons stored in this file}
   NumIcons:=ExtractIcon(hInstance,PChar(FileListBox1.FileName), -1);

   {display this number}
   Label2.Caption:=IntToStr(NumIcons)+' icon(s) in this file';

   {create an icon object}
   TheIcon:=TIcon.Create;
   {find the icon in the selected application, if one exists}
   TheIcon.Handle:=ExtractIcon(hInstance, PChar(FileListBox1.FileName), 0);

   {display the icon. if no icon exists, the currently
    displayed icon will be cleared}
   Image1.Picture.Assign(TheIcon);

   {free our icon object}
   TheIcon.Free;
end;
```

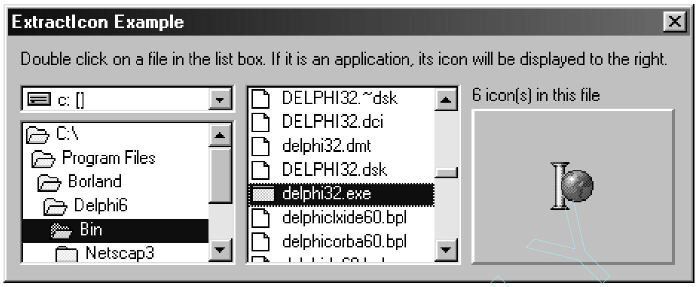


Figure 9-12: The extracted icon

ExtractIconEx ShellAPI.pas

Syntax

ExtractIconEx(	
lpszFile: PChar;	{a pointer to a filename string}
nIconIndex: Integer;	{the icon index}
var phiconLarge: HICON;	{a pointer to a large icon handle}
var phiconSmall: HICON;	{a pointer to a small icon handle}
nIcons: UINT	{the number of icons to retrieve}
): UINT;	{returns the number of icons stored in the file}

Description

This function returns the handle to both a large and small icon stored in an executable file, DLL, or icon file.

Parameters

lpszFile: A pointer to a null-terminated string containing the filename of an executable file, DLL, or icon file from which to extract the icons.

nIconIndex: The index of the icon to retrieve. The index is zero-based, so a value of 0 will retrieve the first large and small icon in the file, if any exist. If this value is –1, the return value will be the total number of icons stored in the specified file.

phiconLarge: A pointer to an icon handle. If the function succeeds, this value will point to the handle of a large icon extracted from the given file.

phiconSmall: A pointer to an icon handle. If the function succeeds, this value will point to the handle of a small icon extracted from the given file.

nIcons: A value indicating the number of icons to extract.

Return Value

If the function succeeds and the value of nIconIndex is –1, it returns the total number of icons stored in the file. This function counts a large icon and its associated small icon as one icon (i.e., if there are three large icons and three small icons in a file, this function would return a 3). If the function extracts icons, it returns the total number of large and small icons extracted (i.e., if it extracted one small and one large icon, it returns 2). If the function fails, or there are no icons stored in the indicated file, it returns 0..

See Also

DrawIcon, DrawIconEx, ExtractIcon, LoadIcon

Example

Listing 9-12: Extracting large and small icons

```
var
  Form1: TForm1;
  LargeIconsBitmap: TBitmap;  // this holds the icon images
  SmallIconsBitmap: TBitmap;

implementation

{$R *.DFM}

procedure TForm1.FileListBox1DblClick(Sender: TObject);
var
   NumIcons: Integer;         // holds the icon count
   LIcon: HICON;              // holds the handles to extracted icons
   SIcon: HICON;
   LoopCount: Integer;        // a general loop counter
begin
   {determine the number of icons stored in this file}
   NumIcons:=ExtractIconEx(PChar(FileListBox1.FileName), -1, LIcon, SIcon, 0);

   {display this number}
   Label4.Caption:='Total Number of Icons: '+IntToStr(NumIcons);

   {resize the images and clear the canvases of the offscreen bitmaps.
    we add a 1 to the width in case there are no icons. this prevents
    the Height of these objects from being reset to 1.}
   Image1.Width:=NumIcons*40+1;
   Image2.Width:=NumIcons*40+1;
   LargeIconsBitmap.Width:=NumIcons*40+1;
   LargeIconsBitmap.Canvas.FillRect(LargeIconsBitmap.Canvas.ClipRect);
   SmallIconsBitmap.Width:=NumIcons*40+1;
   SmallIconsBitmap.Canvas.FillRect(SmallIconsBitmap.Canvas.ClipRect);

   {extract each large and small icon from the file}
   for LoopCount:=0 to NumIcons-1 do
   begin
      {find the icon in the selected application, if one exists}
      ExtractIconEx(PChar(FileListBox1.FileName), LoopCount, LIcon, SIcon, 1);

      {display the large icon}
      DrawIcon(LargeIconsBitmap.Canvas.Handle, (LoopCount*40)+4, 2, LIcon);

      {draw the small icon to the correct dimensions}
      DrawIconEx(SmallIconsBitmap.Canvas.Handle, (LoopCount*40)+4, 2, SIcon,
                 GetSystemMetrics(SM_CXSMICON), GetSystemMetrics(SM_CYSMICON),
                 0, 0, DI_NORMAL);
   end;

   {assign the offscreen bitmaps to the images for display}
   Image1.Picture.Bitmap.Assign(LargeIconsBitmap);
   Image2.Picture.Bitmap.Assign(SmallIconsBitmap);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   {create the offscreen bitmaps to hold the images of the icons.}
   LargeIconsBitmap:=TBitmap.Create;
   LargeIconsBitmap.Height:=53;
   LargeIconsBitmap.Width:=40;

   SmallIconsBitmap:=TBitmap.Create;
   SmallIconsBitmap.Height:=53;
   SmallIconsBitmap.Width:=40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
   {free the offscreen bitmaps}
   LargeIconsBitmap.Free;
   SmallIconsBitmap.Free;
end;
```

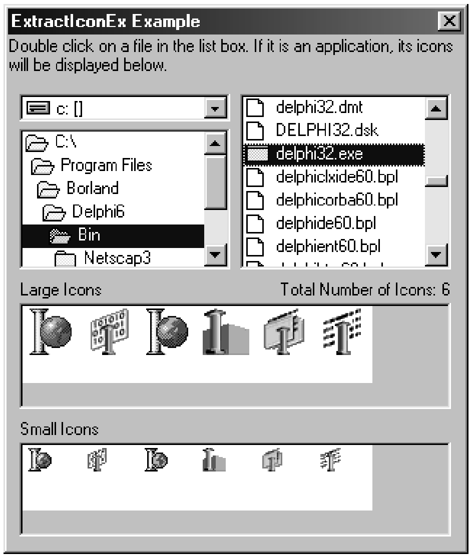


Figure 9-13: The large and small icons extracted from a file

GetCursor Windows.pas

Syntax

```
GetCursor: HCURSOR;         {returns a handle to a cursor}
```

Description

This function retrieves a handle to the current cursor.

Return Value

If the function succeeds, it returns a handle to the current cursor; otherwise, it returns zero.

See Also

CreateCursor, SetCursor

Example

Please see Listing 9-5 under CreateCursor.

GetIconInfo Windows.pas

Syntax

```
GetIconInfo(
hIcon: HICON;               {a handle to an icon or cursor}
var piconinfo: TIconInfo    {a pointer to an icon information structure}
): BOOL;                    {returns TRUE or FALSE}
```

Description

This function retrieves information about an icon or cursor, including hot spots and mask images.

Parameters

hIcon: A handle to an icon or cursor whose information is to be retrieved. To retrieve information about a standard icon or cursor, this parameter may be set to one value from Table 9-4.

piconinfo: A pointer to a TIconInfo structure. This structure is filled with the requested information on the specified icon or cursor when the function returns. The TIconInfo structure is defined as:

```
TIconInfo = packed record
      fIcon: BOOL;             {indicates icon or cursor information}
      xHotspot: DWORD;         {the hot spot horizontal coordinate}
      yHotspot: DWORD;         {the hot spot vertical coordinate}
      hbmMask: HBITMAP;        {a bitmap handle}
      hbmColor: HBITMAP;       {a bitmap handle}
end;
```

fIcon: A flag indicating if the structure contains information on an icon or a cursor. If this member contains TRUE, the structure contains information on an icon; otherwise, it contains information on a cursor.

xHotspot: Specifies the horizontal coordinate of the cursor's hot spot. If the structure contains information on an icon, the hot spot is always in the center of the icon and this member is ignored.

yHotspot: Specifies the vertical coordinate of the cursor's hot spot. If the structure contains information on an icon, the hot spot is always in the center of the icon and this member is ignored.

hbmMask: A handle to the AND mask bitmap of the icon or cursor. If the structure contains information on a black and white icon or cursor, the AND mask is formatted so that the upper half contains the AND mask and the lower half contains the OR mask. In this case, the hbmColor member may contain a zero. The application must delete this bitmap when it is no longer needed.

hbmColor: A handle to the OR mask bitmap of the icon or cursor. For animated cursors, this will be the first frame of the cursor's color images. The application must delete this bitmap when it is no longer needed.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

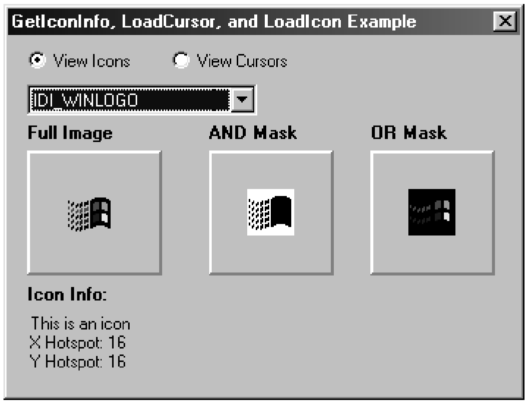
CreateIcon, CreateIconFromResource, CreateIconIndirect, DestroyIcon, DrawIcon, DrawIconEx, LoadIcon

Example

Listing 9-13: Retrieving information on system cursors and icons

```
procedure TForm1.ComboBox1Change(Sender: TObject);
var
   TheIcon: HICON;              // holds an icon
   TheCursor: HCURSOR;          // holds a cursor

   TheIconInfo: TIconInfo;      // holds the cursor or icon information

{this specifies the system icons and cursors}
type
   TIconTypes = array[0..8] of PAnsiChar;
   TCursorTypes = array[0..12] of PAnsiChar;
const
   IconTypes: TIconTypes = (IDI_APPLICATION,IDI_ASTERISK,IDI_ERROR,
                            IDI_EXCLAMATION,IDI_HAND,IDI_INFORMATION,
                            IDI_QUESTION,IDI_WARNING,IDI_WINLOGO);
   CursorTypes: TCursorTypes = (IDC_ARROW,IDC_IBEAM,IDC_WAIT,IDC_CROSS,
                                IDC_UPARROW,IDC_SIZENWSE,IDC_SIZENESW,
                                IDC_SIZEWE,IDC_SIZENS,IDC_SIZEALL,
                                IDC_NO,IDC_APPSTARTING,IDC_HELP);
begin
```

```
   {erase the last image}
   Image1.Canvas.Brush.Color:=clBtnFace;
   Image1.Canvas.Fillrect(Image1.Canvas.Cliprect);
   if TheIconInfo.hbmMask<>0 then DeleteObject(TheIconInfo.hbmMask);
   if TheIconInfo.hbmColor<>0 then DeleteObject(TheIconInfo.hbmColor);

   {if we have selected icons, get an icon...}
   if RadioButton1.Checked then
   begin
      {load the selected system icon}
      TheIcon:=LoadIcon(0, IconTypes[ComboBox1.ItemIndex]);

      {fill the information structure for this icon}
      GetIconInfo(TheIcon, TheIconInfo);

      {now draw the icon on the TImage canvas}
      DrawIconEx(Image1.Canvas.Handle,25,25,TheIcon,0,0,0,
                 Image1.Canvas.Brush.Handle,DI_DEFAULTSIZE OR DI_NORMAL);
   end
   else
   {...otherwise, get a cursor}
   begin
      {load the selected system cursor}
      TheCursor:=LoadCursor(0, CursorTypes[ComboBox2.ItemIndex]);

      {fill the information structure for this cursor}
      GetIconInfo(TheCursor, TheIconInfo);

      {now draw the cursor on the TImage canvas}
      DrawIconEx(Image1.Canvas.Handle,25,25,TheCursor,0,0,0,
                 Image1.Canvas.Brush.Handle,DI_DEFAULTSIZE OR DI_NORMAL);
   end;

   {clear the listbox}
   ListBox1.Items.Clear;

   {fill the listbox with the icon or cursor information}
   if TheIconInfo.fIcon then
      ListBox1.Items.Add('This is an icon')
   else
      ListBox1.Items.Add('This is a cursor');

   {specify hotspots}
   ListBox1.Items.Add('X Hotspot: '+IntToStr(TheIconInfo.xHotspot));
   ListBox1.Items.Add('Y Hotspot: '+IntToStr(TheIconInfo.yHotspot));

   {display the AND and OR masks for this cursor or icon}
   Image2.Picture.Bitmap.Handle:=TheIconInfo.hbmMask;
   Image3.Picture.Bitmap.Handle:=TheIconInfo.hbmColor;
end;
```



This is an icon
X Hotspot: 16
Y Hotspot: 16

Figure 9-14: The icon and cursor information

Table 9-4: GetIconInfo hIcon values

Value	Description
IDI_APPLICATION	The default application icon.
IDI_ASTERISK	The information system icon.
IDI_ERROR	The stop system icon.
IDI_EXCLAMATION	The exclamation point system icon.
IDI_HAND	Same as the IDI_ERROR value.
IDI_INFORMATION	Same as the IDI_ASTERISK value.
IDI_QUESTION	The question mark system icon.
IDI_WARNING	Same as the IDI_EXCLAMATION value.
IDI_WINLOGO	The Windows logo icon.
IDC_ARROW	The standard arrow cursor.
IDC_IBEAM	The I beam cursor.
IDC_WAIT	The hourglass cursor.
IDC_CROSS	The crosshair cursor.
IDC_UPARROW	The vertical arrow cursor.
IDC_SIZENWSE	A diagonal arrow cursor pointing northwest and southeast.
IDC_SIZENESW	A diagonal arrow cursor pointing northeast and southwest.
IDC_SIZEWE	An arrow cursor pointing east and west.
IDC_SIZENS	An arrow cursor pointing north and south.
IDC_SIZEALL	A four-pointed arrow cursor pointing north, south, east, and west.
IDC_NO	The slashed circle cursor.
IDC_APPSTARTING	The standard arrow cursor with a small hourglass.
IDC_HELP	The standard arrow cursor with a question mark.
IDC_HAND	The hand cursor.

HideCaret Windows.pas

Syntax

```
HideCaret(
hWnd: HWND            {a handle to the window that owns the caret}
): BOOL;              {returns TRUE or FALSE}
```

Description

This function hides the caret from the screen, but it does not destroy it or lose the insertion point. Hiding is cumulative. Each time the HideCaret function is called, a subsequent ShowCaret function must be called to display the caret.

Parameters

hWnd: A handle to the window that owns the caret. If this parameter is set to FALSE, the function searches all windows in the current task. If no window in the current task owns the caret, the HideCaret function fails.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CreateCaret, DestroyCaret, ShowCaret

Example

Please see Listing 9-4 under CreateCaret.

LoadCursor Windows.pas

Syntax

```
LoadCursor(
hInstance: HINST;           {an instance handle}
lpCursorName: PAnsiChar     {the cursor resource name or identifier}
): HCURSOR;                 {returns a handle to a cursor}
```

Description

This function retrieves a handle to a cursor from cursor resources stored in the executable file associated with the given instance handle. If the cursor is not currently loaded, it will load the specified cursor resource and return its handle; otherwise, it returns a handle to the existing cursor.

Parameters

hInstance: A handle to the module instance whose executable file contains the cursor resource to be loaded.

lpCursorName: A pointer to a null-terminated string containing the name of the cursor resource to load. The MakeIntResource function can be used with a resource identifier to provide a value for this parameter. To load one of the user-defined cursors, set the hInstance parameter to zero and set this parameter to one of the values from the following table. The user-defined cursors are set from the Mouse applet under the Control Panel and are stored in the registry under the key HKEY_CURRENT_USER\Control Panel\Cursors.

Return Value

If the function succeeds, it returns a handle to a cursor loaded from the executable file resources; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

GetCursor, GetIconInfo, LoadImage*, SetCursor, ShowCursor

Example

Please see Listing 9-13 under GetIconInfo.

Table 9-5: LoadCursor lpCursorName values

Value	Description
IDC_ARROW	The standard arrow cursor.
IDC_IBEAM	The I beam cursor.
IDC_WAIT	The hourglass cursor.
IDC_CROSS	The crosshair cursor.
IDC_UPARROW	The vertical arrow cursor.
IDC_SIZENWSE	A diagonal arrow cursor pointing northwest and southeast.
IDC_SIZENESW	A diagonal arrow cursor pointing northeast and southwest.
IDC_SIZEWE	An arrow cursor pointing east and west.
IDC_SIZENS	An arrow cursor pointing north and south.
IDC_SIZEALL	A four-pointed arrow cursor pointing north, south, east, and west.
IDC_NO	The slashed circle cursor.
IDC_APPSTARTING	The standard arrow cursor with a small hourglass.
IDC_HELP	The standard arrow cursor with a question mark.
IDC_HAND	The hand cursor.

LoadCursorFromFile Windows.pas

Syntax

```
LoadCursorFromFile(
lpFileName: PAnsiChar        {a cursor filename}
): HCURSOR;                  {returns a handle to a cursor}
```

Description

This function creates a cursor based on the cursor data stored in the specified file, returning a handle to the new cursor. The cursor file can be a normal cursor file (*.cur), or it can contain animated cursor data (*.ani).

Parameters

lpFileName: A null-terminated string identifying the cursor file used to create the cursor.

Return Value

If the function succeeds, it returns a handle to the new cursor; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

LoadCursor, SetCursor, SetSystemCursor

Example

Listing 9-14: Loading a cursor from a file

```
const
  DI_NOMIRROR = $0010;    // this constant is not defined by Delphi

procedure TForm1.FileListBox1Click(Sender: TObject);
var
   TheCursor: HCURSOR;   // a handle to a cursor loaded from a file
begin
   {erase the last image}
   Image1.Canvas.Brush.Color:=clBtnFace;
   Image1.Canvas.Fillrect(Image1.Canvas.Cliprect);

   {load the cursor from the selected file}
   TheCursor:=LoadCursorFromFile(PChar(FileListBox1.FileName));

   {now draw the cursor on the TImage canvas}
   DrawIconEx(Image1.Canvas.Handle,35,35,TheCursor,0,0,0,0,DI_DEFAULTSIZE OR
              DI_NORMAL);

   {we no longer need the cursor, so delete it}
   DeleteObject(TheCursor);
end;
```

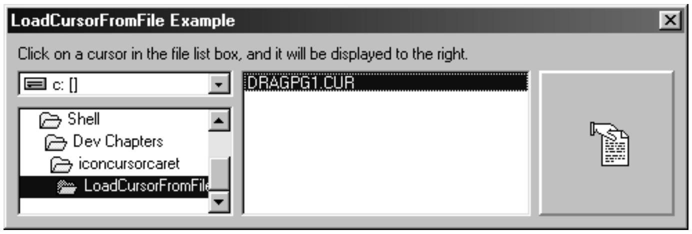

LoadCursorFromFile Example
Click on a cursor in the file list box, and it will be displayed to the right.
c: []
Shell
Dev Chapters
iconcursorcaret
LoadCursorFromFil
DRAGPG1.CUR

Figure 9-15: The loaded cursor

LoadIcon Windows.pas

Syntax

```
LoadIcon(
hInstance: HINST;       {an instance handle}
lpIconName: PChar       {an icon resource name}
): HICON;               {returns a handle to an icon}
```

Description

This function retrieves a handle to an icon from icon resources stored in the executable file associated with the given instance handle. If the icon is not currently loaded, it will load the specified icon resource and return its handle; otherwise, it returns a handle to the existing icon. The icon must have the same dimensions as those reported by the SM_CXICON and SM_CYICON system metric values. Use the LoadImage function to load icons of other sizes.

Parameters

hInstance: A handle to the module instance whose executable file contains the icon resource to be loaded.

lpIconName: A pointer to a null-terminated string containing the name of the icon resource to load. The MakeIntResource function can be used with a resource identifier to provide a value for this parameter. To load one of the predefined icons used by the Win32 API, set the hInstance parameter to zero and set this parameter to one of the values from the following table.

Return Value

If the function succeeds, it returns a handle to an icon loaded from the executable file resources; otherwise, it returns zero. To get extended error information, call the GetLastError function.

See Also

CreateIcon, LoadImage*

Example

Please see Listing 9-13 under GetIconInfo.

Table 9-6: LoadIcon lpIconName values

Value	Description
IDI_APPLICATION	The default application icon.
IDI_ASTERISK	The information system icon.
IDI_ERROR	The stop system icon.
IDI_EXCLAMATION	The exclamation point system icon.
IDI_HAND	Same as the IDI_ERROR value.
IDI_INFORMATION	Same as the IDI_ASTERISK value.
IDI_QUESTION	The question mark system icon.

Value	Description
IDI_WARNING	Same as the IDI_EXCLAMATION value.
IDI_WINLOGO	The Windows logo icon.

LookupIconIdFromDirectory ***Windows.pas***

Syntax

```
LookupIconIdFromDirectory(
presbits: PByte;            {a pointer to icon or cursor resource bits}
fIcon: BOOL                 {indicates an icon or cursor}
): Integer;                 {returns an integer resource identifier}
```

Description

This function searches through icon or cursor resource information to find the icon or cursor that is the most appropriate for the current display device. It is intended for use with resource files containing icon and cursor images in several device-dependent and device-independent formats. The return value from this function can be used with the MakeIntResource and FindResource functions to locate the cursor or icon in the module's resources.

Parameters

presbits: A pointer to icon or cursor resource bits. Use the return value from the LockResource function for this parameter.

fIcon: A flag indicating whether an icon or cursor is desired. A value of TRUE indicates an icon should be found, and FALSE indicates a cursor.

Return Value

If the function succeeds, it returns an integer resource identifier for the most appropriate icon or cursor for the current display device; otherwise, it returns zero.

See Also

CreateIconFromResource, LoadCursor, LookupIconIdFromDirectoryEx

Example

Please see Listing 9-7 under CreateIconFromResource.

LookupIconIdFromDirectoryEx ***Windows.pas***

Syntax

```
LookupIconIdFromDirectoryEx(
presbits: PByte;            {a pointer to icon or cursor resource bits}
fIcon: BOOL                 {indicates an icon or cursor}
cxDesired: Integer;         {the preferred width of the icon or cursor}
cyDesired: Integer;         {the preferred height of the icon or cursor}
Flags: UINT                 {color flags}
): Integer;                 {returns an integer resource identifier}
```

Description

This function searches through icon or cursor resource information to find the icon or cursor that is the most appropriate for the current display device. It is intended for use with resource files containing icon and cursor images in several device-dependent and device-independent formats. The return value from this function can be used with the MakeIntResource and FindResource functions to locate the cursor or icon in the module's resources. Unlike the LookupIconIdFromDirectory function, this function allows the developer to specify the dimensions and color format of the icon or cursor.

Parameters

presbits: A pointer to icon or cursor resource bits. Use the return value from the LockResource function for this parameter.

fIcon: A flag indicating whether an icon or cursor is desired. A value of TRUE indicates an icon should be found, and FALSE indicates a cursor.

cxDesired: Specifies the preferred width of the icon or cursor in pixels. If this parameter is zero, the function uses the value returned from GetSystemMetrics(SM_CXICON).

cyDesired: Specifies the preferred height of the icon or cursor in pixels. If this parameter is zero, the function uses the value returned from GetSystemMetrics(SM_CYICON).

Flags: A value indicating the color format for the icon or cursor. This parameter can be one value from the following table.

Return Value

If the function succeeds, it returns an integer resource identifier for the most appropriate icon or cursor for the current display device; otherwise, it returns zero.

See Also

CreateIconFromResourceEx, LoadCursor, LookupIconIdFromDirectory

Example

Please see Listing 9-8 under CreateIconFromResourceEx.

Table 9-7: LookupIdFromDirectoryEx Flags values

Value	Description
LR_DEFAULTCOLOR	Create a color cursor or icon using the default system colors.
LR_MONOCHROME	Create a monochrome cursor or icon.

SetCursor Windows.pas

Syntax

```
SetCursor(
hCursor: HICON          {a handle to a cursor}
): HCURSOR;             {returns a handle to the previous cursor}
```

Description

This function sets the shape of the mouse cursor to the cursor associated with the specified cursor handle. A new cursor is set only if the new cursor is different from the current cursor. When using the SetCursor function to change the cursor, the class cursor for the application's window (and child windows) must be set to zero. If the class cursor of a window is not set to zero, Windows restores the class cursor shape every time the mouse is moved over that particular window. Use the ShowCursor function to increase the internal display count to display the new cursor.

Parameters

hCursor: A handle to the cursor replacing the current mouse cursor shape. This cursor handle must be retrieved from the CreateCursor, LoadCursor, or LoadImage functions. In addition, the width and height of the cursor must match those returned by the GetSystemMetrics function, and the color depth must be equal or less than the color depth of the current display.

Return Value

If the function succeeds, it returns a handle to the previous cursor, if one existed; otherwise, it returns zero.

See Also

CreateCursor, GetCursor, ShowCursor

Example

Please see Listing 9-5 under CreateCursor.

SetSystemCursor Windows.pas

Syntax

```
SetSystemCursor(
hcur: HICON;            {a handle to the new cursor}
id: DWORD               {a system cursor identifier}
): BOOL;                {returns TRUE or FALSE}
```

Description

This function replaces the image of the specified system cursor with the image of the cursor identified by the hcur parameter. The Windows registry is not updated with this new cursor selection, and the original system cursor is reset when Windows is rebooted.

Parameters

hcur: A handle to the cursor replacing the specified system cursor.

id: A system cursor identifier. This system cursor image is replaced by the cursor indicated by the hcur parameter. This parameter can be one value from the following table.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CreateCursor, GetCursor, LoadCursor, LoadCursorFromFile, SetCursor

Example

Listing 9-15: Setting a new system cursor

```
var
  Form1: TForm1;
  CurSysCursor: HCURSOR;   // holds the current system cursor

procedure TForm1.Button1Click(Sender: TObject);
begin
  {save a handle to the current system cursor}
  CurSysCursor := GetCursor;

  {set a new system cursor}
  SetSystemCursor(Screen.Cursors[crHandPoint],OCR_NORMAL);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  {restore the previous system cursor}
  SetSystemCursor(CurSysCursor, OCR_NORMAL);
end;
```

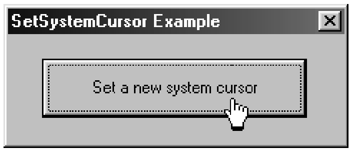


Figure 9-16: The new system cursor

Table 9-8: SetSystemCursor id values

Value	Description
OCR_APPSTARTING	The small hourglass with arrow cursor.
OCR_CROSS	The crosshair cursor.
OCR_IBEAM	The text insert cursor.
OCR_NO	The international no symbol cursor.
OCR_NORMAL	The normal arrow cursor.
OCR_SIZEALL	The all directions sizing arrow cursor.

Value	Description
OCR_SIZENESW	The northeast to southwest sizing arrow cursor.
OCR_SIZENS	The vertical sizing arrow cursor.
OCR_SIZENWSE	The northwest to southeast sizing arrow cursor.
OCR_SIZEWE	The horizontal sizing arrow cursor.
OCR_UP	The up arrow cursor.
OCR_WAIT	The hourglass cursor.
OCR_HAND	The hand cursor.

ShowCaret Windows.pas

Syntax

```
ShowCaret(
hWnd: HWND     {a handle to a window}
): BOOL;       {returns TRUE or FALSE}
```

Description

This function displays the caret on the screen at the current insertion point. The caret appears only if the specified window owns it, it has a shape, and HideCaret has not been called two or more times sequentially. Hiding is cumulative. For each time the HideCaret function is called, a subsequent ShowCaret function must be called to display the caret.

Parameters

hWnd: A handle to the window that owns the caret. If this parameter is set to FALSE, the function searches all windows in the current task. If no window in the current task owns the caret, the ShowCaret function fails.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE. To get extended error information, call the GetLastError function.

See Also

CreateCaret, DestroyCaret, HideCaret

Example

Please see Listing 9-4 under CreateCaret.

ShowCursor Windows.pas

Syntax

```
ShowCursor(
bShow: BOOL          {the cursor visibility flag}
): Integer;          {returns the cursor display counter}
```

Description

This function displays or hides the cursor by incrementing or decrementing an internal display counter. When the internal display counter goes below 0, the mouse cursor is hidden. If a mouse is installed, the initial display counter is set to 0; otherwise, it is set to –1.

Parameters

bShow: A Boolean value indicating whether the internal cursor display counter should be incremented or decremented. A value of TRUE increments the display counter, and FALSE decrements it.

Return Value

If the function succeeds, it returns the new internal display counter. If the function fails, it returns zero. A return value of zero does not necessarily mean the function failed; a comparison with previous return values should be made to determine failure in this case.

See Also

GetCursor, GetCursorPos, SetCursor, SetCursorPos

Example

Listing 9-16: Hiding and displaying the cursor

```
var
    RefCount: Integer = 0;   // holds the count of successive show/hides

procedure TForm1.Button1Click(Sender: TObject);
begin
  {if the Show button was pressed...}
  if TButton(Sender).Caption='Show' then
  begin
    {...show the cursor and increase the reference count}
    ShowCursor(TRUE);
    Inc(RefCount);
  end
  else
  {if the Hide button was pressed...}
  begin
    {...hide the cursor and decrease the reference count}
    ShowCursor(FALSE);
    Inc(RefCount,-1);
  end;

  {display the current reference count}
  Edit1.Text := IntToStr(RefCount);
end;
```

Chapter 10

Help Functions

All applications, from the simplest utility to the most massively complex suite of interconnected products, should offer online help to users. This online help typically takes the form of a dialog box that displays various tables of contents, search tools, and topics containing all manner of information about the associated application that a user would need to know. No matter how intuitive or simple an application may be, there will always be some user somewhere who will need clarification on various functions or concepts, and an online help system is the best way to get the user the information needed.

Help information can be presented in any number of ways. Some systems provide help via the Internet, opening a web browser to display pages from a specific web site. Other systems use animated "helpers" that provide an interactive feel. However, the two most commonly used systems for providing help rely on displaying information from a help file that ships with the application. These two systems, WinHelp and HTMLHelp, work very similarly. While they both present informational topics to the user in a dialog box, the capabilities of each system are somewhat different, and the format of the help files themselves are vastly dissimilar. In this chapter, we'll examine both WinHelp and HTMLHelp and compare their similarities and differences.

WinHelp

WinHelp was the standard for quite some time. It has been in use since Windows 3.0, and many applications still use this format today (Delphi 6, for example). WinHelp provides many useful features, such as a table of contents, index, and keyword search, all of which make it easy for users to find the information they seek. The WinHelp help file format provides graphics support and rich text formatting, allowing help file authors the ability to make visually pleasing and engaging topics. These features and more are the reason why the WinHelp format was the standard for so long, and why it sees continued use today.

WinHelp Architecture

When an application calls the WinHelp API function, Windows runs the WinHelp.exe executable, which is responsible for opening the WinHelp help file and displaying the specified topic. The WinHelp.exe executable is a standard part of the Windows operating system, so users will not need to download anything extra to display WinHelp help files, nor will it be necessary for applications to install any extra redistributable files. However, all WinHelp API function calls run through this one process. The result is that WinHelp displays dialog boxes only from the last calling application; it is not possible for two or more applications to have multiple help windows open at the same time.

Creating a WinHelp Help File

WinHelp help files are authored in rich text format. Therefore, just about any editor capable of saving a document in RTF format can be used to create a help file. The RTF format allows authors to use various styles of text formatting, such as bolding, italics, different font sizes, and various font colors, as well as embed graphics with text. However, WinHelp requires the use of different types of footnotes, paragraph styles, and underlining to achieve various effects within the help file, such as hypertext links, keyword searches, etc. All of these requirements are somewhat esoteric and can make the editing and creation of help files a somewhat daunting task.

Note: Covering the actual authoring of help files is beyond the scope of this book.

Once the help file is created in RTF format, it must be compiled into the proprietary format that is used by the WinHelp executable. Microsoft Help Workshop version 4.03 is the official compiler and is compatible with RTF files created with Word 97. This compiler is available as a free download from http://www.microsoft.com/downloads/release.asp?ReleaseID=23216&area=search&ordinal=1. Note that while this URL was current at the time of publicaiton, Microsoft routinely changes their web site and this URL may change all too rapidly. In case this URL has become out of date, it may be possible to find the Microsoft Help Workshop (hcwsetup.exe) by going to the Microsoft download center (http://www.microsoft.com/downloads/search.asp) and performing a keyword search on “help workshop” for the Windows 95 operating system. Fortunately, there are several commercial help file compilers still available, such as RoboHelp, that are both much more functional and provide many more features than the free help file compiler available from Microsoft.

Using WinHelp

Calling WinHelp to display a topic is relatively straightforward and can be accomplished in only one line of code. The following example demonstrates how simple it is to display a topic in a WinHelp help file.

Listing 10-1: Displaying a WinHelp topic

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  {display a topic}
  WinHelp(Handle, PChar(ExtractFilePath(Application.ExeName) +
          '\Winhelpxample.hlp'), HELP_CONTEXT, 1);
end;
```

The WinHelp API function takes only four parameters: a handle to the window calling the function, the path and filename of the help file, a help command, and a value that is dependent on the help command. The HELP_CONTEXT command used in this example tells WinHelp to simply display the topic with, in this case, the identifier of 1.

Sometimes, it is helpful to display a primary topic window with a secondary topic window to the side, perhaps one that contains links to related data. This is accomplished in the same manner as above, except that we must provide a name for the secondary window after the help file with the two separated by the > symbol, as the following example demonstrates.

Listing 10-2: Displaying a WinHelp topic and a secondary window

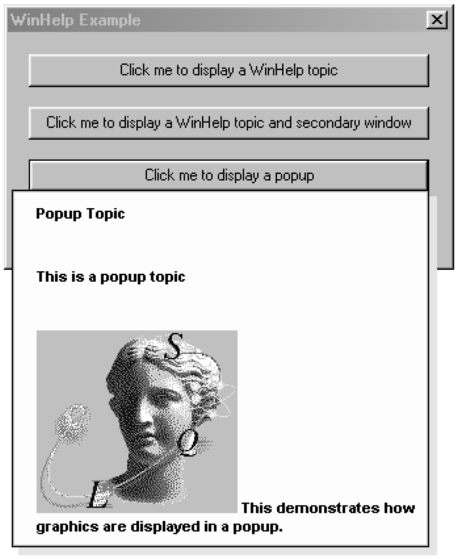
```
procedure TForm1.Button2Click(Sender: TObject);
begin
  {display a topic}
  WinHelp(Handle, PChar(ExtractFilePath(Application.ExeName) +
          '\Winhelpxample.hlp'), HELP_CONTEXT, 1);

  {display another topic in a secondary window}
  WinHelp(Handle, PChar(ExtractFilePath(Application.ExeName) +
          '\Winhelpxample.hlp>second'), HELP_CONTEXT, 3);
end;
```

When displaying context-sensitive help topics, instead of displaying such topics in a normal help window as the above examples demonstrate, it is more elegant to display the topic in a floating pop-up window. WinHelp pop-up windows can contain both text and graphics and can provide for very effective and visually pleasing context-sensitive help. The following example demonstrates this technique.

Listing 10-3: Displaying a pop-up topic

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  {display a topic as a popup}
  WinHelp(Handle, PChar(ExtractFilePath(Application.ExeName) +
          '\Winhelpxample.hlp'), HELP_CONTEXTPOPUP, 4);
end;
```

It is also relatively easy to programmatically provide access to the WinHelp table of contents and search tabs through the Topics dialog box. The following example demonstrates this technique.


WinHelp Example
Click me to display a WinHelp topic
Click me to display a WinHelp topic and secondary window
Click me to display a popup

Popup Topic

This is a popup topic

This demonstrates how graphics are displayed in a popup.

Figure 10-1: The Popup topic

Listing 10-4: Displaying the Topics dialog box

```
procedure TForm1.Button4Click(Sender: TObject);
begin
  {display the Topics dialog box}
  WinHelp(Handle, PChar(ExtractFilePath(Application.ExeName) +
          '\Winhelpxample.hlp'), HELP_FINDER, 0);
end;
```

When an application is closing, it should inform WinHelp that it is no longer needed by using the HELP_QUIT help command. If no other application is using WinHelp, this causes Windows to shut down the WinHelp executable and free it from memory. Typically, this should occur in the main form's OnClose event, as in the following example.

Listing 10-5: Informing WinHelp it is no longer needed

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  {inform WinHelp that we're quitting. if no other application has
   WinHelp open, this causes WinHelp to close}
  WinHelp(Handle, PChar(ExtractFilePath(Application.ExeName) +
          '\Winhelpxample.hlp'), HELP_QUIT, 3);
end;
```

Advantages/Disadvantages

When making the choice between help systems, the developer should keep in mind the following advantages and disadvantages of using the WinHelp help system:

- The WinHelp system has been established for many years. It is flexible, stable, and very familiar to users. Depending on your targeted user base, your users may be more comfortable using WinHelp than the newer HTMLHelp system.

- WinHelp can display pop-ups that contain both text and graphics. HTMLHelp does not offer this ability. If you need to render pop-up help with graphical images, only WinHelp is capable of this at the time of publication.
- Since WinHelp is older technology, it is no longer supported by Microsoft and is no longer in development. It is very possible that WinHelp support could be dropped altogether in future versions of Windows.
- WinHelp help files are much harder to develop than HTMLHelp files due to their reliance on strange rich text formatting tricks. While Microsoft does provide a free help file compiler, the learning curve is very high using these freely available tools, meaning that you can expect a longer development time for help files if the developer is inexperienced.

HTMLHelp

In February 1996, Microsoft unveiled the next generation in help systems. This system features WinHelp's strong points, such as rich text layout and multimedia support, but is based off of the HTML rendering engine utilized by Internet Explorer. Known as HTMLHelp, the system offers all the WinHelp advantages, but because it is based off of HTML, it is much more powerful and flexible. HTMLHelp topics can use ActiveX objects and JavaScript to provide unparalleled flexibility, functionality, and extensibility.

HTMLHelp Architecture

HTMLHelp relies on several COM objects provided by Internet Explorer. HTMLHelp uses these objects as in-process COM servers, and the calling process owns the help viewer window created by a call to HTMLHelp. The result is that multiple applications can have multiple HTMLHelp windows open concurrently, something that cannot be done with WinHelp. Additionally, the help window can be embedded in the calling window itself as is demonstrated in Listing 10-9. Users must have Internet Explorer installed before HTMLHelp is available to an application, and some advanced functionality used by the HTMLHelp file itself (such as DHTML) may also be dependent on the version of IE on the local system.

Creating an HTMLHelp Help File

HTMLHelp files are, as the name implies, authored in HTML format. Therefore, almost any plain text editor (such as Notepad) can be used to create HTMLHelp topics. All the power of HTML can be utilized, including ActiveX components, style sheets, Java, and DHTML. This makes creating HTMLHelp topics very easy, as there is an abundance of information on HTML editing as well as many commercially available applications for designing and creating HTML pages.

Note: Covering the actual authoring of help files is beyond the scope of this book.

There's a little more to creating an HTMLHelp file than putting together the topic pages. HTMLHelp provides the user with a table of contents, a keyword search, and other such tools that allow users to find the information they seek. These are also constructed from HTML files. Microsoft provides a free editor, the HTMLHelp Workshop version 1.31, that includes all of the functionality needed to put together topic files, the table of contents, keyword searching, and anything else the HTMLHelp file needs. At the time of publication, the HTML Help Workshop can be downloaded from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/htmlhelp/html/hwMicrosoftHTMLHelpDownloads.asp. This editor even has the ability to convert old WinHelp projects into the new HTMLHelp format, which helps ease the migration to the new system.

Once the help file is created, it must be compiled into the compressed format that is used by the HTMLHelp API function. As one would expect, the HTML Help Workshop also provides such functionality. The resulting compiled HTML help file contains all of the HTML, graphics, and other files referenced in the help file project. This file is in a structured storage file format, examined in Chapter 11. In addition to HTMLHelp Workshop, there are many other commercially available packages that ease the creation and compiling of HTMLHelp files, such as RoboHelp.

Using HTMLHelp

Before the HTMLHelp API function can be used in an application, the developer needs to acquire the appropriate files that contain the translation of this function, as it is not available as part of the run-time library code that ships with Delphi. At the time of publication, the HTMLHelp file translation unit is available on the Delphi 6 Companion Tools CD or at the following URLs:

http://codecentral.borland.com/codecentral/ccweb.exe/listing?id=15981

http://www.delphi-jedi.org/Jedi:APILIBRARY:44934

The HTMLHelp API function is very similar to the WinHelp API function, containing the same number of parameters that perform pretty much the same function. Calling HTMLHelp to display a topic is relatively straightforward and can be accomplished in only one line of code. The following example demonstrates how simple it is to display a topic in an HTMLHelp help file.

Listing 10-6: Displaying an HTMLHelp topic

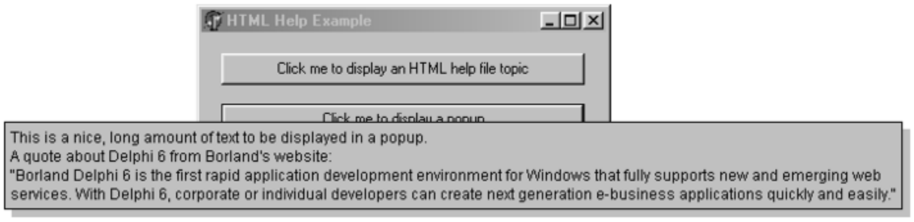
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  {display a help topic}
  HTMLHelp(0, PChar(ExtractFilePath(Application.ExeName) +
           '\HtmlHelpXmple.chm::/Entry3.htm'), HH_DISPLAY_TOPIC, 0);
end;
```

The HTMLHelp API function takes only four parameters: a handle to the window calling the function, the path and filename of the help file, a help command, and a value that is dependent on the help command. The HH_DISPLAY_TOPIC command is very similar to WinHelp's HELP_CONTEXT, which in this example tells HTMLHelp to simply display the topic.

Like WinHelp, HTMLHelp can also display topics in a pop-up window. HTMLHelp provides a little more control over the appearance of the pop-up window, however, such as the font used and the foreground and background colors. The following example demonstrates displaying a pop-up.

Listing 10-7: Displaying an HTMLHelp pop-up

```
procedure TForm1.Button3Click(Sender: TObject);
var
  Popup: THHPopup;    // the popup structure
  Pt: TPoint;         // holds some coordinates
begin
  {initialize the popup information structure}
  Popup.cbStruct := SizeOf(Popup);
  Popup.hinst := 0;
  Popup.idString := 0;
  Popup.pszText := nil;

  {get the mouse cursor position}
  GetCursorPos(Pt);
  Popup.pt := Pt;

  {specify colors}
  Popup.clrForeGround := TColorRef(-1);
  Popup.clrBackground := clLime;

  {specify margins and font}
  Popup.rcMargins := Rect(-1, -1, -1, -1);
  Popup.pszFont := 'Arial, 9';

  {indicate the popup text to display}
  Popup.idString := 2;

  {show the popup}
    HTMLHelp(0, PChar(ExtractFilePath(Application.ExeName) +
             '\HtmlHelpXmple.chm::\cshelp.txt'), HH_DISPLAY_TEXT_POPUP,
             DWORD(@Popup));
end;
```


HTML Help Example
Click me to display an HTML help file topic
Click me to display a popup

This is a nice, long amount of text to be displayed in a popup.
A quote about Delphi 6 from Borland's website:
"Borland Delphi 6 is the first rapid application development environment for Windows that fully supports new and emerging web services. With Delphi 6, corporate or individual developers can create next generation e-business applications quickly and easily."

Figure 10-2: The HTMLHelp popup

Using HTMLHelp to implement context-sensitive help requires a little more work. The most common method for implementing context-sensitive HTML help is to provide a handler for the application's OnHelp event and call HTMLHelp from there. Normally, when the user presses F1, the application's OnHelp event fires, which is sent the help command and any data necessary for the command, and a variable Boolean parameter controls whether or not Delphi ultimately sends the call to WinHelp. However, under Delphi 6, this event is broken. It still functions somewhat if biHelp is included in the BorderIcons property of the form. Fortunately, you can go to http://www.helpware.net/downloads/index.htm to download a unit that fixes this problem.

Assuming that you are using the unit that fixes this problem, or that you've included biHelp in the BorderIcons property (as is done in the following example), it is easy to call HTMLHelp to display context-sensitive topics. Simply provide a handler for the application's OnHelp event and call HTMLHelp, passing it the value from the Data parameter, which will be the value of the HelpContext property of the focused object at the time F1 is pressed. The context ID has been placed in the object's HelpContext property, and the following example demonstrates this technique.

Note: The application's HelpFile property must not contain a value. If the HelpFile property is set, WinHelp is called automatically.

Listing 10-8: Using HTMLHelp for context-sensitive help

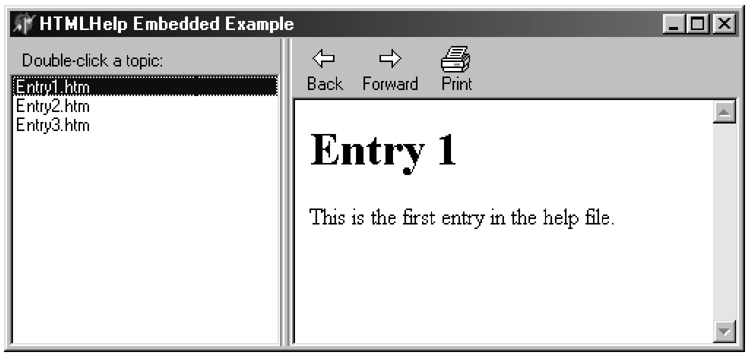
```
procedure TForm1.FormCreate(Sender: TObject);
begin
  {set the application's OnHelp event handler}
  Application.OnHelp := AppHelp;
end;

function TForm1.AppHelp(Command: Word; Data: Integer;
  var CallHelp: Boolean): Boolean;
begin
  {indicate that we do not want the application to call WinHelp}
  CallHelp := FALSE;

  {display the appropriate help topic}
  if Command in [HELP_CONTEXT, HELP_CONTEXTPOPUP] then
    HTMLHelp(0, PChar(ExtractFilePath(Application.ExeName) +
             '\HtmlHelpXmple.chm'), HH_HELP_CONTEXT, Data);

  {indicate that we have handled the message}
  Result := TRUE;
end;
```

HTMLHelp gives the developer a great deal of control over how the help window is displayed. A useful technique for displaying help is to embed the help window into the application itself. For example, such a technique might be employed when providing a walkthrough tutorial, wizard, or other training aid. Embedding the HTMLHelp window into an application takes a little bit of work, but it is a straightforward implementation, as the following example demonstrates.

Listing 10-9: Embedding the HTMLHelp window into an application

```
const
  {the name of our embedded window}
  WINNAME = 'EmbeddedWinXample';

procedure TForm1.FormCreate(Sender: TObject);
var
  WinDef: THHWinType;
begin
  {begin defining the window}
  FillChar(WinDef, SizeOf(WinDef), 0);
  WinDef.cbStruct := SizeOf(WinDef);

  {indicate which members of this structure are valid}
  WinDef.fsValidMembers := HHWIN_PARAM_PROPERTIES or HHWIN_PARAM_STYLES or
                           HHWIN_PARAM_EXSTYLES or HHWIN_PARAM_RECT or
                           HHWIN_PARAM_SHOWSTATE or HHWIN_PARAM_TB_FLAGS;

  {define the appropriate properties for the window}
  WinDef.fsWinProperties := HHWIN_PROP_NOTITLEBAR or HHWIN_PROP_NODEF_STYLES or
                            HHWIN_PROP_NODEF_EXSTYLES or HHWIN_PROP_TRI_PANE;

  {define the appropriate styles for the window. it is the WS_CHILDWINDOW
   style that causes the window to appear embedded}
  WinDef.dwStyles := WS_VISIBLE or WS_CHILDWINDOW;
  WinDef.dwExStyles := 0;

  {define the buttons to be shown}
  WinDef.fsToolBarFlags := HHWIN_BUTTON_BACK or HHWIN_BUTTON_FORWARD or
                           HHWIN_BUTTON_HOME or HHWIN_BUTTON_PRINT;

  WinDef.fUniCodeStrings := False;            // use ASCII strings
  WinDef.pszType := PChar(WINNAME);           // define the window name
  WinDef.pszCaption := nil;                   // no caption (no caption bar)
  WinDef.nShowState := SW_SHOW;               // initially visible
  WinDef.fNotExpanded := True;                // not expanded

  {make the window size the same as the panel that will hold it}
  WinDef.rcWindowPos := Rect(0, 0, pnlHTMLHelpHome.ClientWidth,
                             pnlHTMLHelpHome.ClientHeight);

  {finally, create the window. note that this simply creates a window
   definition, it does not display the window}
  HTMLHelp(0, nil, HH_SET_WIN_TYPE, DWORD(@WinDef));
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  {make sure we close the HTMLHelp window before the application exits}
  HTMLHelp(0, nil, HH_CLOSE_ALL, 0);
end;

procedure TForm1.ListBox1DblClick(Sender: TObject);
begin
```

```
  {when an item in the list box is double-clicked, display the topic. the window handle
   provided in the first parameter becomes the parent window of the help viewer window
   (which is displayed the first time this function is called) because of the WS_CHILD-
   WINDOW style defined in the window definition. note that we must save the handle of the
   created help viewer window so we can resize it when the parent window is resized.}
  HTMLWinHandle := HTMLHelp(pnlHTMLHelpHome.Handle,
                           PChar(ExtractFilePath(Application.ExeName) +
                           'HtmlHelpXmple.chm::/' +
                           ListBox1.Items[ListBox1.ItemIndex] + '>' +
                           WINNAME), HH_DISPLAY_TOPIC, 0);
end;

procedure TForm1.pnlHTMLHelpHomeResize(Sender: TObject);
begin
  {when the panel is resized, we must also resize the embedded window}
  if IsWindow(HTMLWinHandle) then
    SetWindowPos(HTMLWinHandle, 0, 0, 0, pnlHTMLHelpHome.ClientWidth,
                 pnlHTMLHelpHome.ClientHeight,
                 SWP_NOMOVE or SWP_NOACTIVATE or SWP_SHOWWINDOW);
end;

procedure TForm1.FormShow(Sender: TObject);
begin
  {display the first topic when the form is shown}
  ListBox1.ItemIndex := 0;
  ListBox1DblClick(nil);
end;
```



Entry 1

This is the first entry in the help file.

Figure 10-3: The embedded help window

Advantages/Disadvantages

When making the choice between help systems, the developer should keep in mind the following advantages and disadvantages of using the HTMLHelp help system:

- WinHelp can display pop-ups that contain both text and graphics. HTMLHelp does not offer this ability. If you need to render pop-up help with graphical images, only WinHelp is capable of this ability at the time of publication.
- Context-sensitive help using HTMLHelp requires a little more work. The application's OnHelp event must be handled, which makes the call to HTMLHelp. None of this is necessary when using WinHelp, as Delphi calls WinHelp by default.

- Since HTMLHelp is the latest help file technology, it is currently supported and in development. The files necessary for using HTMLHelp are now part of the Windows operating system, and this system should be around for many years to come.
- HTMLHelp files, being based on HTML, are very easy to create. There is an abundance of technical information about HTML editing, as well as many talented individuals who make their living editing HTML files. HTMLHelp files have much more power than the old WinHelp format and are easier to expand with additional functionality.

Delphi vs. the Windows API

Delphi provides a fairly automated method for displaying context-sensitive help topics. By selecting Project | Options, supplying the location of the application's help file, and setting the HelpContext property of controls on the form to the appropriate topic IDs, Delphi will automatically call WinHelp and display the appropriate topic when a user presses F1 on a focused control. This all happens behind the scenes with no extra coding required by the developer. This is an extremely useful feature and should be exploited at every opportunity. The Application object also provides a few methods that encapsulate the most common uses of the WinHelp API function. However, the HTMLHelp API function must be called directly to use any of the HTMLHelp functionality. It is also necessary to call the WinHelp API function directly in order to use any of the more advanced functionality offered by WinHelp.

Help Functions

The following help functions are covered in this chapter.

Table 10-1: Help functions

Function	Description
HTMLHelp	Displays help topics from a compiled HTML help file.
WinHelp	Displays help topics from a Windows help file.

HTMLHelp

Syntax

```
HTMLHelp(
hwndCaller: HWND;          {handle of the calling window}
pszFile: PChar;            {help file or topic}
uCommand: UINT;            {a command flag}
dwData: DWORD              {command-specific data}
): HWND;                   {returns a window handle}
```

Description

This function starts the Microsoft HTML help system and displays help topics from compiled HTML help files. Many different commands are available to affect the appearance and behavior of the HTML help system.

The HTML help system can send messages back to the calling window for certain events. These notification messages are sent through the standard Windows WM_NOTIFY message. The idFrom member of the NMHdr structure pointed to by the lParam member of the WM_NOTIFY message will contain a unique identifier (as defined by the application) that indicates the WM_NOTIFY message was sent from the HTML help viewer window. The Code member of the NMHdr structure may contain one value from Table 10-2. Depending on this code, the lParam member may be typecast into one of the following two data structures, which carry further information about the notification:

For the HHN_NAVCOMPLETE and HHN_WINDOW_CREATE notification messages:

```
THHNNotify = packed record
    hdr: NMHDR;                {standard WM_NOTIFY NMHdr structure}
    pszUrl: PCSTR;             {topic or help window name}
end;
```

hdr: Points to the WM_NOTIFY NMHdr structure (not covered in this text).

pszUrl: A Unicode string containing either the topic to which the user navigated or the name of the help window being created, depending on the notification message code.

For the HHN_TRACK message:

```
THHNTrack = packed record
    hdr: NMHDR;                {standard WM_NOTIFY NMHdr structure}
    pszCurUrl: PCSTR;          {current topic}
    idAction: Integer;         {action flag}
    phhWinType: PHHWinType; {pointer to a THHWinType structure}
end;
```

hdr: Points to the WM_NOTIFY NMHdr structure (not covered in this text).

pszCurUrl: A Unicode string containing the current topic.

idAction: A flag indicating the type of action. This can be one value from Table 10-3.

phhWinType: A pointer to the current THHWinType structure (detailed below).

Parameters

hwndCaller: Specifies the handle of a window from the calling application. The specified window will own any HTML help window created by this function. This window also receives any notification messages sent from the HTML help system.

pszFile: A null-terminated string whose value is dependent on the value of uCommand. For some uCommand values, this parameter specifies the path to a compiled HTML help file or a topic file within a specified help file. A window type name can also be specified. This parameter is formatted in the following manner:

```
Helpfile.chm[::/Topic.htm[>Window Name]]
```

Note: Square brackets ([]) indicate optional elements. You must use "::/" to separate the help file name from the topic and ">" to separate the window name from the other elements.

uCommand: A flag indicating a function to perform. This parameter may be set to one value from Table 10-4.

dwData: Specifies command-specific data. This value is dependent on the value in the uCommand parameter. See Table 10-4 for more details. For various commands, this parameter will contain a pointer to different data structures. These data structures are defined as:

```
THHAKLink = packed record
    cbStruct: Integer;            {size of the structure, in bytes}
    fReserved: BOOL;              {reserved}
    pszKeywords: LPCTSTR;         {list of keywords}
    pszUrl: LPCTSTR;              {lookup failure topic file}
    pszMsgText: LPCTSTR;          {lookup failure text}
    pszMsgTitle: LPCTSTR;         {lookup failure message box title}
    pszWindow: LPCTSTR;           {window type name}
    fIndexOnFail: BOOL;           {display index flag}
end;
```

cbStruct: Indicates the size of the structure, in bytes. This value must be set to SizeOf(THHAKLink).

fReserved: Reserved, but must be set to FALSE.

pszKeywords: A string containing the ALink names or KLink keywords to search. Separate entries by semicolons. ALink names and KLink keywords are case sensitive.

pszUrl: This member indicates the topic file to open and display in case the lookup fails. This member must contain the name of a valid topic; Internet URLs are not supported.

pszMsgText: Indicates text to display in a message box if the lookup fails. A message box is displayed when the lookup fails only when fIndexOnFail is FALSE and pszURL is NIL.

pszMsgTitle: Indicates the caption of the message box that displays the pszMsgText text.

pszWindow: Specifies the name of the window type in which to display the selected topic if the lookup succeeds, the topic specified by pszUrl if the lookup fails, or the index tab if the lookup fails and fIndexOnFail is set to TRUE.

fIndexOnFail: Indicates if the specified keywords should be displayed in the index tab of the help file viewer if the lookup fails.

```
THHFtsQuery = packed record
    cbStruct: Integer;                {size of the structure, in bytes}
    fUniCodeStrings: BOOL;            {Unicode string indicator}
    pszSearchQuery: LPCTSTR;          {search query string}
    iProximity: LongInt;              {word proximity}
    fStemmedSearch: BOOL;             {stemmed search indicator}
    fTitleOnly: BOOL;                 {title only search indicator}
    fExecute: BOOL;                   {initiate search flag}
    pszWindow: LPCTSTR;               {window type name}
end;
```

cbStruct: Indicates the size of the structure, in bytes. This value must be set to SizeOf(THHFtsQuery).

fUniCodeStrings: Set to TRUE if all strings are Unicode.

pszSearchQuery: A null-terminated string indicating the search query.

iProximity: A value indicating the word proximity. This can be set to HH_FTS_DEFAULT_PROXIMITY.

fStemmedSearch: Set to TRUE to indicate a stemmed search.

fTitleOnly: Set to TRUE to perform a title search.

fExecute: Set to TRUE to immediately initiate the search.

pszWindow: Specifies the window type name.

```
THHPopup = packed record
    cbStruct: Integer;                {size of the structure, in bytes}
    hinst: HINST;                     {instance handle}
    idString: UINT;                   {resource id or topic number}
    pszText: LPCTSTR;                 {display text}
    pt: TPoint;                       {pop-up window origins}
    clrForeGround: TColorRef;         {pop-up window foreground color}
    clrBackground: TColorRef;         {pop-up window background color}
    rcMargins: TRect;                 {pop-up window margins}
    pszFont: LPCTSTR;                 {font attributes}
end;
```

cbStruct: Indicates the size of the structure, in bytes. This value must be set to SizeOf(THHPopup).

hinst: Specifies the instance handle of the executable or DLL from which to retrieve a string resource. This member is ignored if idString is set to zero.

idString: Indicates a resource string identifier or a topic number within a text file.

pszText: Indicates the text to display if idString is set to zero.

pt: Indicates the display coordinates of the top-left corner of the pop-up window, in pixels.

clrForeGround: Indicates the foreground color of the pop-up window. This member can be set to –1 to indicate the system color.

clrBackground: Indicates the background color of the pop-up window. This member can be set to –1 to indicate the system color.

rcMargins: Indicates the size of the margins on the top, left, bottom, and right of the pop-up window, in pixels. This member can be set to –1 to use a default margin width.

pszFont: A string indicating the font attributes for the pop-up window text. This string must be in the format:

```
facename[, point size[, charset[ BOLD ITALIC UNDERLINE]]]
```

Note: The square brackets ([]) indicate optional values. To skip a value, simply put in a comma, but leave the value blank.

```
THHLastError = packed record
    cbStruct: Integer;              {size of the structure, in bytes}
    hr: HRESULT;                    {last error code}
    description: PWideChar;         {error description string}
end;
```

cbStruct: Indicates the size of the structure, in bytes. This value must be set to SizeOf(THHLastError).

hr: The last error code value.

description: A Unicode string containing the description of the last error.

```
THHWinType = packed record
    cbStruct: Integer;              {size of the structure, in bytes}
    fUniCodeStrings: BOOL;          {Unicode string indicator}
    pszType: LPCTSTR;               {window type name}
    fsValidMembers: DWORD;          {valid members flags}
    fsWinProperties: DWORD;         {window properties flags}
    pszCaption: LPCTSTR;            {title bar caption}
    dwStyles: DWORD;                {window style flags}
    dwExStyles: DWORD;              {extended window style flags}
    rcWindowPos: TRect;             {window position coordinates}
    nShowState: Integer;            {window display state}
    hwndHelp: HWND;                 {a window handle}
    hwndCaller: HWND;               {a window handle}
    paInfoTypes: PHHInfoType;       {array of information types}

    { The following members are only valid if HHWIN_PROP_TRI_PANE is set }

    hwndToolBar: HWND;              {toolbar window handle}
    hwndNavigation: HWND;           {navigation pane window handle}
    hwndHTML: HWND;                 {topic pane window handle}
    iNavWidth: Integer;             {navigation pane width}
    rcHTML: TRect;                  {coordinates of topic pane}
    pszToc: LPCTSTR;                {contents file}
    pszIndex: LPCTSTR;              {index file}
    pszFile: LPCTSTR;                    {default HTML file}
    pszHome: LPCTSTR;                    {home file}
    fsToolBarFlags: DWORD;               {toolbar appearance flags}
    fNotExpanded: BOOL;                  {navigation pane status}
    curNavType: Integer;                 {default navigation pane tab}
    tabpos: Integer;                     {navigation bar position flags}
    idNotify: Integer;                   {notification message identifier}
    tabOrder: array[0..HH_MAX_TABS] of Byte; {navigation pane tab order}
    cHistory: Integer;                   {number of history items}
    pszJump1: LPCTSTR;                   {jump1 button text}
    pszJump2: LPCTSTR;                   {jump2 button text}
    pszUrlJump1: LPCTSTR;                {jump1 button URL}
    pszUrlJump2: LPCTSTR;                {jump2 button URL}
    rcMinSize: TRect;                    {minimum window size}
    cbInfoTypes: Integer;                {size of paInfoTypes array}
    pszCustomTabs: LPCTSTR;              {custom tab titles}
end;
```

cbStruct: Indicates the size of the structure, in bytes. This member must be set to SizeOf(THHWinType).

fUniCodeStrings: Indicates if the strings specified in this structure are Unicode.

pszType: A null-terminated string containing the name of the window type.

fsValidMembers: A series of flags indicating which members of this structure contain information and should be used. This member can contain one or more values from Table 10-5.

fsWinProperties: A series of flags that specify the properties of the window. This member can contain one or more values from Table 10-6.

pszCaption: A null-terminated string indicating the caption to be displayed in the window title bar.

dwStyles: Indicates the window styles for the new window. This member can contain one or more values from Table 10-7.

dwExStyles: Indicates the extended window styles for the new window. This member can contain one or more values from Table 10-8.

rcWindowPos: A TRect structure indicating the coordinates and dimensions of the window, in pixels. Any negative value will not affect the window coordinate or position (i.e., setting the Right and Bottom members to –1 would allow a calling application to move the window without affecting its current width and height).

nShowState: Indicates how the window is initially displayed. This member can contain one value from Table 10-9.

hwndHelp: A handle to the help viewer window, if the window has been created.

hwndCaller: A handle to the window that receives WM_NOTIFY messages sent from the HTMLHelp viewer window.

paInfoTypes: A pointer to an array of information types (the PHHInfoType type is defined as a pointer to a DWORD). Each information type is a bit flag. When bit zero is not set, all HTMLHelp commands that use information types will use this information to determine what navigational interface should be displayed and where hyperlinks should jump.

hwndToolBar: Specifies the handle of the toolbar (available only when the fsWinProperties member contains HHWIN_PROP_TRI_PANE).

hwndNavigation: Specifies the handle of the navigation pane (available only when the fsWinProperties member contains HHWIN_PROP_TRI_PANE).

hwndHTML: Specifies the handle of the topic pane (available only when the fsWinProperties member contains HHWIN_PROP_TRI_PANE).

iNavWidth: Specifies the width of the navigation pane (available only when the fsWinProperties member contains HHWIN_PROP_TRI_PANE).

rcHTML: Specifies the rectangular coordinates of the topic pane (available only when the fsWinProperties member contains HHWIN_PROP_TRI_PANE).

pszToc: A null-terminated string indicating the contents file to display in the navigation pane (available only when the fsWinProperties member contains HHWIN_PROP_TRI_PANE).

pszIndex: A null-terminated string indicating the index file to display in the navigation pane (available only when the fsWinProperties member contains HHWIN_PROP_TRI_PANE).

pszFile: A null-terminated string indicating the default HTML file to display in the topic pane (available only when the fsWinProperties member contains HHWIN_PROP_TRI_PANE).

pszHome: A null-terminated string indicating the HTML file to display in the topic pane when the Home button is clicked (available only when the fsWinProperties member contains HHWIN_PROP_TRI_PANE).

fsToolBarFlags: A series of flags indicating which buttons appear on the toolbar. This may be a combination of values from Table 10-10 (available only when the fsWinProperties member contains HHWIN_PROP_TRI_PANE).

fNotExpanded: A value of TRUE indicates that the HTMLHelp viewer should open with the navigation pane closed (available only when the fsWinProperties member contains HHWIN_PROP_TRI_PANE).

curNavType: A flag indicating which tab in the navigation pane receives focus by default (available only when the fsWinProperties member contains HHWIN_PROP_TRI_PANE). This member may be set to one value from Table 10-11.

tabpos: A flag indicating the position of the tabs in the navigation pane (available only when the fsWinProperties member contains HHWIN_PROP_TRI_PANE). This member may be set to one value from Table 10-12.

idNotify: A unique value that is used to identify the HTMLHelp viewer window. This value is sent in the idFrom member of the NMHdr structure pointed to by the lParam member of the WM_NOTIFY message sent to the owning window when notification is enabled.

tabOrder: An array of bytes indicating the order in which the tabs should appear on the navigation pane. Each element of the array indicates one of the tabs (available only when the fsWinProperties member contains HHWIN_PROP_TRI_PANE). See Table 10-13 for a list of defined tabs. The value assigned to an element in the array indicates the zero-based order in which that tab appears on the navigation pane (i.e., tabOrder[HH_TAB_SEARCH] := 0 and tabOrder[HH_TAB_CONTENTS] := 1 would put the Search tab first, followed by the Contents tab).

cHistory: Indicates the number of items to keep in history. The default value of this member is 30.

pszJump1: A null-terminated string indicating the text to display under the Jump1 button.

pszJump2: A null-terminated string indicating the text to display under the Jump2 button.

pszUrlJump1: A null-terminated string indicating the URL to display when the Jump1 button is clicked.

pszUrlJump2: A null-terminated string indicating the URL to display when the Jump2 button is clicked.

rcMinSize: Indicates the minimum size of the HTMLHelp viewer window.

cbInfoTypes: Indicates the number of items pointed to by the paInfoTypes member.

pszCustomTabs: A series of strings indicating the titles for custom tabs in the navigation pane. Each string is separated by a null terminator, and the entire string is terminated by a double null.

Return Value

The return value is dependent on some commands. However, this function typically returns the handle to the HTMLHelp window when successful or a zero when the function failed.

See Also

WinHelp

Example

Please see Listing 10-6 and other examples in the introduction.

Table 10-2: HTML help notification message code values (WM_NOTIFY NMHdr.Code values)

Value	Description
HHN_NAVCOMPLETE	Sent when navigation to a new topic has completed. The lParam member of the WM_NOTIFY message can be typecast to a THHNNotify structure to retrieve additional information.

Value	Description
HHN_TRACK	Sent when the user clicks on a button on the HTMLHelp viewer window or when a new tab is selected in the navigation pane. This message is sent before the event is completed. The lParam member of the WM_NOTIFY message can be typecast to a THHNTrack structure to retrieve additional information.
HHN_WINDOW_CREATE	Sent just before the HTMLHelp viewer window is created. The lParam member of the WM_NOTIFY message can be typecast to a THHNNotify structure to retrieve additional information.

Table 10-3: HTMLHelp THHNTrack.idAction values

Value	Description
HHACT_BACK	The user clicked the Back button.
HHACT_CONTRACT	The user clicked the Hide button.
HHACT_CUSTOMIZE	The user clicked the Customize button.
HHACT_EXPAND	The user clicked the Show button.
HHACT_FORWARD	The user clicked the Forward button.
HHACT_HIGHLIGHT	The user clicked the Highlight button.
HHACT_HOME	The user clicked the Home button.
HHACT_JUMP1	The user clicked the Jump1 button.
HHACT_JUMP2	The user clicked the Jump2 button.
HHACT_NOTES	The user clicked the Notes button.
HHACT_OPTIONS	The user clicked the Options button.
HHACT_PRINT	The user clicked the Print button.
HHACT_REFRESH	The user clicked the Refresh button.
HHACT_STOP	The user clicked the Stop button.
HHACT_SYNC	The user clicked the Locate button.
HHACT_TAB_CONTENTS	The user clicked the Contents tab.
HHACT_TAB_FAVORITES	The user clicked the Favorites tab.
HHACT_TAB_HISTORY	The user clicked the History tab.
HHACT_TAB_INDEX	The user clicked the Index tab.
HHACT_TAB_SEARCH	The user clicked the Search tab.
HHACT_TOC_NEXT	The user clicked the Next button.
HHACT_TOC_PREV	The user clicked the Previous button.
HHACT_ZOOM	The user clicked the Zoom button.

Table 10-4: HTMLHelp uCommand values

Value	Description
HH_ALINK_LOOKUP	Looks up one or more associative link (or Alink) names in the compiled help file. pszFile: Contains the path and file name of the compiled help file. dwData: Contains a pointer to a THHAKLink structure which defines the ALink names to search for, as well as what action to perform if none of the ALink names are found. See the dwData parameter for more details.
HH_CLOSE_ALL	Closes all open HTML help file windows opened by the calling application. hwndCaller: Set to zero. pszFile: Set to NIL. dwData: Set to zero.
HH_DISPLAY_INDEX	Selects the Index tab in the HTML help file viewer window and searches for a specified keyword. pszFile: Specifies the path and file name of the compiled help file or a topic within the compiled help file. dwData: Contains a pointer to the search keyword.
HH_DISPLAY_SEARCH	Selects the Search tab in the HTML help file viewer window. pszFile: Specifies the path and file name of the compiled help file or a topic within the compiled help file. dwData: Contains a pointer to a THHFtsQuery structure containing the search parameters. See the dwData parameter for more details.
HH_DISPLAY_TEXT_POPUP	Opens a pop-up window displaying an explicit text string, a resource string, or the contents of a text file compiled into the HTML help file. dwData: Contains a pointer to a THHPopup structure containing the attributes of the pop-up window. See the dwData parameter for more details.
HH_DISPLAY_TOC	Selects the Contents tab in the HTML help file viewer window. pszFile: Specifies the path and file name of the compiled help file or a topic within the compiled help file. dwData: Contains zero or a pointer to a string specifying a topic within the compiled help file.
HH_DISPLAY_TOPIC	Opens a compiled help file and displays the specified topic within the file. pszFile: Specifies the path and file name of the compiled help file or a topic within the compiled help file dwData: Contains zero or a pointer to a string specifying a topic within the compiled help file.

Value	Description
HH_GET_LAST_ERROR	Retrieves information about the last error that occurred in the HTML help system. pszFile: Contains NIL. dwData: Contains a pointer to a THHLastError structure containing information about the last error that occurred in the HTML help system. See the dwData parameter for more details.
HH_GET_WIN_HANDLE	Retrieves the handle of the specified window type. pszFile: Specifies the path and file name of the compiled help file containing the desired window type. dwData: A pointer to a string containing the name of the window type whose handle is to be retrieved. HTMLHelp will return zero if this help window has not yet been created.
HH_GET_WIN_TYPE	Retrieves a pointer to a THHWinType data structure containing the attributes for the specified window type. pszFile: Specifies the path and file name of the compiled help file and the name of the window type whose information is to be retrieved. dwData: Contains a pointer to a THHWinType structure. This structure is filled out and returned by the function. See the dwData parameter for more details. If successful, the function returns the handle of the help window or zero if the help window has not yet been created. It returns –1 on failure.
HH_HELP_CONTEXT	Opens a compiled help file and displays the topic associated with the specified topic identifier. This is most commonly used for displaying context-sensitive topics. pszFile: Specifies the path and file name of the compiled help file. dwData: Contains the numeric identifier of the topic to display.
HH_KEYWORD_LOOKUP	Looks up one or more keywords in the compiled help file. pszFile: Contains the path and file name of the compiled help file. dwData: Contains a pointer to a THHAKLink structure which defines the keywords to search for, as well as what action to perform if none of the keywords are found. See the dwData parameter for more details.

Value	Description
HH_SET_WIN_TYPE	Creates a new HTMLHelp viewer window or modifies an existing window. pszFile: Specifies the path and file name of the compiled help file and the name of the window type to create or whose information is to be modified. dwData: Contains a pointer to a THHWinType structure which defines the new window attributes. See the dwData parameter for more details. If successful, the function returns the handle of the help window or 0 if the help window has not yet been created. It returns –1 on failure.
HH_SYNC	Selects the entry in the Contents tab for the topic currently displayed in the topic pane. pszFile: Specifies the path and file name of the compiled help file and the name of the window type that is displayed. dwData: Contains a pointer to a null-terminated string containing the topic within the help file to which the contents tab is to be synchronized.

Table 10-5: HTMLHelp THHWinType.fsValidMembers values

Value	Description
HHWIN_PARAM_PROPERTIES	Valid fsWinProperties member
HHWIN_PARAM_STYLES	Valid dwStyles member
HHWIN_PARAM_EXSTYLES	Valid dwExStyles member
HHWIN_PARAM_RECT	Valid rcWindowPos member
HHWIN_PARAM_NAV_WIDTH	Valid iNavWidth member
HHWIN_PARAM_SHOWSTATE	Valid nShowState member
HHWIN_PARAM_INFOTYPES	Valid paInfoTypes member
HHWIN_PARAM_TB_FLAGS	Valid fsToolBarFlags member
HHWIN_PARAM_EXPANSION	Valid fNotExpanded member
HHWIN_PARAM_TABPOS	Valid tabpos member
HHWIN_PARAM_TABORDER	Valid tabOrder member
HHWIN_PARAM_HISTORY_COUNT	Valid cHistory member
HHWIN_PARAM_CUR_TAB	Valid curNavType member

Table 10-6: HTMLHelp THHWinType.fsWinProperties values

Value	Description
HHWIN_PROP_TAB_AUTOHIDESHOW	Automatically hides or shows the tri-pane window.
HHWIN_PROP_ONTOP	Causes the window to stay on top of all windows on the desktop.

Value	Description
HHWIN_PROP_ONTOP (cont.)	**Note**: This flag is ignored if the HHWIN_PROP_NODEF_ EXSTYLES flag is included.
HHWIN_PROP_NOTITLEBAR	The window has no title bar.
HHWIN_PROP_NODEF_STYLES	Indicates that the window should not use default window styles when created. When this flag is not specified, the window is created with the WS_THICKFRAME, WS_OVERLAPPED, and WS_VISIBLE styles by default.
HHWIN_PROP_NODEF_EXSTYLES	Indicates that the window should not use default extended window styles when created.
HHWIN_PROP_TRI_PANE	Creates the standard HTML help viewer tri-pane window.
HHWIN_PROP_NOTB_TEXT	Suppresses display of text below the icon on buttons on the toolbar.
HHWIN_PROP_POST_QUIT	Sends a WM_QUIT message to the window specified by the hwndCaller parameter when the HTML help viewer closes.
HHWIN_PROP_AUTO_SYNC	Locates and selects the table of contents or index entry (depending on which tab is selected) for the displayed help topic.
HHWIN_PROP_TRACKING	Indicates that notification messages should be sent to the window specified by the hwndCaller parameter.
HHWIN_PROP_TAB_SEARCH	Creates a window with a Search tab in the navigation pane.
HHWIN_PROP_TAB_HISTORY	Creates a window with a History tab in the navigation pane.
HHWIN_PROP_TAB_FAVORITES	Creates a window with a Favorites tab in the navigation pane.
HHWIN_PROP_CHANGE_TITLE	Changes the title bar of the window to the title of the current displayed topic.
HHWIN_PROP_NAV_ONLY_WIN	Causes the window to contain only a navigation pane and toolbar.
HHWIN_PROP_NO_TOOLBAR	Creates a window with no toolbar.
HHWIN_PROP_MENU	Creates a window with a menu.
HHWIN_PROP_TAB_ADVSEARCH	Creates a window with a full-text Search tab in the navigation pane. **Note**: HHWIN_PROP_TAB_SEARCH must also be specified when this flag is used.
HHWIN_PROP_USER_POS	When reopened, this flag causes the window to reposition itself to its last known size and position.
HHWIN_PROP_TAB_CUSTOM1	Creates a window with a custom tab in the navigation pane. 1 through 9 can be specified to create up to nine new custom tabs.

Table 10-7: HTMLHelp THHWinType.dwStyles values

Value	Description
WS_BORDER	Gives the window a thin line border.
WS_CAPTION	Gives the window a title bar and includes the WS_BORDER style.
WS_CHILD	Creates a child window. The WS_POPUP style cannot be used if this style is specified.
WS_CHILDWINDOW	The same as the WS_CHILD style.
WS_CLIPCHILDREN	Clips around child windows during painting and is used when creating parent windows.
WS_CLIPSIBLINGS	Clips windows relative to each other during painting. Without this style, the entire area of the window will be included in the update region even if overlapped by a sibling window, making it possible to draw in the client area of the overlapping child window. When this style is used, the siblings' overlapping area is left out of the update region.
WS_DISABLED	The window is initially disabled and cannot receive user input.
WS_DLGFRAME	Creates a window with the dialog box border style and cannot have a title bar.
WS_GROUP	Marks the beginning of a group of controls. The next controls created will belong to this group, and when the WS_GROUP style is used again, it will end the grouping and create a new group. The user can change the focus from one control to the next in a group by using the cursor keys. This is commonly used when creating radio buttons.
WS_HSCROLL	Gives the window a horizontal scroll bar.
WS_ICONIC	This is the same as WS_MINIMIZE.
WS_MAXIMIZE	The window starts out maximized.
WS_MAXIMIZEBOX	Includes the maximize button in the title bar.
WS_MINIMIZE	The window starts out minimized.
WS_MINIMIZEBOX	Includes the minimize button in the title bar.
WS_OVERLAPPED	Gives the window both a title bar and a border. This is the same as the WS_TILED style.
WS_OVERLAPPEDWINDOW	Combines the WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX, and WS_MAXIMIZEBOX styles. This is a standard window and is the same as the WS_TILEDWINDOW style.
WS_POPUP	Creates a pop-up window. The WS_CHILD style cannot be used with this style.

Value	Description
WS_POPUPWINDOW	Combines the WS_BORDER, WS_POPUP, and WS_SYSMENU styles. The WS_CAPTION style must be specified before the system menu becomes visible.
WS_SIZEBOX	The window has a sizing border. This is the same as the WS_THICKFRAME style.
WS_SYSMENU	The system menu box is present in the title bar. The WS_CAPTION style must also be specified.
WS_TABSTOP	Indicates that the control can receive the keyboard focus when the user presses the Tab key. Pressing the Tab key again will change the focus to the next control with this style.
WS_THICKFRAME	Gives the window a sizing border.
WS_TILED	This is the same as the WS_OVERLAPPED style.
WS_TILEDWINDOW	This is the same as the WS_OVERLAPPEDWINDOW style.
WS_VISIBLE	The window is initially visible.
WS_VSCROLL	Gives the window a vertical scroll bar.

Table 10-8: HTMLHelp THHWinType.dwExStyles values

Value	Description
WS_EX_ACCEPTFILES	Accepts files dragged and dropped from other applications, such as Windows Explorer.
WS_EX_APPWINDOW	Forces a top-level window onto the taskbar when the window is minimized.
WS_EX_CLIENTEDGE	The window border has a sunken edge.
WS_EX_CONTEXTHELP	Causes the context-sensitive help button (a small button with a question mark) to appear in the title bar. When pressed, the mouse cursor changes to a pointer and a question mark. If the user clicks on a child window or control, it receives a WM_HELP message. The child should pass the message to the parent's window procedure, which should then call the WinHelp function using the HELP_WM_HELP command. The help application displays a pop-up window that usually contains help information for the child window. The WS_MAXIMIZEBOX and WS_MINIMIZEBOX styles must not be included or the context help button will be obscured by the minimize and maximize buttons.
WS_EX_CONTROLPARENT	Allows users to press the Tab key to move from child window to child window.
WS_EX_DLGMODALFRAME	This window has a double border. The WS_CAPTION style must be used to add a title to this style of window.

Value	Description
WS_EX_LAYERED	**Windows 2000 or later**: Creates a layered window (not covered in this text). This flag cannot be used for child windows nor can it be used in conjunction with the CS_OWNDC or CS_CLASSDC styles.
WS_EX_LAYOUTRTL	**Windows 2000 or later, Arabic and Hebrew versions of Windows 98 and Me**: Creates a window with a horizontal origin on its right edge.
WS_EX_LEFT	Creates a window with left-aligned properties. This is the default style.
WS_EX_LEFTSCROLLBAR	If the shell's language is Hebrew, Arabic, or any other language that supports reading order alignment, the vertical scroll bar, if any, will be placed to the left of the client area. For other languages, this style is simply ignored.
WS_EX_LTRREADING	Text displayed in this window is in a left-to-right reading order. This is the default style.
WS_EX_MDICHILD	Creates an MDI child window.
WS_EX_NOACTIVATE	**Windows 2000 or later**: Any top-level window created with this style will not become the foreground window when it is clicked nor will the window be brought to the foreground when the user minimizes or closes the foreground window. Windows with this style will also not appear on the taskbar by default.
WS_EX_NOINHERITLAYOUT	**Windows 2000 or later**: This style prevents the window from passing its window layout to its child windows.
WS_EX_NOPARENTNOTIFY	A window with this style does not send WM_PARENTNOTIFY messages to its parent when it is created or destroyed.
WS_EX_OVERLAPPEDWINDOW	Combines the WS_EX_CLIENTEDGE and WS_EX_WINDOWEDGE styles.
WS_EX_PALETTEWINDOW	Combines the WS_EX_WINDOWEDGE, WS_EX_TOOLWINDOW, and WS_EX_TOPMOST styles.
WS_EX_RIGHT	If the shell's language is Hebrew, Arabic, or any other language that supports reading order alignment, this window has generic right-aligned properties. For other languages, this style is simply ignored.
WS_EX_RIGHTSCROLLBAR	Places the vertical scroll bar, if present, on the right side of the client area. This is the default style.
WS_EX_RTLREADING	If the shell's language is Hebrew, Arabic, or any other language that supports reading order alignment, the window is displayed using right-to-left reading order properties. For other languages, this style is simply ignored.

Value	Description
WS_EX_STATICEDGE	Creates a window with a three-dimensional border style.
WS_EX_TOOLWINDOW	Creates a floating toolbar style window. The title bar is shorter than a normal title bar, and the window caption is drawn in a smaller font. This style of window will not show up on the task bar or when the user presses Alt+Tab.
WS_EX_TOPMOST	This window stays above all other windows, even when deactivated. This style can be set using the SetWindowPos function.
WS_EX_TRANSPARENT	Any sibling windows that are beneath this window are not obscured by it and will receive the WM_PAINT message first.
WS_EX_WINDOWEDGE	This window has a border with a raised edge.

Table 10-9: HTMLHelp THHWinType.nShowState values

Value	Description
SW_HIDE	The window is hidden and another window is activated.
SW_MINIMIZE	The window is minimized and the next top-level window in the system window list is activated.
SW_RESTORE	The window is activated and displayed in its original size and position.
SW_SHOW	The window is activated and displayed in its current size and position.
SW_SHOWMAXIMIZED	The window is activated and displayed in a maximized state.
SW_SHOWMINIMIZED	The window is activated and displayed as an icon.
SW_SHOWMINNOACTIVE	The window is displayed as an icon. The active window remains active.
SW_SHOWNA	The window is displayed in its current state. The active window remains active.
SW_SHOWNOACTIVE	The window is displayed in its most recent state. The active window remains active.
SW_SHOWNORMAL	This is the same as SW_RESTORE.

Table 10-10: HTMLHelp THHWinType.fsToolBarFlags values

Value	Description
HHWIN_BUTTON_EXPAND	Expand button
HHWIN_BUTTON_BACK	Back button
HHWIN_BUTTON_FORWARD	Forward button
HHWIN_BUTTON_STOP	Stop button
HHWIN_BUTTON_REFRESH	Refresh button
HHWIN_BUTTON_HISTORY	History button

Value	Description
HHWIN_BUTTON_FAVORITES	Favorites button
HHWIN_BUTTON_SYNC	Synchronize button
HHWIN_BUTTON_CONTENTS	Table of Contents button
HHWIN_BUTTON_INDEX	Index button
HHWIN_BUTTON_SEARCH	Search button
HHWIN_BUTTON_OPTIONS	Options button
HHWIN_BUTTON_PRINT	Print button
HHWIN_BUTTON_JUMP1	Jump button (custom-defined)
HHWIN_BUTTON_JUMP2	Jump button (custom-defined)
HHWIN_BUTTON_ZOOM	Zoom button
HHWIN_BUTTON_TOC_NEXT	Table of Contents Next button
HHWIN_BUTTON_TOC_PREV	Table of Contents Previous button
HHWIN_DEF_BUTTONS	Default buttons (Expand, Back, Options, and Print)

Table 10-11: HTMLHelp THHWinType.curNavType values

Value	Description
HHWIN_NAVTYPE_TOC	Displays the Contents tab.
HHWIN_NAVTYPE_INDEX	Displays the Index tab.
HHWIN_NAVTYPE_SEARCH	Displays the Search tab.
HHWIN_NAVTYPE_FAVORITES	Displays the Favorites tab.
HHWIN_NAVTYPE_AUTHOR	Displays the Author tab.

Table 10-12: HTMLHelp THHWinType.tabpos values

Value	Description
HHWIN_NAVTAB_TOP	Tabs are at the top.
HHWIN_NAVTAB_LEFT	Tabs are to the left.
HHWIN_NAVTAB_BOTTOM	Tabs are on the bottom.

Table 10-13: HTMLHelp THHWinType.tabOrder array index values

Value	Description
HH_TAB_CONTENTS	Contents tab
HH_TAB_INDEX	Index tab
HH_TAB_SEARCH	Search tab
HH_TAB_FAVORITES	Favorites tab
HH_TAB_HISTORY	History tab
HH_TAB_AUTHOR	Author tab

WinHelp Windows.pas

Syntax

```
WinHelp(
hWndMain: HWND;          {handle of calling window}
lpszHelp: PChar;         {help file path}
uCommand: UINT;          {a command flag}
dwData: DWORD            {command-specific data}
): BOOL;                 {returns TRUE or FALSE}
```

Description

This function starts the Microsoft help system and displays help topics from a help file. Many different commands are available to affect the appearance and behavior of the Microsoft help system.

Parameters

hWndMain: Specifies the handle of a window from the calling application. This window also receives the notification messages sent from the help system when using training cards.

lpszHelp: A null-terminated string containing the path and name of the help file. A secondary window name can also be specified (in which case the topic is displayed in the secondary window). This parameter is formatted in the following manner:

```
Helpfile.hlp[>Secondary Window Name]
```

Note: Square brackets ([]) indicate optional elements. You must use ">" to separate the window name from the other elements.

uCommand: A flag indicating a function to perform. This parameter may be set to one value from Table 10-14.

dwData: Specifies command-specific data. This value is dependent on the value in the uCommand parameter. See Table 10-14 for more details. For various commands, this parameter will contain a pointer to different data structures. These data structures are defined as:

```
TMultiKeyHelp = record
    mkSize: DWORD;                          {structure size, in bytes}
    mkKeylist: AnsiChar;                    {keyword table}
    szKeyphrase: array[0..0] of AnsiChar;   {keyword}
end;
```

mkSize: Specifies the size of the structure, in bytes. This member must be set to SizeOf(TMultiKeyHelp).

mkKeylist: A character identifying the keyword table to search. Keyword tables are defined in the help file.

szKeyphrase: A null-terminated string containing the keyword to find.

```
THelpWinInfo = record
    wStructSize: Integer;                    {size of structure, in bytes}
    x: Integer;                              {horizontal coordinate}
    y: Integer;                              {vertical coordinate}
    dx: Integer;                             {width}
    dy: Integer;                             {height}
    wMax: Integer;                           {show flag}
    rgchMember: array[0..1] of AnsiChar;     {window name}
end;
```

wStructSize: Specifies the size of the structure, in bytes. This member must be set to SizeOf(THelpWinInfo).

x: The horizontal coordinate of the upper-left corner of the help window, in pixels.

y: The vertical coordinate of the upper-left corner of the help window, in pixels.

dx: The width of the window, in pixels.

dy: The height of the window, in pixels.

wMax: A flag indicating how the window should be shown. This can be one value from Table 10-15.

rgchMember: The name of the window to which these position and size values should be applied.

Return Value

If the function was successful, it returns TRUE; otherwise, it returns FALSE. To retrieve extended error information, use the GetLastError function.

See Also

HTMLHelp

Example

Please see Listing 10-1 and other examples from the introduction.

Table 10-14: WinHelp uCommand values

Value	Description
HELP_COMMAND	Executes a help macro or macro string. dwData: A pointer to a null-terminated string containing the name of the help macro to run or the macro string to run. Multiple help macro names can be specified by separating them with a semicolon.
HELP_CONTENTS	Displays the topic identified as the Contents topic of the help file. dwData: Unused, set to zero. **Note**: This command is for backward compatibility only. Modern applications should include a Contents file with the help file and use the HELP_FINDER command.

Value	Description
HELP_CONTEXT	Displays the topic associated with the specific topic identifier.
	dwData: Contains the topic identifier.
HELP_CONTEXTPOPUP	Displays the topic associated with the specified topic identifier in a pop-up window.
	dwData: Contains the topic identifier.
HELP_FINDER	Displays the Topics dialog box.
	dwData: Unused, set to zero.
HELP_FORCEFILE	Forces the correct help file to be displayed. If a help file is displayed other than the one specified, the specified help file is opened and displayed.
	dwData: Unused, set to zero.
HELP_HELPONHELP	Displays help topics on how to use the help system.
	dwData: Unused, set to zero.
	Note: The winhlp32.hlp file must be available before this command can function.
HELP_INDEX	Same as HELP_CONTENTS.
HELP_KEY	Displays the topic associated with the specified keyword in the help file's keyword table. This must be an exact match. In the event that multiple topics are found, the Index tab is displayed and all matching topics are listed.
	dwData: Specifies the address of a null-terminated string containing the keyword. Multiple keywords can be specified by separating them with a semicolon.
HELP_MULTIKEY	Displays the topic associated with the specified keyword in the help file's alternative keyword table. This must be an exact match. In the event that multiple topics are found, the Index tab is displayed and all matching topics are listed.
	dwData: Specifies a pointer to a TMultiKeyHelp structure containing the keyword to find and the keyword table to search.
HELP_PARTIALKEY	Displays the topic associated with the specified keyword in the help file's keyword table. This does not have to be an exact match. In the event that multiple topics are found, the Index tab is displayed and all matching topics are listed.
	dwData: Specifies the address of a null-terminated string containing the keyword. Multiple keywords can be specified by separating them with a semicolon.
HELP_QUIT	Informs the system that the application no longer needs the help system. If no other application is using the help system, the help system is shut down and removed from memory.
	dwData: Unused, set to zero.

Value	Description
HELP_SETCONTENTS	Identifies a topic to be used as the Contents topic. This topic is displayed when the user clicks the Contents button, if no contents file is available.
	dwData: Specifies the context identifier for the topic.
HELP_SETPOPUP_POS	Sets the position of pop-up windows.
	dwData: Specifies the horizontal and vertical position of the upper-left corner of the pop-up window. Use MakeLong to package the two values into one number.
HELP_SETWINPOS	Sets the size and position of the help window.
	dwData: Specifies a pointer to a THelpWinInfo structure containing the size and position information.
HELP_TCARD	Indicates that the command is for a training card session. This command must be combined with other commands from this table and cannot be used alone.

Table 10-15: WinHelp THelpWinInfo.wMax values

Value	Description
SW_HIDE	The window is hidden and another window is activated.
SW_MINIMIZE	The window is minimized and the next top-level window in the system window list is activated.
SW_RESTORE	The window is activated and displayed in its original size and position.
SW_SHOW	The window is activated and displayed in its current size and position.
SW_SHOWMAXIMIZED	The window is activated and displayed in a maximized state.
SW_SHOWMINIMIZED	The window is activated and displayed as an icon.
SW_SHOWMINNOACTIVE	The window is displayed as an icon. The active window remains active.
SW_SHOWNA	The window is displayed in its current state. The active window remains active.n
SW_SHOWNOACTIVE	The window is displayed in its most recent state. The active window remains active.
SW_SHOWNORMAL	This is the same as SW_RESTORE.

Chapter 11

Shell File Functions

The file system is an integral part of any operating system. As such, the presentation of the file system is an important duty of an operating system's shell, resulting in many shell-based functions that affect or manipulate the file system. This chapter examines some of the available Windows shell functions that manipulate files or provide additional functionality to an application that uses files.

File-based Applications

The drag-and-drop functions and file manipulation functions can greatly enhance the functionality of file-based applications. The ability to drag a file onto an application to open it is essential for a well-behaved, robust file-based application. The DragAcceptFiles, DragQueryFile, and DragFinish functions are all a developer needs to implement a very user-friendly and intuitive method for users to open files.

The following launch bar example demonstrates the use of a variety of the shell file functions. When files are dragged from the Explorer and dropped on the launch bar window, the application looks at each file dropped and creates a speed button for each executable file. An icon is extracted from the executable file and used as the glyph on the speed button. Then, the ShellExecute function is used in the OnClick event of the button to launch the executable file.

Listing 11-1: An application launch bar with file drag-and-drop functionality

```
unit LaunchU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, Buttons, ShellAPI, ExtCtrls;

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure SpeedButtonClick(Sender: TObject);
  private
    { Private declarations }
    {the message handler required to process dropped files}
    procedure WMDropFiles(var Msg: TWMDropFiles); message WM_DROPFILES;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  {register the form as a file drop target}
  DragAcceptFiles(Form1.Handle, TRUE);
end;

procedure TForm1.WMDropFiles(var Msg: TWMDropFiles);
var
  NumDroppedFiles: UINT;        // holds the number of files dropped
  TextBuffer: PChar;            // holds the filename
  BufferSize: UINT;             // the buffer size required to hold the filename
  Count: Integer;               // a general loop control variable
  LargeIcon, SmallIcon: HICON;  // holds handles to the icons for the file
begin
  {retrieve the number of files dropped on the form}
  NumDroppedFiles := DragQueryFile(Msg.Drop, $FFFFFFFF, nil, 0);

  {for every file dropped on the form...}
  for Count := 0 to NumDroppedFiles - 1 do
  begin
    {get the size of the filename and allocate a string large enough to hold it (add one
     to the text buffer size for the null terminator)}
    BufferSize := DragQueryFile(Msg.Drop,Count,nil,0);
    TextBuffer := StrAlloc(BufferSize + 1);

    {the filename}
    DragQueryFile(Msg.Drop, Count, TextBuffer, BufferSize + 1);

    {if the file is an executable...}
    if (ExtractFileExt(UpperCase(string(TextBuffer))) = '.EXE')
         then
      {...create a speed button for it and initialize properties}
      with TSpeedButton.Create(Form1) do
      begin
        Parent := Form1;

        {the hint is used to store the path and filename of the executable dropped on the
         form. the shorthint part holds only the filename (this is what is displayed when
         the mouse cursor is moved over the button), and the longhint part holds the full
         path and filename. this part is used to launch the executable.}
        Hint := ExtractFileName(string(TextBuffer)) + '|' +
                TextBuffer;
        ShowHint := TRUE;
        {set the left side of the button. if it is the first one
         on the form, its left side is set to 4.}
        if Form1.ComponentCount=1 then
         Left := 4
        else
         Left := TSpeedButton(Form1.Components[Form1.Component-Count-2]).Left +
                TSpeedButton(Form1.Components[Form1.ComponentCount-2]).Width+4;

        Top := 4;

        {set the OnClick method so the button does something}
        OnClick := SpeedButtonClick;

        {this extracts the small icon from the executable
         and displays it in the glyph for the speedbutton}
        with Glyph do
        begin
          ExtractIconEx(TextBuffer, 0, LargeIcon, SmallIcon, 1);

          {we must set the width and height of the glyph
           so it is large enough to display the small icon}
          Width  := GetSystemMetrics(SM_CXSMICON);
          Height := GetSystemMetrics(SM_CYSMICON);
          DrawIconEx(Glyph.Canvas.Handle, 0, 0, SmallIcon,
                     GetSystemMetrics(SM_CXSMICON),
                     GetSystemMetrics(SM_CYSMICON), 0, 0,
                     DI_NORMAL);

          DeleteObject(SmallIcon);
        end;
      end;

    {delete our filename text buffer}
    StrDispose(TextBuffer);
  end;

  {dispose of the memory allocated for the dropfile structure}
  DragFinish(Msg.Drop);
end;

procedure TForm1.SpeedButtonClick(Sender: TObject);
begin
  {when the button is pressed, the longhint portion of the hint
   contains the full path and filename of the executable. extract
   this path and filename and launch the file.}
  ShellExecute(Form1.Handle, 'open', PChar(GetLongHint(TControl
              (Sender).Hint)),
               nil,nil,SW_SHOWNORMAL);
end;

end.
```

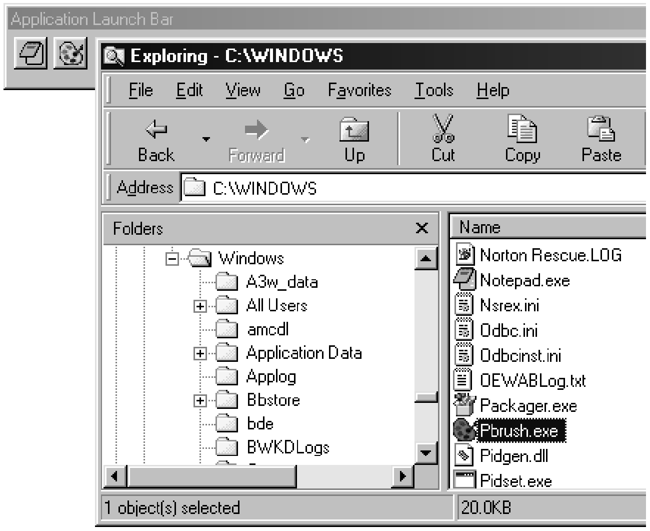


Figure 11-1: Files dropped on the launch bar

A file-based application may want to give the user the ability to go directly to a Windows Explorer window centered on the application's directory. With one line of code, the developer can do just this using the ShellExecute command. When ShellExecute "explores" a folder, it opens a new Windows Explorer window with the specified directory selected. The following example demonstrates this technique.

Listing 11-2: Exploring a folder

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  {open a new explorer window with the Common Files folder selected}
  ShellExecute(Form1.Handle, 'explore', 'C:\Program Files\Common Files',
               nil, nil, SW_SHOWNORMAL);
end;
```

Applications that produce temporary files should provide the functionality to delete these files without user intervention. A file-based application may also need to provide a mechanism allowing the user to copy, move, rename, or delete files within a controlled environment. The SHFileOperation function provides the ability to accomplish any file manipulation requirements. The following example demonstrates deleting a file to the recycle bin.

Listing 11-3: Deleting a file to the recycle bin

```
procedure TForm1.FileListBox1DblClick(Sender: TObject);
var
  FileOperation: TSHFileOpStruct;   // holds information on the file to delete
begin
  {initialize the TSHFileOpStruct with the necessary information. the FOF_ALLOWUNDO flag
   indicates the deleted file will go to the recycle bin}
  FileOperation.fFlags          := FOF_ALLOWUNDO or FOF_SIMPLEPROGRESS;
  FileOperation.Wnd             := Form1.Handle;
  FileOperation.wFunc           := FO_DELETE;
  FileOperation.pFrom           := PChar(FileListBox1.FileName);
  FileOperation.lpszProgressTitle := 'Deleting Files';

  {delete the specified file}
  SHFileOperation(FileOperation);

  {update the file list box information}
  FileListBox1.Update;
end;
```

Structured Storage

A structured storage file is a new, powerful technique for storing persistent application data. This method allows an application to easily store data in multiple different formats, while providing a non-linear method for accessing this information. For example, take an application that must store information in several different record structures. A typical approach would involve writing each record structure out to a stream, perhaps with some additional information about the size of each record structure. Reading this information back in would be a complicated matter, involving reading structure sizes and performing seeks to get to the desired information. Structured storage files solve this problem and have many other powerful benefits.

A structured storage file (sometimes known as a compound structured storage file) is a disk file that is managed via specific COM interfaces. Structured storage files consist of two types of objects: storages and streams. A storage object is a container object and can contain other storages or streams. A stream object contains actual data and may not contain other objects. The relationship between storages and streams is directly analogous to the relationship between directories and files; indeed, a structured storage file is simply a file system encapsulated into a single file. The storages and streams themselves are referred to by application-defined names, making the manipulation of structured storage file contents very easy compared to the complex seeking methods required of regular, linear files.

This structure has many benefits for an application. Because substorages and streams are segregated, it is very easy to find and manipulate the exact data an application needs without having to linearly read the entire file. Additionally, using structured storage files makes it much easier for future versions of the application to update their structure. It is also much easier to store a variety of data types within a structured storage file without the need to include additional information to identify the data within the file.

Creating and Reading Structured Storage Files

The StgCreateDocFile API function is used to create structured storage files. This creates an actual file and returns a pointer to an IStorage interface that represents the root storage (which can be thought of as a directory) of this file. Use the StgOpenStorage API function to open an existing storage file, which also returns an IStorage interface. The IStorage interface provides several methods for manipulating the structured storage file, including the creation of substorages and streams. When a stream is opened or created, the application receives an IStream interface, which includes additional methods for manipulating stream data. These methods are very similar to methods available to the TStream object (such as Read, Write, and Seek), making conversion to structured storage files easy if TStream objects are originally used in an application that creates files.

Note: While the interfaces for manipulating structured storage files are defined in the ActiveX.pas unit, the error messages returned by these methods are defined in the Windows.pas unit.

Transacted Storage Files

A structured storage file supports two modes of operation: direct and transacted. In the direct mode, any changes to substorages or streams are immediately written to disk. In transacted mode, these changes are tracked but are not written to disk unless explicitly instructed to by a call to the Commit method. Calling the Revert method discards all changes made to the structured storage file since the last call to the Commit method.

Note: The structured storage file implementation of streams (i.e., IStream) does not support opening streams in transacted mode. Therefore, the IStream.Commit and IStream.Revert methods are not discussed in this text. Additionally, range locking is not supported by this implementation, so the IStream.LockRegion and IStream.UnlockRegion methods are also not discussed in this text.

The following complex example provides a demonstration of what structured storage files can offer an application. Many of the IStorage and IStream methods are used, showcasing how these two interfaces interact with one another. The example is a structured storage file editor, allowing the user to both create new and manipulate existing structured storage files, including creating substorages and editing stream contents.

Listing 11-4: A structured storage file editor

```
implementation

uses
  PropsU, StreamEditU;

{*****************************************************************
  Note: This is not a polished application. This application is meant as a showcase for
  several of the IStorage and IStream methods, and as such, it is written to demonstrate
  their methods and how these methods work together, but it is not meant to be a
  production application. Writing larger examples in this manner results in additional
  code that is extraneous to the actual methods being demonstrated, resulting in an
  example that is often more complex than it should be to demonstrate the methods or
  functions involved.
  Additionally, there is very little error checking and exception handling, as
  such code would further complicate the example and distract from the methods
  in discussion. Therefore, while this application demonstrates a real-world
  example of how to use the IStorage and IStream methods, it is not complete,
  and would need to be fleshed out in order to make a more viable product.
 *******************************************************************************}

 {-----------------------}
 { >>> W A R N I N G <<< }
 {-----------------------}
 {Several professional applications use structured storage files for saving
  document data, including Microsoft Office products. YOU CAN SERIOUSLY
  DAMAGE OR CORRUPT Microsoft Office documents or any other type of file that
  uses structured storage by using this application. We highly recommend that
  you use this application only to create and test structured storage files or
  to edit structured storage files created with this application. Please do
  not use this application to view or modify any files that were not created
  with this application.}


const
  {these constants are not defined in Delphi 6}
  STG_S_MULTIPLEOPENS        = $00030204;
  STG_S_CONSOLIDATIONFAILED  = $00030205;
  STG_S_CANNOTCONSOLIDATE    = $00030206;
  STG_S_CONVERTED            = $00030200;
  STG_E_NOTSIMPLEFORMAT      = $80030112;


{ -- General Functions --}

{returns a date and time string based on an element date and time}
function GetElementDate(ElmTime: TFileTime): string;
var
  FileDateTime: Integer;
  LocalFileTime: TFileTime;
begin
  {convert the file time to a datetime value (uses a trick
   found in SysUtils.pas)}
  FileDateTime := 0;
  FileTimeToLocalFileTime(ElmTime, LocalFileTime);
  FileTimeToDosDateTime(LocalFileTime, LongRec(FileDateTime).Hi,
                        LongRec(FileDateTime).Lo);

  {convert the date and time into a string and return}
  if FileDateTime <> 0 then
    Result := DateToStr(FileDateToDateTime(FileDateTime))
  else
    Result := '<no time reported>';
end;

{clears the treeview}
procedure TfrmMain.Clear;
var
  iCount: Integer;
begin
  {free all objects}
  for iCount := 0 to treStorageTree.Items.Count - 1 do
    IUnknown(treStorageTree.Items[iCount].Data)._Release;
  {turn off the change event}
  treStorageTree.OnChange := nil;

  {clear the treeview}
  treStorageTree.Items.Clear;

  {turn changing back on}
  treStorageTree.OnChange := treStorageTreeChange;
end;

{commits or reverts changes to the structured storage file}
procedure TfrmMain.SaveFile;
var
  RootStore: IStorage;
begin
  {if the tree has items, then something has been opened}
  if treStorageTree.Items.Count > 0 then
    begin
      {retrieve a storage interface for the root storage}
      IUnknown(treStorageTree.Items[0].Data).QueryInterface(IStorage,
                                                           RootStore);

      {commit or revert changes as selected by the user}
      if MessageBox(Handle, 'Do you wish to commit changes to the structured
                    storage file?', 'Commit Changes', MB_ICONWARNING or
                    MB_YESNO) = IDYES then
        RootStore.Commit(STGC_DEFAULT)
      else
        RootStore.Revert;
    end;
end;


{ -- Form Event Handlers -- }

{called when the form is closed}
procedure TfrmMain.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  {save the file and clean up}
  SaveFile;
  Clear;
end;

{called when the user selects a tree item}
procedure TfrmMain.treStorageTreeChange(Sender: TObject; Node: TTreeNode);
var
  StorageElement: IUnknown;
  Unused: IStorage;
begin
  {get the stored interface}
  StorageElement := IUnknown(treStorageTree.Selected.Data);

  {enable view contents only if this is a stream}
  mnuViewContents.Enabled := StorageElement.QueryInterface(IStorage, Unused) <> S_OK;
end;
```

```
{called when the user renames a tree item}
procedure TfrmMain.treStorageTreeEdited(Sender: TObject; Node: TTreeNode;
  var S: String);
var
  StorageElement: IUnknown;           // holds a generic interface
  elmStore, SubStore: IStorage;       // holds IStorage interface
  elmStream, SubStream: IStream;      // holds IStream interface
  StatInfo: TStatStg;                 // statistic information record
begin
  {get the stored interface}
  StorageElement := IUnknown(Node.Data);

  {if this is a storage or stream element...}
  if StorageElement.QueryInterface(IStorage, elmStore) = S_OK then
  begin
    {get the statistics and release the object}
    elmStore.Stat(StatInfo, STATFLAG_DEFAULT);
    elmStore._Release;
    elmStore._Release;
  end
  else
  begin
    {get the statistics and release the object}
    StorageElement.QueryInterface(IStream, elmStream);
    elmStream.Stat(StatInfo, STATFLAG_DEFAULT);
    elmStream._Release;
    elmStream._Release;
  end;

  {the elements were released because the elements to be renamed must not
   be opened}

  {retrieve a pointer to the parent node}
  StorageElement := IUnknown(Node.Parent.Data);
  StorageElement.QueryInterface(IStorage, elmStore);

  {rename the element}
  elmStore.RenameElement(PWideChar(WideString(Node.Text)),
                       PWideChar(WideString(S)));

  {now that the element has been renamed, reopen the element and set up
   its treenode, so that other functions in this application will work}
  case StatInfo.dwType of
    STGTY_STORAGE : begin
                      elmStore.OpenStorage(PWideChar(WideString(S)), nil,
                                           STGM_READWRITE or STGM_SHARE_EXCLUSIVE,
                                           nil, 0, SubStore);
                      Node.Data := Pointer(SubStore);
                      SubStore._AddRef;
                    end;
    STGTY_STREAM  : begin
                      elmStore.OpenStream(PWideChar(WideString(S)), nil,
                                        STGM_READWRITE or STGM_SHARE_EXCLUSIVE,
                                          0, SubStream);
                      SubStream._AddRef;
                      SubStream._AddRef;
                    Node.Data := Pointer(SubStream);
                  end;
  end;
end;

{called when a tree item is being edited}
procedure TfrmMain.treStorageTreeEditing(Sender: TObject; Node: TTreeNode;
  var AllowEdit: Boolean);
begin
  {do not allow renaming of the parent node}
  AllowEdit := Node.Parent <> nil;
end;

{a recursive function called to enumerate a storage and populate the tree view}
procedure TfrmMain.EnumerateStorage(ParentNode: TTreeNode; Store: IStorage);
var
  SubStore: IStorage;             // an IStorage interface
  Stream: IStream;                // an IStream interface
  Enumerator: IEnumStatStg;       // the enumerator interface
  StatInfo: TStatStg;             // statistical information record
  SubNode: TTreeNode;             // tree node
begin
  {get the enumerator}
  Store.EnumElements(0, nil, 0, Enumerator);

  {begin enumerating the subelements in this storage}
  while Enumerator.Next(1, StatInfo, nil) = S_OK do
  begin
    {if this is a storage element...}
    if StatInfo.dwType = STGTY_STORAGE then
    begin
      {open the storage}
      Store.OpenStorage(StatInfo.pwcsName, nil, STGM_READWRITE or
                        STGM_SHARE_EXCLUSIVE, nil, 0, SubStore);

      {add one to the reference count, so it is not deleted when
       this function goes out of scope}
      SubStore._AddRef;

      {add a node to the treeview. it points to the storage object}
      SubNode := treStorageTree.Items.AddChildObject(ParentNode,
                                                     StatInfo.pwcsName,
                                                     Pointer(SubStore));

      {set appropriate image indexes}
      SubNode.ImageIndex := 0;
      SubNode.SelectedIndex := 0;

      {recursively enumerate this storage}
      EnumerateStorage(SubNode, SubStore);
    end
    else
    begin
      {open the stream object}
      Store.OpenStream(StatInfo.pwcsName, nil, STGM_READWRITE or
                       STGM_SHARE_EXCLUSIVE, 0, Stream);
      {add one to the reference count, so it is not deleted when
       this function goes out of scope}
      Stream._AddRef;

      {add a node to the treeview. it points to the storage object}
      SubNode := treStorageTree.Items.AddChildObject(ParentNode,
                                                     StatInfo.pwcsName,
                                                     Pointer(Stream));

      {set appropriate image indexes}
      SubNode.ImageIndex := 1;
      SubNode.SelectedIndex := 1;
    end;
  end;
end;


{ -- Menu Handlers -- }

procedure TfrmMain.mnuOpenClick(Sender: TObject);
var
  RootStore: IStorage;
  TopNode: TTreeNode;
begin
  {if the user selects a file...}
  if dlgOpen.Execute then
    {...and it is a storage file...}
    if StgIsStorageFile(PWideChar(WideString(dlgOpen.FileName))) = S_OK then
    begin
      {save the file and clean up}
      SaveFile;
      Clear;

      {open this structured storage file in a transacted mode}
      StgOpenStorage(PWideChar(WideString(dlgOpen.FileName)), nil,
                     STGM_READWRITE or STGM_SHARE_EXCLUSIVE or STGM_TRANSACTED,
                     nil, 0, RootStore);

      {add one to the reference count, so it is not deleted when
       this function goes out of scope}
      RootStore._AddRef;

      {add a node to the treeview, pointing at the storage object}
      TopNode := treStorageTree.Items.AddObject(nil,
                                             ExtractFileName(dlgOpen.FileName),
                                             Pointer(RootStore));

      {set the image index}
      TopNode.ImageIndex := 0;
      TopNode.SelectedIndex := 0;

      {enumerate this root storage object}
      EnumerateStorage(TopNode, RootStore);

      {expand the treeview}
      TopNode.Expand(TRUE);
      {enable the properties menu}
      mnuProperties.Enabled := TRUE;
    end
    else
      MessageBox(Handle, 'The selected file is not a structured storage file',
                 'Not a Structured Storage File', MB_ICONWARNING or MB_OK);
end;

procedure TfrmMain.mnuPropertiesClick(Sender: TObject);
var
  StatInfo: TStatStg;            // statistical information record
  StorageElement: IUnknown;      // general interface pointer
  elmStore: IStorage;            // an IStorage interface
  elmStream: IStream;            // an IStream interface
begin
  {retrieve the stored interface}
  StorageElement := IUnknown(treStorageTree.Selected.Data);

  {if this is a storage element...}
  if StorageElement.QueryInterface(IStorage, elmStore) = S_OK then
  begin
    {retrieve the statistical information}
    elmStore.Stat(StatInfo, STATFLAG_DEFAULT);

    {set form element properties appropriately}
    frmProps.lblType.Caption := 'Storage';
    frmProps.Image1.Visible := TRUE;
    frmProps.Image2.Visible := FALSE;
    frmProps.edtClassID.Enabled := TRUE;
  end
  else
  begin
    {the stream interface and retrieve statistical information}
    StorageElement.QueryInterface(IStream, elmStream);
    elmStream.Stat(StatInfo, STATFLAG_DEFAULT);

    {set form element properties appropriately}
    frmProps.lblType.Caption := 'Stream';
    frmProps.Image1.Visible := FALSE;
    frmProps.Image2.Visible := TRUE;
    frmProps.edtClassID.Enabled := FALSE;
  end;

  {set form properties appropriately}
  frmProps.lblName.Caption       := StatInfo.pwcsName;
  frmProps.lblSize.Caption       := IntToStr(StatInfo.cbSize) + ' bytes';
  frmProps.lblModTime.Caption    := GetElementDate(StatInfo.mtime);
  frmProps.lblAccessTime.Caption := GetElementDate(StatInfo.atime);
  frmProps.lblCreateTime.Caption := GetElementDate(StatInfo.ctime);
  frmProps.edtClassID.Text       := GUIDToString(StatInfo.clsid);

  {if the OK button is clicked...}
  if frmProps.ShowModal = mrOK then
    {set the class id, if it's a storage object}
    if frmProps.edtClassID.Enabled then
```

```
      elmStore.SetClass(StringToGUID(frmProps.edtClassID.Text));
end;

procedure TfrmMain.mnuDeleteElementClick(Sender: TObject);
var
  StorageElement: IUnknown;
  elmStore: IStorage;
begin
  {do not allow delete of the root element}
  if treStorageTree.Selected = treStorageTree.Items[0] then
    raise Exception.Create('Cannot delete the root.');

  {if the user really wants to delete this element...}
  if MessageBox(Handle, 'Are you sure you want to delete this element?',
                'Confirm Delete', MB_ICONWARNING or MB_YESNO) = IDYES then
  begin
    {get a pointer to the root storage element}
    StorageElement := IUnknown(treStorageTree.Selected.Parent.Data);
    StorageElement.QueryInterface(IStorage, elmStore);

    {release the element pointed to by the selected tree item}
    IUnknown(treStorageTree.Selected.Data)._Release;

    {destroy the selected element (must be done from its parent storage)}
    elmStore.DestroyElement(PWideChar(WideString(
                            treStorageTree.Selected.Text)));

    {delete the tree item}
    treStorageTree.Items.Delete(treStorageTree.Selected);
  end;
end;

procedure TfrmMain.mnuStreamContentsClick(Sender: TObject);
var
  StorageElement: IUnknown;       // generic interface pointer
  elmStream: IStream;             // an IStream interface
  StatInfo: TStatStg;             // statistical information record
  StreamBuffer: Pointer;          // buffer used to hold stream contents
  OutputString: string;           // string used to set buffer contents
  SeekPos: Largeint;              // seek position variable
  BytesReadWritten: Longint;      // bytes written variable
begin
  {determine if the selected element is a stream element}
  StorageElement := IUnknown(treStorageTree.Selected.Data);
  if StorageElement.QueryInterface(IStream, elmStream) <> S_OK then
    raise Exception.Create('Selected element is not a stream element.');

  {retrieve statistical information}
  elmStream.Stat(StatInfo, STATFLAG_NONAME);

  {allocate a buffer large enough to hold the entire stream contents}
  GetMem(StreamBuffer, StatInfo.cbSize);

  {retrieve the entire stream contents}
  elmStream.Seek(0, STREAM_SEEK_SET, SeekPos);
  elmStream.Read(StreamBuffer, StatInfo.cbSize, @BytesReadWritten);
  {display the stream contents}
  frmStreamEdit.memStream.Text := PChar(StreamBuffer);

  {free our memory buffer}
  FreeMem(StreamBuffer);

  {if the user clicks OK...}
  if frmStreamEdit.ShowModal = mrOK then
  begin
    {get the text to put into the stream}
    OutputString := frmStreamEdit.memStream.Text;

    {set the stream size because the text length could be smaller than original
     stream size}
    elmStream.SetSize(Length(frmStreamEdit.memStream.Text));

    {write the information starting at the front of the stream}
    elmStream.Seek(0, STREAM_SEEK_SET, SeekPos);
    elmStream.Write(PChar(OutputString), Length(frmStreamEdit.memStream.Text),
                    @BytesReadWritten);
  end;
end;

procedure TfrmMain.mnuStorageClick(Sender: TObject);
var
  StorageElement: IUnknown;           // general interface pointer
  elmStore, NewStore: IStorage;       // IStorage interfaces
  ParentNode, NewNode: TTreeNode;     // tree node pointers
  NewStoreName: string;               // string to hold new storage name
begin
  {retrieve the name for the new storage. Note: this should be a unique string}
  if not InputQuery('New Storage', 'Please specify a name for the new storage',
                    NewStoreName) then
    Exit;

  {get the appropriate parent node}
  if treStorageTree.Selected = treStorageTree.Items[0] then
    ParentNode := treStorageTree.Selected
  else
    ParentNode := treStorageTree.Selected.Parent;

  {retrieve a pointer to the storage element}
  StorageElement := IUnknown(ParentNode.Data);
  StorageElement.QueryInterface(IStorage, elmStore);

  {create a substorage element}
  elmStore.CreateStorage(PWideChar(WideString(NewStoreName)), STGM_READWRITE or
                         STGM_SHARE_EXCLUSIVE, 0, 0, NewStore);

  {add one to the reference count, so it is not deleted when
   this function goes out of scope}
  NewStore._AddRef;

  {add the tree node}
  NewNode := treStorageTree.Items.AddChildObjectFirst(ParentNode,
                                                       ExtractFileName(NewStoreName),
                                                Pointer(NewStore));

  {set appropriate image indexes}
  NewNode.ImageIndex := 0;
  NewNode.SelectedIndex := 0;
end;

procedure TfrmMain.mnuStreamClick(Sender: TObject);
var
  StorageElement: IUnknown;         // general interface pointer
  elmStore: IStorage;               // an IStorage interface
  NewStream: IStream;               // an IStream interface
  ParentNode, NewNode: TTreeNode;   // tree node pointers
  NewStreamName: string;            // holds the new stream name
begin
  {retrieve the name of the new stream. note: this should be a unique name}
  if not InputQuery('New Stream', 'Please specify a name for the new stream',
                    NewStreamName) then
    Exit;

  {retrieve the parent node}
  ParentNode := treStorageTree.Selected;

  {retrieve the storage element in which this stream will reside}
  StorageElement := IUnknown(ParentNode.Data);
  StorageElement.QueryInterface(IStorage, elmStore);

  {retrieve the parent storage, if the selected node points to a
   stream element}
  if elmStore = nil then
  begin
    ParentNode := treStorageTree.Selected.Parent;
    StorageElement := IUnknown(ParentNode.Data);
    StorageElement.QueryInterface(IStorage, elmStore);
  end;

  {create the stream within this storage element}
  elmStore.CreateStream(PWideChar(WideString(NewStreamName)), STGM_READWRITE or
                        STGM_SHARE_EXCLUSIVE, 0, 0, NewStream);

  {add one to the reference count, so it is not deleted when
   this function goes out of scope}
  NewStream._AddRef;

  {add the tree node}
  NewNode := treStorageTree.Items.AddChildObjectFirst(ParentNode,
                                                ExtractFileName(NewStreamName),
                                                Pointer(NewStream));

  {set appropriate image indexes}
  NewNode.ImageIndex := 1;
  NewNode.SelectedIndex := 1;
end;

procedure TfrmMain.mnuStorageFileClick(Sender: TObject);
var
```

```
  RootStore: IStorage;
  TopNode: TTreeNode;
begin
  {if the user specifies a new file name...}
  if dlgSave.Execute then
  begin
    {clean up}
    Clear;

    {create a new structured storage file in transacted mode}
    StgCreateDocFile(PWideChar(WideString(dlgSave.FileName)), STGM_READWRITE or
                     STGM_SHARE_EXCLUSIVE or STGM_TRANSACTED, 0, RootStore);

    {add one to the reference count, so it is not deleted when
     this function goes out of scope}
    RootStore._AddRef;

    {add the tree node}
    TopNode := treStorageTree.Items.AddObject(nil,
                                         ExtractFileName(dlgSave.FileName),
                                         Pointer(RootStore));

    {set appropriate image indexes}
    TopNode.ImageIndex := 0;
    TopNode.SelectedIndex := 0;

    {enable the properties menu}
    mnuProperties.Enabled := TRUE;
  end;
end;

procedure TfrmMain.mnuCopyStreamClick(Sender: TObject);
var
  StorageElement: IUnknown;                 // general interface pointer
  elmStore: IStorage;                       // an IStorage interface
  elmStream, NewStream: IStream;            // IStream interfaces
  ParentNode, NewNode: TTreeNode;           // tree node pointers
  NewSeekPos, cbRead, cbWritten: Largeint;  // variables to hold various values
begin
  {retrieve a pointer to the stream element interface}
  StorageElement := IUnknown(treStorageTree.Selected.Data);
  StorageElement.QueryInterface(IStream, elmStream);

  {if this is a storage element, exit}
  if elmStream = nil then
    Exit;

  {retrieve a pointer to the selected storage element}
  ParentNode := treStorageTree.Selected;
  StorageElement := IUnknown(ParentNode.Data);
  StorageElement.QueryInterface(IStorage, elmStore);

  if elmStore = nil then
  begin
    ParentNode := treStorageTree.Selected.Parent;
    StorageElement := IUnknown(ParentNode.Data);
    StorageElement.QueryInterface(IStorage, elmStore);
  end;

  {create a sibling stream in the same storage as the stream to copy}
  elmStore.CreateStream(PWideChar(WideString('CopyOf' +
                        treStorageTree.Selected.Text)), STGM_READWRITE or
                        STGM_SHARE_EXCLUSIVE, 0, 0, NewStream);

  {add one to the reference count, so it is not deleted when
   this function goes out of scope}
  NewStream._AddRef;

  {add a new tree node}
  NewNode := treStorageTree.Items.AddChildObjectFirst(ParentNode,
                                                 ExtractFileName('CopyOf'+
                                                 treStorageTree.Selected.Text),
                                                 Pointer(NewStream));

  {set appropriate image indexes}
  NewNode.ImageIndex := 1;
  NewNode.SelectedIndex := 1;

  {copy the entire stream}
  elmStream.Seek(0, STREAM_SEEK_SET, NewSeekPos);
  elmStream.CopyTo(NewStream, MAXINT, cbRead, cbWritten);
end;
```

Delphi vs. the Windows API

The functions discussed in this chapter provide functionality to the Delphi programmer that is simply not available when using the VCL alone. Very little, if any, of these API functions are encapsulated in a library function or VCL object. Unfortunately, some of these API functions are somewhat complex, such as the structured storage interfaces and methods, but they provide powerful abilities to applications that exploit their functionality.

Shell File Functions

The following shell file functions are covered in this chapter.

Table 11-1: Shell file functions

Function	Description
DragAcceptFiles	Registers a window as a drop target for dragged files.
DragFinish	Completes the file drag-and-drop process.
DragQueryFile	Retrieves information about a dropped file.
DragQueryPoint	Retrieves the mouse coordinates at the time of a file drop.
FindExecutable	Retrieves the name of the executable file associated with a specified file.
IStorage.Commit	Commits changes to a transacted storage object.

Function	Description
IStorage.CopyTo	Copies one storage object into another.
IStorage.CreateStorage	Creates a substorage element.
IStorage.CreateStream	Creates a stream element.
IStorage.DestroyElement	Destroys a substorage or stream element.
IStorage.EnumElements	Enumerates all elements in a storage.
IStorage.MoveElementTo	Moves an element from one storage object into another.
IStorage.OpenStorage	Opens a substorage element.
IStorage.OpenStream	Opens a stream element.
IStorage.RenameElement	Renames a substorage or stream element.
IStorage.Revert	Reverts a transacted storage object.
IStorage.SetClass	Sets the class identifier of a storage object.
IStorage.Stat	Retrieves statistical information on the storage object.
IStream.Clone	Creates a copy of the stream with its own seek position.
IStream.CopyTo	Copies the stream into another stream.
IStream.Read	Reads bytes from the stream into a buffer.
IStream.Seek	Moves the seek position within a stream.
IStream.SetSize	Sets the size of the stream.
IStream.Stat	Retrieves statistical information on the stream object.
IStream.Write	Writes bytes from a buffer into the stream.
SHAddToRecentDocs	Adds or clears a registered document type to the Documents menu item under the Start button.
SHFileOperation	Copies, moves, renames, or deletes a file.
SHFreeNameMappings	Frees a name mapping object.
SHGetFileInfo	Retrieves information about the specified file.
StgCreateDocFile	Creates a compound structured storage file.
StgIsStorageFile	Indicates if the specified file is a compound structured storage file.
StgOpenStorage	Opens a compound structured storage file.

DragAcceptFiles ShellAPI.pas

Syntax

```
DragAcceptFiles(
Wnd: HWND;              {a handle to a window}
Accept: BOOL            {the acceptance flag}
);                      {this procedure does not return a value}
```

Description

This procedure registers a window to accept or decline dropped files. If an application registers a window to accept dropped files, it receives a WM_DROPFILES message when files are dragged and dropped onto the window.

Parameters

Wnd: A handle to the window that will accept or decline dropped files.

Accept: A Boolean value that determines if the window will accept or decline dropped files. A value of TRUE registers the window as accepting dropped files; a value of FALSE will decline dropped files.

See Also

DragFinish, DragQueryFile, DragQueryPoint, WM_DROPFILES

Example

Listing 11-5: Retrieving information on dropped files

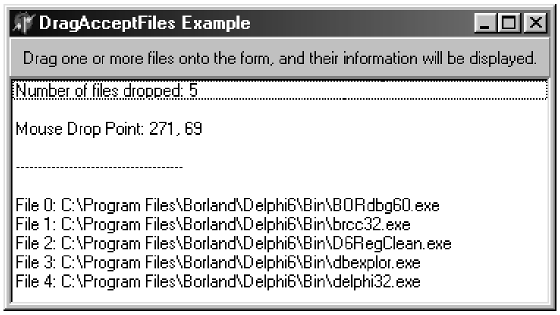
Note that this example requires this line to be added to the public section of the form's class definition:

```
procedure WMDropFiles(var DropFileMsg: TWMDropFiles); message WM_DROPFILES;
```

When a file is dropped onto the form, Windows sends the form a WM_DROPFILES message. This line declares a procedure that will handle this message.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   {this registers the window to accept files}
   DragAcceptFiles(Handle,TRUE);
end;

procedure TForm1.WMDropFiles(var DropFileMsg: TWMDropFiles);
var
   FileCount: Integer;                  // holds the number of files dropped
   TheFileName: array[0..500] of char;  // holds a filename
   DropPoint: TPoint;                   // holds drop point coordinates
   LoopCount: Integer;                  // a general loop count variable
begin
   {clear our list box that displays file information}
   ListBox1.Items.Clear;

   {get the number of files that were dropped and display it}
   FileCount:=DragQueryFile(DropFileMsg.Drop,$FFFFFFFF,nil,0);
   ListBox1.Items.Add('Number of files dropped: '+IntToStr(FileCount));
   ListBox1.Items.Add('');

   {get the coordinates relative to the window where the files were dropped}
   DragQueryPoint(DropFileMsg.Drop, DropPoint);
   ListBox1.Items.Add('Mouse Drop Point: '+IntToStr(DropPoint.X)+', '+
                      IntToStr(DropPoint.Y));
   ListBox1.Items.Add('');
   ListBox1.Items.Add('--------------------------------------');
   ListBox1.Items.Add('');

   {retrieve the full path and filename of each file that was dropped}
   for LoopCount:=0 to FileCount-1 do
   begin
      DragQueryFile(DropFileMsg.Drop, LoopCount, TheFileName, 500);
```

```
        ListBox1.Items.Add('File '+IntToStr(LoopCount)+': '+string(TheFileName));
      end;

      {release the memory that was allocated
       for the file drop information structure}
      DragFinish(DropFileMsg.Drop);
   end;
```



Drag one or more files onto the form, and their information will be displayed.

```
Number of files dropped: 5

Mouse Drop Point: 271, 69

------------------------------------

File 0: C:\Program Files\Borland\Delphi6\Bin\BORdbg60.exe
File 1: C:\Program Files\Borland\Delphi6\Bin\brcc32.exe
File 2: C:\Program Files\Borland\Delphi6\Bin\D6RegClean.exe
File 3: C:\Program Files\Borland\Delphi6\Bin\dbexplor.exe
File 4: C:\Program Files\Borland\Delphi6\Bin\delphi32.exe
```

Figure 11-2: Dropped file information

DragFinish ShellAPI.pas

Syntax

DragFinish(	
Drop: HDROP	{a handle to a file drop information structure}
);	{this procedure does not return a value}

Description

This procedure frees memory that Windows allocated for the data structure holding dropped file information.

Parameters

Drop: A handle to the dropped file information data structure. This handle is passed in the wParam member of the WM_DROPFILES message. This is also accessible from Delphi as the Drop member of the TWMDropFiles message structure passed to the WM_DROPFILES message handling routine.

See Also

DragAcceptFiles, WM_DROPFILES

Example

Please see Listing 11-5 under DragAcceptFiles.

DragQueryFile ShellAPI.pas

Syntax

DragQueryFile(	
Drop: HDROP;	{a handle to a file drop information structure}
FileIndex: UINT;	{the index to a filename}

```
FileName: PChar;        {a pointer to a buffer to hold a filename}
cb: UINT                {the size of the filename buffer}
): UINT;                {returns an unsigned integer based on the parameters}
```

Description

This function retrieves the filename of a dropped file. The FileIndex parameter indicates the position of the dropped file in the list of dropped files identified by the Drop parameter for which to retrieve the filename. The full path and filename of the dropped file is stored in the buffer pointed to by the FileName parameter.

Parameters

Drop: A handle to the dropped file information data structure. This handle is passed in the wParam member of the WM_DROPFILES message. This is also accessible from Delphi as the Drop member of the TWMDropFiles message structure passed to the WM_DROPFILES message handling routine.

FileIndex: This identifies the index of the dropped file. If this parameter is $FFFFFFFF, this function returns the total number of files dropped. The array of files is zero-based, and a value between zero and the total number of files dropped will cause the full path and filename of the corresponding file to be copied into the buffer pointed to by the FileName parameter.

FileName: This points to a buffer that receives the filename of a dropped file. If this parameter is NIL, this function returns the required size of the buffer, in characters.

cb: Specifies the size of the buffer pointed to by the FileName parameter, in characters.

Return Value

If this function succeeds, the return value is dependent on the values passed in the parameters. If the value of the FileIndex parameter is $FFFFFFFF, this function returns the total number of files dropped. If the FileIndex parameter is between zero and the total number of files dropped, and the value of the FileName parameter is NIL, this function returns the size of the buffer required to hold the full path and filename of the corresponding file, in characters. This does not include the null terminating character. If this function copies a filename to the buffer pointed to by the FileName parameter, it returns the total number of characters copied, excluding the null terminating character. If the function fails, it returns zero.

See Also

DragAcceptFiles, DragQueryPoint, WM_DROPFILES

Example

Please see Listing 11-5 under DragAcceptFiles.

DragQueryPoint ShellAPI.pas

Syntax

```
DragQueryPoint(
Drop: HDROP;               {a handle to a file drop information structure}
var Point: TPoint          {a pointer to a structure for coordinate information}
): BOOL;                   {returns TRUE or FALSE}
```

Description

This function fills a TPoint structure with the coordinates of the mouse cursor at the time that files were dropped onto the window.

Parameters

Drop: A handle to the dropped file information data structure. This handle is passed in the wParam member of the WM_DROPFILES message. This is also accessible from Delphi as the Drop member of the TWMDropFiles message structure passed to the WM_DROPFILES message handling routine.

Point: A pointer to a TPoint structure that will be filled with the X and Y coordinates of the mouse cursor at the time the files were dropped. These coordinates are relative to the window in which the drop occurred.

Return Value

If this function succeeds and the drop point was within the client area of the window, it returns TRUE. If the function fails, or if the drop point was not within the client area, it returns FALSE.

See Also

DragAcceptFiles, DragQueryFile, WM_DROPFILES

Example

Please see Listing 11-5 under DragAcceptFiles.

FindExecutable ShellAPI.pas

Syntax

```
FindExecutable(
FileName: PChar;           {a pointer to a filename string}
Directory: PChar;          {a pointer to a default directory string}
Result: PChar              {a pointer to a buffer that receives a filename}
): HINST;                  {returns an integer value}
```

Description

This function retrieves the name and path of the executable file associated with the filename passed in the FileName parameter.

Parameters

FileName: A pointer to a null-terminated string that contains a filename. This parameter can specify either a document or an executable file. If this parameter contains the name of an executable file, the Result parameter will contain an exact copy of this parameter when the function returns.

Directory: A pointer to a null-terminated string specifying a path to use as the default directory.

Result: A pointer to a buffer that receives a null-terminated string identifying the executable file associated with the file or document indicated by the FileName parameter. This executable file is launched when an "open" action is performed on the file specified by the FileName parameter, either by right-clicking on the file and selecting Open in the Windows Explorer or using the ShellExecute function. The registry records which executable file is associated with specific file types. When the FindExecutable function is called, it first looks in the registry under the key HKEY_LOCAL_MACHINE\SOFTWARE\Classes\<file type>. The value at this location is the name of another key. FindExecutable then takes this value and looks under the key HKEY_LOCAL_MACHINE\SOFTWARE\Classes\<key name>\Shell\Open\Command. The value at this location contains the full path and filename of the associated executable. For example, if the FileName parameter specified a file with the extension .pas, FindExecutable would first look under HKEY_LOCAL_MACHINE\ SOFTWARE\Classes\.pas. The value at this location is "DelphiUnit." FindExecutable takes this value and looks under the key HKEY_LOCAL_MACHINE\SOFTWARE\Classes\DelphiUnit\Shell\Open\Command. If Delphi has been installed into the default location, the value found at this location will be "C:\Program Files\Borland\Delphi\Bin\Delphi32.EXE" "%1".

Note: If the path and filename of the executable file stored in this registry key contains spaces, as in the following example, and it is not surrounded by double quotes, the FindExecutable function replaces any spaces in the value with a null character.

Return Value

If the function succeeds, it returns a value greater than 32. If the function fails, it returns a value from Table 11-2.

See Also

ShellExecute, ShellExecuteEx

Example

Listing 11-6: Finding an executable file and opening documents

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
   {set a new edit mask based on the contents of Edit1}
   if ((Key = Chr(13)) AND (Edit1.Text <> '')) then
   begin
      FileListBox1.Mask := '*.'+Edit1.Text;
      {this prevents the speaker from beeping}
      Key := #0;
   end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
   {launch the executable file found by FindExecutable}
   if Label1.Caption <> '' then
      ShellExecute(Form1.Handle, 'open', PChar(Label1.Caption),
                   nil, nil, SW_SHOWNORMAL);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
   {open the selected file by starting its associated application}
   ShellExecute(Form1.Handle, 'open', PChar(FileListBox1.Filename),
                nil, nil, SW_SHOWNORMAL);
end;

procedure TForm1.FileListBox1Click(Sender: TObject);
var
   Buffer: array[0..500] of char;   // a buffer for a path and filename
begin
   {find the executable associated with the selected file}
   FindExecutable(PChar(FileListBox1.FileName),nil,@Buffer);

   {if an executable was not found...}
   if StrLen(Buffer)<1 then
   begin
      {...display a message and disable the launch buttons...}
      Label1.Caption:='No Associated executable';
      Button1.Enabled:=FALSE;
      Button2.Enabled:=FALSE;
   end
   else
   begin
      {...otherwise display the executable path and filename}
      Label1.Caption:=string(Buffer);
      Button1.Enabled:=TRUE;
      Button2.Enabled:=TRUE;
   end;
end;
```

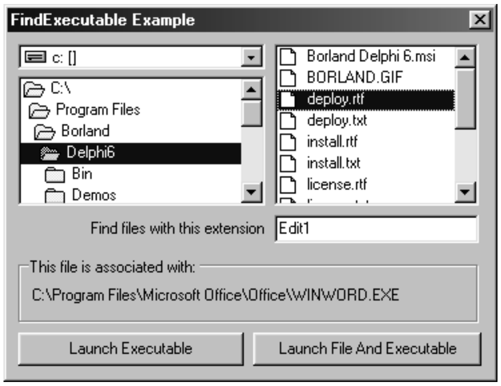


Figure 11-3: Found an executable file

Table 11-2: FindExecutable return value error codes

Value	Description
0	Not enough memory or resources.
ERROR_GEN_FAILURE	The specified file does not have an associated executable file.
ERROR_FILE_NOT_FOUND	The file specified by the FileName parameter could not be found.
ERROR_PATH_NOT_FOUND	The default directory path specified by the Directory parameter could not be found.
ERROR_BAD_FORMAT	The associated executable file is invalid or corrupt.

IStorage.Commit ActiveX.pas

Syntax

```
Commit(
grfCommitFlags: Longint     {commit flag}
): HResult;                 {returns an OLE result}
```

Description

This function processes any changes that have been made to a storage object opened in transact mode, ensuring that they are written to disk. This method has no effect if called on a substorage opened in direct mode.

Parameters

grfCommitFlags: A flag that controls how changes are committed. This can be one value from Table 11-3.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from Table 11-4.

See Also

IStorage.Revert, StgCreateDocFile, StgOpenStorage

Example

Please see Listing 11-4 in the introduction.

Table 11-3: IStorage.Commit grfCommitFlags values

Value	Description
STGC_DEFAULT	Indicates changes should be written in the default manner.
STGC_OVERWRITE	Indicates that the commit operation can overwrite existing data to reduce overall space requirements. However, this is not a safe flag to use in most instances, as it is possible for the commit operation to fail before all changes are made, resulting in a corrupted file. This value does cause the system to check for adequate space before writing.
STGC_ONLYIFCURRENT	Commits changes to the object only if there have been no changes to it since the calling process opened it. If another process has opened the storage object and committed changes, this flag causes the function to fail.
STGC_DANGEROUSLYCOMMIT-MERELYTODISKCACHE	Commits changes to a write-behind disk cache but does not write the cache to disk. The cache is typically written to disk some time after the commit operation returns.
STGC_CONSOLIDATE	**Windows 2000 and later**: Consolidates and defragments a storage file after changes are committed. This flag is valid only for root storages open in transact mode and has no effect on substorages. This flag can be combined with any other flag in this table.

Table 11-4: IStorage.Commit return value failure codes

Value	Description
STG_S_MULTIPLEOPENS	Changes were committed, but the storage file could not be consolidated because it was opened multiple times with the STGM_NOSNAPSHOT flag specified.
STG_S_CANNOTCONSOLIDATE	Changes were committed, but the storage file could not be consolidated because the storage was opened in an incorrect mode.
STG_S_CONSOLIDATIONFAILED	Changes were committed, but the storage file could not be consolidated because of an internal error.
E_PENDING	There is a pending commitment of data, some or all of which may not be available. This is used for asynchronous storages only.
STG_E_INVALIDFLAG or STG_E_INVALIDPARAMETER	The value for the grfCommitFlags parameter is not valid.
STG_E_NOTCURRENT	Indicates that another instance of the storage object is open, and the other instance has committed changes. Committing changes from the current instance will overwrite any previous changes.
STG_E_MEDIUMFULL	There is not enough space left on the storage device to write changes.
STG_E_TOOMANYOPENFILES	Indicates that there are too many open files.
STG_E_REVERTED	Indicates that the storage object has been reverted.

IStorage.CopyTo ActiveX.pas

Syntax

CopyTo(	
ciidExclude: Longint;	{number of exclude elements}
rgiidExclude: PIID;	{array of interface identifiers}
snbExclude: TSNB;	{string name block}
const stgDest: IStorage	{destination storage interface}
): HResult;	{returns on OLE result}

Description

This method copies the contents of one storage into another. The copy process is recursive, and will copy all substorages and streams unless otherwise excluded. The information from the source storage is merged with that of the destination storage. Any streams in the destination storage that have the same name as streams in the source storage will be copied over. However, substorages with the same name will not be replaced in this manner; any information in the substorage of the source is copied into the storage of the same name in the destination.

Parameters

ciidExclude: Indicates the number of elements in the array pointed to by the rgiidExclude parameter. If rgiidExclude is set to NIL, this parameter is ignored.

rgiidExclude: An array of interface identifiers that indicate which interfaces should be excluded from the copy operation. For example, including IID_IStorage in this array will cause the operation to copy everything but IStorage objects.

snbExclude: A null-terminated string that specifies a block of storage or stream objects to be excluded from the copy operation. If IID_IStorage is included in the array pointed to by the rgiidExclude parameter, this parameter is ignored.

stgDest: A pointer to the IStorage interface of the destination storage object to which information is copied.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from the following table.

See Also

IStorage.MoveElementTo, IStorage.Revert, IStream.CopyTo, StgCreateDocFile, StgOpenStorage

Example

Listing II-7: Copying a structured storage file

```
procedure TfrmMain.Button1Click(Sender: TObject);
var
  OrgStore, DestStore: IStorage;
begin
```

```
  {if the user selects a file...}
  if dlgOpen.Execute then
    {...and it is a storage file...}
    if StgIsStorageFile(PWideChar(WideString(dlgOpen.FileName))) = S_OK then
    begin
      {open this structured storage file}
      StgOpenStorage(PWideChar(WideString(dlgOpen.FileName)), nil,
                     STGM_READWRITE or STGM_SHARE_EXCLUSIVE,
                     nil, 0, OrgStore);

      {create a new structured storage file}
      StgCreateDocFile(PWideChar(WideString(ExtractFilePath(dlgOpen.FileName) +
                       'CopyOf' + ExtractFileName(dlgOpen.FileName))),
                       STGM_READWRITE or STGM_SHARE_EXCLUSIVE,
                       0, DestStore);

      {copy the file to the destination}
      OrgStore.CopyTo(0, nil, nil, DestStore);

      {the new file may be smaller than the original due to consolidation}
      ShowMessage('Copy completed, open and view the copied file in the structured storage
                   file editor');
    end
    else
      MessageBox(Handle, 'The selected file is not a structured storage file',
                 'Not a Structured Storage File', MB_ICONWARNING or MB_OK);
end;
```

Table 11-5: IStorage.CopyTo return values

Value	Description
E_PENDING	There is a pending commitment of data, some or all of which may not be available. This is used for asynchronous storages only.
STG_E_ACCESSDENIED	Indicates that the destination storage object is a child of the source storage object or the application does not have permission to access the destination file.
STG_E_INSUFFICIENTMEMORY	Operation failed due to insufficient available memory.
STG_E_INVALIDPOINTER	The destination storage object pointer is invalid.
STG_E_INVALIDPARAMETER	The value of one of the parameters is invalid.
STG_E_MEDIUMFULL	There is not enough space left on the storage device to write changes.
STG_E_TOOMANYOPENFILES	Indicates that there are too many open files.
STG_E_REVERTED	Indicates that the storage object has been reverted.

IStorage.CreateStorage ActiveX.pas

Syntax

CreateStorage(	
pwcsName: POleStr;	{the storage name}
grfMode: Longint;	{access mode flags}

```
dwStgFmt: Longint;          {reserved, set to zero}
reserved2: Longint;         {reserved, set to zero}
out stg: IStorage           {pointer to an IStorage interface}
): HResult;                 {returns an OLE result}
```

Description

This method creates and opens a new substorage within the source storage object.

Parameters

pwcsName: A null-terminated string indicating the name of the newly created storage object. If a storage with this name already exists and the STGM_CREATE flag is specified in grfMode, the existing storage is deleted and a new one created in its place. If the STGM_CONVERT flag is specified in grfMode, the existing storage is converted into a stream named "CONTENTS" and placed into the new storage.

grfMode: Flags indicating the access mode for the new storage. This parameter can be one or more values from Table 11-6.

dwStgFmt: Reserved; must be set to zero.

reserved2: Reserved; must be set to zero.

stg: When the function returns, this parameter will point to a valid IStorage interface for the newly created storage object.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from Table 11-7 and the stg parameter returns NIL.

See Also

IStorage.CopyTo, StgCreateDocFile

Example

Please see Listing 11-4 under StgCreateDocFile.

Table 11-6: IStorage.CreateStorage grfMode values

Value	Description
STGM_READ	Indicates that the object's data can be accessed but not modified. Cannot be combined with STGM_WRITE or STGM_READWRITE.
STGM_WRITE	Indicates that the object's data can be modified but not accessed. Cannot be combined with STGM_READ or STGM_READWRITE.
STGM_READWRITE	Indicates that the object's data can be both accessed and modified. Cannot be combined with STGM_READ or STGM_WRITE.

Value	Description
STGM_SHARE_DENY_NONE	Indicates that other processes are not denied read or write access to the object when opened. This is the default behavior. Cannot be combined with other STGM_SHAREXXX flags.
STGM_SHARE_DENY_READ	Indicates that other processes cannot open the object with read access. Cannot be combined with other STGM_SHAREXXX flags.
STGM_SHARE_DENY_WRITE	Indicates that other processes cannot open the object with write access. Cannot be combined with other STGM_SHAREXXX flags.
STGM_SHARE_EXCLUSIVE	Indicates that other processes cannot open the object in any mode. Cannot be combined with other STGM_SHAREXXX flags.
STGM_PRIORITY	Provides exclusive access to the most recently committed version of the object and prevents other processes from committing changes to the object while it is opened in priority mode. The STGM_DIRECT and STGM_READ flags must be used with this flag.
STGM_CREATE	Indicates that the specified object should be created. Cannot be used with STGM_CONVERT.
STGM_CONVERT	Creates the specified object and copies any existing data into a stream named CONTENTS. Cannot be used with STGM_CREATE.
STGM_FAILIFTHERE	Causes the function to fail if the specified object already exists. This is the default behavior.
STGM_DIRECT	Causes any changes to be written as they occur. Cannot be combined with STGM_TRANSACTED.
STGM_TRANSACTED	Any changes to the object are buffered and are only written when the Commit method is called. Call the Revert method to discard any changes since the last call to Commit. Cannot be combined with STGM_DIRECT.
STGM_NOSCRATCH	Must be used with STGM_TRANSACTED. This causes the system to use any unused areas of the original file as a "scratch" storage space for uncommitted changes. Without this flag, a temporary file is created to store uncommitted changes.
STGM_NOSNAPSHOT	Any changes to the file are written to the end of the file, instead of making a temporary copy of the file. When the file is opened using this flag, no other process can open the file without also using this flag. This flag can only be used in combination with STGM_TRANSACTED, and only if the STGM_SHARE_EXCLUSIVE and STGM_SHARE_DENY_WRITE flags are not specified. This can lead to very large files.

Value	Description
STGM_SIMPLE	Creates a simple compound structured storage file. It offers efficient performance but does not support substorages, and all streams are a minimum of 4Kb in size.
STGM_DIRECT_SWMR	Provides direct mode for single-writer, multi-reader operations.

Table 11-7: IStorage.CreateStorage return values

Value	Description
E_PENDING	There is a pending commitment of data, some or all of which may not be available. This is used for asynchronous storages only.
STG_E_ACCESSDENIED	Indicates the application does not have permission to access the destination file.
STG_E_FILEALREADYEXISTS	Indicates that a storage of the specified name already exists. This is returned as a result of using the STGM_FAILIFTHERE flag.
STG_E_INSUFFICIENTMEMORY	Operation failed due to insufficient available memory.
STG_E_INVALIDFLAG	The value of the grfMode parameter is not valid.
STG_E_INVALIDFUNCTION	The combination of values specified in grfMode is invalid.
STG_E_INVALIDNAME	The value specified in the pwcsName parameter is invalid.
STG_E_INVALIDPOINTER	The destination storage object pointer is invalid.
STG_E_INVALIDPARAMETER	The value of one of the parameters is invalid.
STG_E_MEDIUMFULL	There is not enough space left on the storage device to write changes.
STG_E_TOOMANYOPENFILES	Indicates that there are too many open files.
STG_S_CONVERTED	Indicates that an existing storage with the same name has been converted into a stream called CONTENTS.

IStorage.CreateStream ActiveX.pas

Syntax

```
CreateStream(
pwcsName: POleStr;          {the stream name}
grfMode: Longint;           {access mode flags}
reserved1: Longint;         {reserved, set to zero}
reserved2: Longint;         {reserved, set to zero}
out stm: IStream            {pointer to an IStream interface}
): HResult;                 {returns an OLE result}
```

Description

This method creates and opens a new stream within the source storage object.

Parameters

pwcsName: A null-terminated string indicating the name of the newly created stream.

grfMode: Flags indicating the access mode for the new stream. This parameter can be one or more values from Table 11-8.

reserved1: Reserved; must be set to zero.

reserved2: Reserved; must be set to zero.

stm: When the function returns, this parameter will point to a valid IStream interface for the newly created stream object.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from Table 11-9 and the stm parameter returns NIL.

See Also

IStorage.OpenStream, IStream.Clone, IStream.CopyTo

Example

Please see Listing 11-13 under StgCreateDocFile.

Table 11-8: IStorage.CreateStream grfMode values

Value	Description
STGM_READ	Indicates that the object's data can be accessed but not modified. Cannot be combined with STGM_WRITE or STGM_READWRITE.
STGM_WRITE	Indicates that the object's data can be modified but not accessed. Cannot be combined with STGM_READ or STGM_READWRITE.
STGM_READWRITE	Indicates that the object's data can be both accessed and modified. Cannot be combined with STGM_READ or STGM_WRITE.
STGM_SHARE_DENY_NONE	Indicates that other processes are not denied read or write access to the object when opened. This is the default behavior. Cannot be combined with other STGM_SHAREXXX flags.
STGM_SHARE_DENY_READ	Indicates that other processes cannot open the object with read access. Cannot be combined with other STGM_SHAREXXX flags.
STGM_SHARE_DENY_WRITE	Indicates that other processes cannot open the object with write access. Cannot be combined with other STGM_SHAREXXX flags.
STGM_SHARE_EXCLUSIVE	Indicates that other processes cannot open the object in any mode. Cannot be combined with other STGM_SHAREXXX flags.

Value	Description
STGM_PRIORITY	Provides exclusive access to the most recently committed version of the object, and prevents other processes from committing changes to the object while it is opened in priority mode. The STGM_DIRECT and STGM_READ flags must be used with this flag.
STGM_CREATE	Indicates that the specified object should be created.
STGM_FAILIFTHERE	Causes the function to fail if the specified object already exists. This is the default behavior.

Table 11-9: IStorage.CreateStream return values

Value	Description
E_PENDING	There is a pending commitment of data, some or all of which may not be available. This is used for asynchronous storages only.
STG_E_ACCESSDENIED	Indicates the application does not have permission to access the destination file.
STG_E_FILEALREADYEXISTS	Indicates that a stream of the specified name already exists. This is returned as a result of using the STGM_FAILIFTHERE flag.
STG_E_INSUFFICIENTMEMORY	Operation failed due to insufficient available memory.
STG_E_INVALIDFLAG	The value of the grfMode parameter is not valid.
STG_E_INVALIDFUNCTION	The combination of values specified in grfMode is invalid.
STG_E_INVALIDNAME	The value specified in the pwcsName parameter is invalid.
STG_E_INVALIDPOINTER	The destination stream object pointer is invalid.
STG_E_INVALIDPARAMETER	The value of one of the parameters is invalid.
STG_E_REVERTED	Indicates that the storage object has been reverted.
STG_E_TOOMANYOPENFILES	Indicates that there are too many open files.

IStorage.DestroyElement *ActiveX.pas*

Syntax

```
DestroyElement(
pwcsName: POleStr          {storage or stream name}
): HResult;                {returns an OLE result}
```

Description

This function removes the indicated storage or stream object from the source storage object. If this method is called on a storage object opened in transacted mode, the storage object's Commit method must be called for the change to take place. After a call to DestroyElement, any interfaces referencing the destroyed element are no longer valid.

Parameters

pwcsName: A null-terminated string indicating the name of the storage or stream object to delete.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from the following table.

See Also

IStorage.CreateStorage, IStorage.CreateStream

Example

Please see Listing 11-4 in the introduction.

Table 11-10: IStorage.DestroyElement return values

Value	Description
E_PENDING	There is a pending commitment of data, some or all of which may not be available. This is used for asynchronous storages only.
STG_E_ACCESSDENIED	Indicates the application does not have permission to access the destination file.
STG_E_FILENOTFOUND	Indicates the named element does not exist.
STG_E_INSUFFICIENTMEMORY	Operation failed due to insufficient available memory.
STG_E_INVALIDNAME	The value specified in the pwcsName parameter is invalid.
STG_E_INVALIDPOINTER	The destination stream object pointer is invalid.
STG_E_INVALIDPARAMETER	The value of one of the parameters is invalid.
STG_E_REVERTED	Indicates that the storage object has been reverted.
STG_E_TOOMANYOPENFILES	Indicates that there are too many open files.

IStorage.EnumElements ActiveX.pas

Syntax

```
EnumElements(
reserved1: Longint;        {reserved, set to zero}
reserved2: Pointer;        {reserved, set to NIL}
reserved3: Longint;        {reserved, set to zero}
out enm: IEnumStatStg      {pointer to an IEnumStatStg interface}
): HResult;                {returns an OLE result}
```

Description

This method retrieves an interface to an enumeration object, allowing the application to enumerate the storages and streams within the object. The storage object must be opened using a read mode before its elements can be enumerated.

Note: The IEnumStatStg interface is not detailed in this book, but an example of its use is given below.

Parameters

reserved1: Reserved; must be set to zero.

reserved2: Reserved; must be set to NIL.

reserved3: Reserved; must be set to zero.

enm: When the function returns, this parameter will point to a valid IEnumStatStg interface. This interface provides methods for enumerating the streams and storages within the storage object. This enumeration returns data about the storage or stream in a TStatStg record. This record is defined as:

```
TStatStg = record
      pwcsName: POleStr;              {the object name}
      dwType: Longint;                {the object type}
      cbSize: Largeint;               {the stream size}
      mtime: TFileTime;               {last modification time}
      ctime: TFileTime;               {the creation time}
      atime: TFileTime;               {last access time}
      grfMode: Longint;               {access mode flags}
      grfLocksSupported: Longint;     {region locking types supported}
      clsid: TCLSID;                  {storage object class identifier}
      grfStateBits: Longint;          {current state bits}
      reserved: Longint;              {reserved, not used}
end;
```

pwcsName: A null-terminated string indicating the name of the storage or stream object.

dwType: Indicates the type of object being enumerated. This can be one value from Table 11-11.

cbSize: Indicates the size of a stream object, in bytes.

mtime: Indicates the last modification time of the stream or storage object.

ctime: Indicates the creation time of the stream or storage object.

atime: Indicates the last access time of the stream or storage object.

grfMode: Indicates the access mode flags used when the storage or stream was created or opened. See IStorage.CreateStorage or IStorage.CreateStream for more information.

grfLocksSupported: Indicates the supported region locking types for a stream object. This can be one value from Table 11-12.

Note: The compound structured storage file implementation of the IStream interface does not support range locking, so this member should always contain zero (unless range locking is implemented in a future version of the structured storage file objects).

clsid: Indicates the class identifier for storage objects. This member is set to CLSID_NULL for newly created storages.

grfStateBits: Indicates the value set by the most recent call to IStorage.SetStateBits for storage objects.

reserved: This member is not used.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from Table 11-13.

See Also

IStorage.CreateStorage, IStorage.CreateStream, IStorage.Stat

Example

Please see Listing 11-4 in the introduction and other examples in this chapter.

Table 11-11: IStorage.EnumElements TStatStg.dwType values

Value	Description
STGTY_STORAGE	Indicates a storage object.
STGTY_STREAM	Indicates a stream object.
STGTY_LOCKBYTES	Indicates a byte-array object.
STGTY_PROPERTY	Indicates a property storage object.

Table 11-12: IStorage.EnumElements TStatStg.grfLocksSupported values

Value	Description
LOCK_WRITE	Write access to a range of bytes.
LOCK_EXCLUSIVE	Read and write access to a range of bytes.

Table 11-13: IStorage.EnumElements return values

Value	Description
E_PENDING	There is a pending commitment of data, some or all of which may not be available. This is used for asynchronous storages only.
STG_E_INSUFFICIENTMEMORY	Operation failed due to insufficient available memory.
STG_E_INVALIDPARAMETER	The value of one of the parameters is invalid.
STG_E_REVERTED	Indicates that the storage object has been reverted.

IStorage.MoveElementTo ActiveX.pas

Syntax

```
MoveElementTo(
pwcsName: POleStr;              {name of element to move}
const stgDest: IStorage;        {pointer to destination IStorage interface}
pwcsNewName: POleStr;           {new name of element}
grfFlags: Longint               {move operation flags}
): HResult;                     {returns an OLE result}
```

Description

This method moves a substorage or stream object from one storage object to another.

Parameters

pwcsName: A null-terminated string containing the name of the substorage or stream to move.

stgDest: A pointer to the destination storage object.

pwcsNewName: A null-terminated string indicating the name given to the substorage or stream in the new storage object.

grfFlags: A flag indicating the type of move operation performed. This may be one value from Table 11-14.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from Table 11-15.

See Also

IStorage.CopyTo

Example

Listing 11-8: Moving substreams into the root storage

```
{a recursive function called to enumerate a storage and populate the tree view}
procedure TfrmMain.EnumerateStorage(Tree: TTreeView; ParentNode: TTreeNode;
                                    Store: IStorage);
var
  SubStore: IStorage;            // an IStorage interface
  Enumerator: IEnumStatStg;      // the enumerator interface
  StatInfo: TStatStg;            // statistical information record
  SubNode: TTreeNode;            // tree node
begin
  {get the enumerator}
  Store.EnumElements(0, nil, 0, Enumerator);

  {begin enumerating the subelements in this storage}
  while Enumerator.Next(1, StatInfo, nil) = S_OK do
  begin
    {if this is a storage element...}
    if StatInfo.dwType = STGTY_STORAGE then
    begin
      {open the storage}
      Store.OpenStorage(StatInfo.pwcsName, nil, STGM_READWRITE or
                        STGM_SHARE_EXCLUSIVE, nil, 0, SubStore);

      {add a node to the treeview}
      SubNode := Tree.Items.AddChild(ParentNode, StatInfo.pwcsName);

      {set appropriate image indexes}
      SubNode.ImageIndex := 0;
      SubNode.SelectedIndex := 0;

      {recursively enumerate this storage}
      EnumerateStorage(Tree, SubNode, SubStore);
    end
    else
    begin
      {add a node to the treeview}
      SubNode := Tree.Items.AddChild(ParentNode, StatInfo.pwcsName);

      {set appropriate image indexes}
      SubNode.ImageIndex := 1;
      SubNode.SelectedIndex := 1;
    end;
  end;
end;

procedure TfrmMain.FormCreate(Sender: TObject);
var
  TopNode: TTreeNode;
  SubStore: IStorage;
begin
  {open this structured storage file}
  StgOpenStorage('TestStrucFile.tmp', nil,
                 STGM_READWRITE or STGM_SHARE_EXCLUSIVE,
                 nil, 0, RootStore);

  {add a node to the treeview}
  TopNode := treBefore.Items.Add(nil, 'TestStrucFile.tmp');

  {set the image index}
  TopNode.ImageIndex := 0;
  TopNode.SelectedIndex := 0;

  {enumerate this root storage object}
  EnumerateStorage(treBefore, TopNode, RootStore);

  {expand the treeview}
  TopNode.Expand(TRUE);

  {open the substorage}
  RootStore.OpenStorage('Substorage1', nil, STGM_READWRITE or
                        STGM_SHARE_EXCLUSIVE, nil, 0, SubStore);
  {move the two streams in this substorage into the root storage}
  SubStore.MoveElementTo('Substream1', RootStore, 'CopyOfSubStream1',
                         STGMOVE_MOVE);
  SubStore.MoveElementTo('Substream2', RootStore, 'CopyOfSubStream2',
                         STGMOVE_MOVE);

  {close the substorage}
  SubStore := nil;

  {add a node to the treeview}
  TopNode := treAfter.Items.Add(nil, 'TestStrucFile.tmp');

  {set the image index}
  TopNode.ImageIndex := 0;
  TopNode.SelectedIndex := 0;

  {enumerate this root storage object}
  EnumerateStorage(treAfter, TopNode, RootStore);

  {expand the treeview}
  TopNode.Expand(TRUE);
end;
```

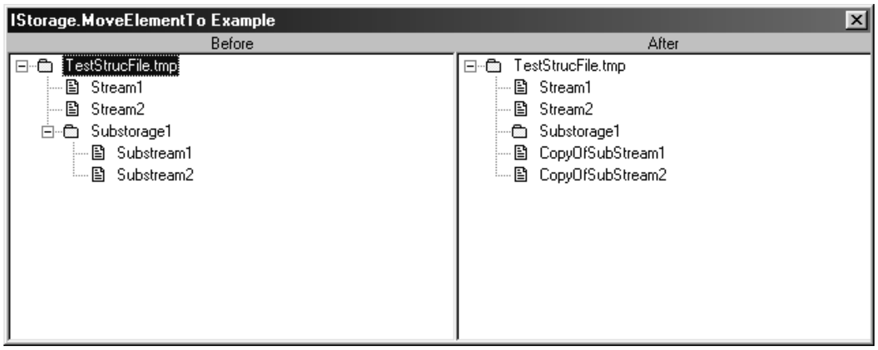


Figure 11-4: The modified structured storage file

Table 11-14: IStorage.MoveElementTo grfFlags values

Value	Description
STGMOVE_MOVE	Copies the information from the source to the destination and then removes it from the source.
STGMOVE_COPY	Copies the information from the source to the destination.

Table 11-15: IStorage.MoveElementTo return values

Value	Description
E_PENDING	There is a pending commitment of data, some or all of which may not be available. This is used for asynchronous storages only.
STG_E_ACCESSDENIED	Indicates the application does not have permission to access the destination file.
STG_E_FILENOTFOUND	Indicates the named element does not exist.

Value	Description
STG_E_FILEALREADYEXISTS	Indicates that a stream or storage of the specified name already exists in the destination storage.
STG_E_INSUFFICIENTMEMORY	Operation failed due to insufficient available memory.
STG_E_INVALIDFLAG	The value of the grfFlags parameter is not valid.
STG_E_INVALIDNAME	The value specified in the pwcsName parameter is invalid.
STG_E_INVALIDPOINTER	The destination storage object pointer is invalid.
STG_E_INVALIDPARAMETER	The value of one of the parameters is invalid.
STG_E_REVERTED	Indicates that the storage object has been reverted.
STG_E_TOOMANYOPENFILES	Indicates that there are too many open files.

IStorage.OpenStorage ActiveX.pas

Syntax

```
OpenStorage(
pwcsName: POleStr;              {the name of the substorage object}
const stgPriority: IStorage;    {IStorage interface opened in priority mode}
grfMode: Longint;               {access mode flags}
snbExclude: TSNB;               {unused, set to NIL}
reserved: Longint;              {unused, set to zero}
out stg: IStorage               {returns a pointer to an IStorage interface}
): HResult;                     {returns an OLE result}
```

Description

This method opens the specified substorage within the source storage object.

Parameters

pwcsName: A null-terminated string indicating the name of the substorage object to open.

stgPriority: A pointer to an IStorage interface for a substorage element of the storage object that has been opened in priority mode. When this function returns, the interface supplied to this parameter is no longer valid, and the application should subsequently use the interface returned in the stg parameter. This parameter can be set to NIL.

grfMode: Flags indicating the access mode for the storage. This parameter can be one or more values from Table 11-16.

Note: The STGM_SHARE_EXCLUSIVE flag must be used in combination with any other specified flag.

snbExclude: Unused; set to NIL.

reserved: Reserved; must be set to zero.

stg: When the function returns, this parameter will point to a valid IStorage interface for the opened substorage object.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from Table 11-17, and the stg parameter returns NIL.

See Also

IStorage.OpenStream, StgCreateDocFile, StgOpenStorage

Example

Please see Listing 11-4 in the introduction.

Table 11-16: IStorage.OpenStorage grfMode values

Value	Description
STGM_READ	Indicates that the object's data can be accessed but not modified. Cannot be combined with STGM_WRITE or STGM_READWRITE.
STGM_WRITE	Indicates that the object's data can be modified but not accessed. Cannot be combined with STGM_READ or STGM_READWRITE.
STGM_READWRITE	Indicates that the object's data can be both accessed and modified. Cannot be combined with STGM_READ or STGM_WRITE.
STGM_SHARE_EXCLUSIVE	Indicates that other processes cannot open the object in any mode. This flag must be used.
STGM_PRIORITY	Provides exclusive access to the most recently committed version of the object. The STGM_DIRECT and STGM_READ flags must be used with this flag.
STGM_DIRECT	Causes any changes to be written as they occur. Cannot be combined with STGM_TRANSACTED.
STGM_TRANSACTED	Any changes to the object are buffered and are only written when the Commit method is called. Call the Revert method to discard any changes since the last call to Commit. Cannot be combined with STGM_DIRECT.
STGM_NOSCRATCH	Must be used with STGM_TRANSACTED. This causes the system to use any unused areas of the original file as a "scratch" storage space for uncommitted changes. Without this flag, a temporary file is created to store uncommitted changes.
STGM_SIMPLE	Opens the compound structured storage file in simple mode. It offers efficient performance but does not support substorages, and all streams are a minimum of 4Kb in size.
STGM_DIRECT_SWMR	Provides direct mode for single-writer, multi-reader operations.

Table 11-17: IStorage.OpenStorage return values

Value	Description
E_PENDING	There is a pending commitment of data, some or all of which may not be available. This is used for asynchronous storages only.
STG_E_ACCESSDENIED	Indicates the application does not have permission to access the destination file.
STG_E_FILENOTFOUND	Indicates the named element does not exist.
STG_E_INSUFFICIENTMEMORY	Operation failed due to insufficient available memory.
STG_E_INVALIDFLAG	The value of the grfMode parameter is not valid.
STG_E_INVALIDFUNCTION	The combination of values specified in grfMode is invalid.
STG_E_INVALIDNAME	The value specified in the pwcsName parameter is invalid.
STG_E_INVALIDPOINTER	The destination storage object pointer is invalid.
STG_E_INVALIDPARAMETER	The value of one of the parameters is invalid.
STG_E_REVERTED	Indicates that the storage object has been reverted.
STG_E_TOOMANYOPENFILES	Indicates that there are too many open files.
STG_S_CONVERTED	Indicates that an existing storage with the same name has been converted into a stream called CONTENTS.

IStorage.OpenStream *ActiveX.pas*

Syntax

```
OpenStream(
pwcsName: POleStr;          {the name of the stream}
reserved1: Pointer;         {reserved, set to NIL}
grfMode: Longint;           {access mode flags}
reserved2: Longint;         {reserved, set to zero}
out stm: IStream            {returns a pointer to an IStream interface}
): HResult;                 {returns an OLE result}
```

Description

This method opens a stream within the storage object.

Parameters

pwcsName: A null-terminated string indicating the name of the stream to open.

reserved1: Reserved; must be set to NIL.

grfMode: Flags indicating the access mode for the stream. This parameter can be one or more values from Table 11-18.

Note: The STGM_SHARE_EXCLUSIVE flag must be used in combination with any other specified flag.

reserved2: Reserved; must be set to zero.

stm: When the function returns, this parameter will point to a valid IStream interface for the opened stream object.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from Table 11-19, and the stm parameter returns NIL.

See Also

IStorage.CreateStream

Example

Please see Listing 11-4 in the introduction.

Table 11-18: IStorage.OpenStream grfMode values

Value	Description
STGM_READ	Indicates that the object's data can be accessed but not modified. Cannot be combined with STGM_WRITE or STGM_READWRITE.
STGM_WRITE	Indicates that the object's data can be modified but not accessed. Cannot be combined with STGM_READ or STGM_READWRITE.
STGM_READWRITE	Indicates that the object's data can be both accessed and modified. Cannot be combined with STGM_READ or STGM_WRITE.
STGM_SHARE_EXCLUSIVE	Indicates that other processes cannot open the object in any mode. This flag must be used.

Table 11-19: IStorage.OpenStream return values

Value	Description
E_PENDING	There is a pending commitment of data, some or all of which may not be available. This is used for asynchronous storages only.
STG_E_ACCESSDENIED	Indicates the application does not have permission to access the destination file.
STG_E_FILENOTFOUND	Indicates the named element does not exist.
STG_E_INSUFFICIENTMEMORY	Operation failed due to insufficient available memory.
STG_E_INVALIDFLAG	The value of the grfMode parameter is not valid.
STG_E_INVALIDFUNCTION	The combination of values specified in grfMode is invalid.
STG_E_INVALIDNAME	The value specified in the pwcsName parameter is invalid.
STG_E_INVALIDPOINTER	The destination stream object pointer is invalid.
STG_E_INVALIDPARAMETER	The value of one of the parameters is invalid.

Value	Description
STG_E_REVERTED	Indicates that the storage object has been reverted.
STG_E_TOOMANYOPENFILES	Indicates that there are too many open files.

IStorage.RenameElement *ActiveX.pas*

Syntax

```
RenameElement(
pwcsOldName: POleStr;           {old element name}
pwcsNewName: POleStr            {new element name}
): HResult;                     {returns an OLE result}
```

Description

This method renames a substorage or stream object within the storage. This method can only rename substorage or stream objects that are not currently open.

Parameters

pwcsOldName: A null-terminated string indicating the name of the substorage or stream to rename.

pwcsNewName: A null-terminated string indicating the new name of the substorage or stream.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from the following table.

See Also

IStorage.CreateStorage, IStorage.CreateStream, IStorage.OpenStorage, IStorage.OpenStream

Example

Please see Listing 11-4 in the introduction.

Table 11-20: IStorage.RenameElement return values

Value	Description
E_PENDING	There is a pending commitment of data, some or all of which may not be available. This is used for asynchronous storages only.
STG_E_ACCESSDENIED	Indicates the application does not have permission to access the destination file, or the renamed element is open.
STG_E_FILENOTFOUND	Indicates the named element does not exist.
STG_E_INSUFFICIENTMEMORY	Operation failed due to insufficient available memory.
STG_E_INVALIDNAME	The value specified in the pwcsOldName or pwcsNewName parameter is invalid.

Value	Description
STG_E_REVERTED	Indicates that the storage object has been reverted.
STG_E_TOOMANYOPENFILES	Indicates that there are too many open files.

IStorage.Revert *ActiveX.pas*

Syntax

```
Revert: HResult;                {returns an OLE result}
```

Description

This method causes storage objects opened in transacted mode to discard all changes made since the last call to IStorage.Commit. Any pointers to open substorages or stream objects within the reverted storage object are invalidated and should be released.

Note: This method has no effect if the storage object is opened in direct mode.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from the following table.

See Also

IStorage.CreateStorage, StgCreateDocFile

Example

Please see Listing 11-4 in the introduction.

Table 11-21: IStorage.Revert return values

Value	Description
E_PENDING	There is a pending commitment of data, some or all of which may not be available. This is used for asynchronous storages only.
STG_E_INSUFFICIENTMEMORY	Operation failed due to insufficient available memory.
STG_E_REVERTED	Indicates that the storage object has been reverted.
STG_E_TOOMANYOPENFILES	Indicates that there are too many open files.

IStorage.SetClass *ActiveX.pas*

Syntax

```
SetClass(
const clsid: TCLSID             {a class identifier}
): HResult;                     {returns an OLE result}
```

Description

This method assigns the specified class identifier to the storage object.

Parameters

clsid: The class identifier to associate with the storage object.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from the following table.

See Also

IStorage.EnumElements, IStorage.Stat

Example

Please see Listing 11-4 in the introduction.

Table 11-22: IStorage.SetClass return values

Value	Description
E_PENDING	There is a pending commitment of data, some or all of which may not be available. This is used for asynchronous storages only.
STG_E_ACCESSDENIED	Indicates the application does not have permission to access the destination file.
STG_E_REVERTED	Indicates that the storage object has been reverted.

IStorage.Stat ActiveX.pas

Syntax

```
Stat(
out statstg: TStatStg;          {TStatStg statistical information structure}
grfStatFlag: Longint            {statistic exclusion flags}
): HResult;                     {returns an OLE result}
```

Description

This method returns a structure initialized with statistical information regarding the storage object.

Parameters

statstg: A pointer to a TStatStg structure that is initialized with storage information. TStatStg is defined as:

```
TStatStg = record
      pwcsName: POleStr;          {the object name}
      dwType: Longint;            {the object type}
      cbSize: Largeint;           {the stream size}
      mtime: TFileTime;           {last modification time}
```

```
        ctime: TFileTime;                 {the creation time}
        atime: TFileTime;                 {last access time}
        grfMode: Longint;                 {access mode flags}
        grfLocksSupported: Longint;       {region locking types supported}
        clsid: TCLSID;                    {storage object class identifier}
        grfStateBits: Longint;            {current state bits}
        reserved: Longint;                {reserved, not used}
    end;
```

See IStorage.EnumElements for more details.

grfStatFlag: Flags indicating whether or not the pwcsName member of the TStatStg structure should be returned. This can be one value from Table 11-23.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from Table 11-24.

See Also

IStorage.EnumElements, IStream.Stat

Example

Please see Listing 11-4 in the introduction.

Table 11-23: IStorage.Stat grfStatFlag values

Value	Description
STATFLAG_DEFAULT	Returns the element name in the pwcsName member.
STATFLAG_NONAME	Does not return the element name in the pwcsName member. This saves on resources and reduces the time required to retrieve the information.

Table 11-24: IStorage.Stat return values

Value	Description
E_PENDING	There is a pending commitment of data, some or all of which may not be available. This is used for asynchronous storages only.
STG_E_ACCESSDENIED	Indicates the application does not have permission to access the file.
STG_E_INSUFFICIENTMEMORY	Operation failed due to insufficient available memory.
STG_E_INVALIDFLAG	The value of the grfStatFlag parameter is not valid.
STG_E_INVALIDPARAMETER	The value of one of the parameters is invalid.

IStream.Clone ActiveX.pas

Syntax

```
Clone(
out stm: IStream          {returns an IStream interface}
): HResult;               {returns an OLE result}
```

Description

This method returns an IStream interface with its own seek position that references the same bytes as the original stream. Any changes made to one stream are reflected in the other, and range locking is shared between the objects. When the new stream is returned, its seek position is at the same point in the stream as that of the original.

Parameters

stm: When the function returns, this parameter will point to a valid IStream interface for the cloned stream object.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from the following table.

See Also

IStorage.CreateStream, IStorage.OpenStream

Example

Listing 11-9: Reading data from a cloned stream

```
procedure TForm1.FormCreate(Sender: TObject);
var
  RootStore: IStorage;
  OrgStream, CloneStream: IStream;
  StreamBuffer: PChar;
  SeekPos: Largeint;             // seek position variable
  BytesReadWritten: Longint;     // bytes written variable
begin
  {open this structured storage file }
  StgOpenStorage('StreamCloneTest.tmp', nil, STGM_READWRITE or
                 STGM_SHARE_EXCLUSIVE, nil, 0, RootStore);

  {open the stream object}
  RootStore.OpenStream('Stream1', nil, STGM_READWRITE or STGM_SHARE_EXCLUSIVE,
                       0, OrgStream);

  {clone the stream}
  OrgStream.Clone(CloneStream);

  {allocate a buffer large enough to hold the entire stream contents}
  GetMem(StreamBuffer, 11);
  FillMemory(StreamBuffer, 11, 0);
  {retrieve the entire stream contents}
  OrgStream.Seek(0, STREAM_SEEK_SET, SeekPos);
  OrgStream.Read(StreamBuffer, 10, @BytesReadWritten);

  {display the stream contents}
  memBefore.Text := StreamBuffer;

  {now, define some new data}
  StreamBuffer := '9876543210'#0;

  {write the information starting at the front of the stream}
  OrgStream.Seek(0, STREAM_SEEK_SET, SeekPos);
  OrgStream.Write(StreamBuffer, 10, @BytesReadWritten);

  {empty the buffer}
  FillMemory(StreamBuffer, 11, 0);

  {now, retrieve the entire stream contents of the cloned stream}
  CloneStream.Seek(0, STREAM_SEEK_SET, SeekPos);
  CloneStream.Read(StreamBuffer, 10, @BytesReadWritten);

  {display the stream contents. this should match what we just wrote,
   even though that data was written to the original stream}
  memAfter.Text := StreamBuffer;
end;
```

Table 11-25: IStream.Clone return values

Value	Description
STG_E_INSUFFICIENTMEMORY	Operation failed due to insufficient available memory.
STG_E_INVALIDPOINTER	The destination stream object pointer is invalid.
STG_E_REVERTED	Indicates that the storage object has been reverted.

IStream.CopyTo ActiveX.pas

Syntax

```
CopyTo(
stm: IStream;                {destination stream interface}
cb: Largeint;                {number of bytes to copy}
out cbRead: Largeint;        {number of bytes read from source}
out cbWritten: Largeint      {number of bytes written to destination}
): HResult;                  {returns an OLE result}
```

Description

This method copies the indicated number of bytes from one stream into another. The copy begins at the current seek position in the source stream and is copied into the destination stream beginning at the destination stream's seek position. The destination stream can be the same as the source, and the seek position in each stream is adjusted by the number of bytes read or written.

Parameters

stm: A pointer to the destination stream object.

cb: Indicates the number of bytes to copy.

cbRead: A variable receiving the number of bytes read from the destination stream.

cbWritten: A variable receiving the number of bytes written to the destination stream.

Return Value

If the function succeeds, it returns S_OK, and the values returned in the cbRead and cbWritten parameters are equal. If the function fails, it returns an OLE failure result code from the following table, but the values returned in cbRead and cbWritten are undefined and the seek positions in each stream are invalid.

See Also

IStorage.CopyTo, IStorage.CreateStream, IStream.Clone

Example

Please see Listing 11-4 in the introduction.

Table 11-26: IStream.CopyTo return values

Value	Description
STG_E_INVALIDPOINTER	The destination stream object pointer is invalid.
STG_E_MEDIUMFULL	There is not enough space left on the storage device to write changes.
STG_E_REVERTED	Indicates that the storage object has been reverted.

IStream.Read ActiveX.pas

Syntax

```
Read(
pv: Pointer;                {a pointer to a destination buffer}
cb: Longint;                {number of bytes to read}
pcbRead: PLongint           {number of bytes read}
): HResult;                 {returns an OLE result}
```

Description

This method reads the indicated number of bytes from the stream into a buffer, starting at the stream's current seek position. The stream must be opened in a read mode, and the seek position is advanced by the number of bytes read.

Parameters

pv: A pointer to a buffer that receives the bytes read from the stream.

cb: Indicates the number of bytes to read from the stream into the buffer.

pcbRead: Indicates the number of bytes actually read from the stream. This parameter can be set to NIL.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from the following table, but the value returned in pcbRead is undefined and the seek position is invalid.

See Also

IStream.Write

Example

Please see Listing 11-4 in the introduction.

Table 11-27: IStream.Read return values

Value	Description
S_FALSE	Indicates that data could not be read from the stream.
STG_E_ACCESSDENIED	Indicates the application does not have read permission to access the stream.
STG_E_INVALIDPOINTER	The destination stream object pointer is invalid.
STG_E_REVERTED	Indicates that the storage object has been reverted.

IStream.Seek ActiveX.pas

Syntax

```
Seek(
dlibMove: Largeint;              {number of bytes to move}
dwOrigin: Longint;               {seek origin flags}
out libNewPosition: Largeint     {returns the new seek position}
): HResult;                      {returns an OLE result}
```

Description

This method moves the seek position of the stream by the specified value according to the flags specified in the dwOrigin parameter.

Note: Setting the seek position before the beginning of the stream causes the method to fail. However, setting the seek position past the end of the stream is legal.

Parameters

dlibMove: Indicates the number of bytes to move the seek pointer, as defined by the dwOrigin value. Set this parameter to zero and dwOrigin to STREAM_SEEK_CUR to retrieve the current seek position.

dwOrigin: A flag indicating the relative origin of the offset specified by the dlibMove parameter. This can be one value from Table 11-28.

libNewPosition: When the function returns, this variable receives the updated seek position, relative to the beginning of the stream.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from Table 11-29.

See Also

IStream.Read, IStream.Write

Example

Please see Listing 11-4 in the introduction.

Table 11-28: IStream.Seek dwOrigin values

Value	Description
STREAM_SEEK_SET	Indicates the new seek pointer is an offset relative to the beginning of the stream. The offset uses the absolute value of the dlibMove parameter.
STREAM_SEEK_CUR	Indicates the new seek pointer is an offset relative to the current seek pointer.
STREAM_SEEK_END	Indicates the new seek pointer is an offset relative to the end of the stream.

Table 11-29: IStream.Seek return values

Value	Description
STG_E_INVALIDFUNCTION	Either the dwOrigin parameter contains an invalid value or the dlibMove parameter contains an invalid offset.
STG_E_REVERTED	Indicates that the storage object has been reverted.

IStream.SetSize ActiveX.pas

Syntax

```
SetSize(
libNewSize: Largeint          {the new size of the stream}
): HResult;                   {returns an OLE result}
```

Description

This method sets the size of the stream object. If the new size is larger than the original stream, the new bytes are initialized to an undefined value. If the new size is smaller, the stream data is truncated. The seek position is not affected by this method.

Parameters

libNewSize: Indicates the new size of the stream object, in bytes.

Note: The structured storage file implementation of streams restricts their size to a maximum of 232 bytes.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from the following table.

See Also

IStorage.CreateStream, IStream.Write

Example

Please see Listing 11-4 in the introduction.

Table 11-30: IStream.SetSize return values

Value	Description
STG_E_INVALIDFUNCTION	Indicates the libNewSize parameter is invalid.
STG_E_MEDIUMFULL	There is not enough space left on the storage device to write changes.
STG_E_REVERTED	Indicates that the storage object has been reverted.

IStream.Stat ActiveX.pas

Syntax

```
Stat(
out statstg: TStatStg;          {TStatStg statistical information structure}
grfStatFlag: Longint            {statistic exclusion flags}
): HResult;                     {returns an OLE result}
```

Description

This method returns a structure initialized with statistical information regarding the stream object.

Parameters

statstg: A pointer to a TStatStg structure that is initialized with stream information. TStatStg is defined as:

```
TStatStg = record
    pwcsName: POleStr;              {the object name}
    dwType: Longint;                {the object type}
    cbSize: Largeint;               {the stream size}
    mtime: TFileTime;               {last modification time}
    ctime: TFileTime;               {the creation time}
    atime: TFileTime;               {last access time}
    grfMode: Longint;               {access mode flags}
    grfLocksSupported: Longint;     {region locking types supported}
```

```
        clsid: TCLSID;                  {storage object class identifier}
        grfStateBits: Longint;          {current state bits}
        reserved: Longint;              {reserved, not used}
end;
```

See IStorage.EnumElements for more details.

grfStatFlag: Flags indicating whether or not the pwcsName member of the TStatStg structure should be returned. This can be one value from Table 11-31.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from Table 11-32.

See Also

IStorage.EnumElements, IStorage.Stat

Example

Please see Listing 11-4 in the introduction.

Table 11-31: IStream.Stat grfStatFlag values

Value	Description
STATFLAG_DEFAULT	Returns the element name in the pwcsName member.
STATFLAG_NONAME	Does not return the element name in the pwcsName member. This saves on resources and reduces the time required to retrieve the information.

Table 11-32: IStream.Stat return values

Value	Description
STG_E_ACCESSDENIED	Indicates the application does not have permission to access the stream.
STG_E_INSUFFICIENTMEMORY	Operation failed due to insufficient available memory.
STG_E_INVALIDFLAG	The value of the grfStatFlag parameter is not valid.
STG_E_INVALIDPARAMETER	The value of one of the parameters is invalid.

IStream.Write   ActiveX.pas

Syntax

```
Write(
pv: Pointer;              {a pointer to a source buffer}
cb: Longint;              {number of bytes to write}
pcbWritten: PLongint      {number of bytes written}
): HResult;               {returns an OLE result}
```

Description

This method writes the indicated number of bytes from the buffer into the stream, starting at the stream's current seek position. The stream must be opened in a write mode, and the seek position is advanced by the number of bytes written. This method increases the size of a stream if the seek position is at the end of the stream data when the write occurs.

Note: If the seek position is past the end of the stream when this method is called, the buffer is written to the stream and the stream size is increased appropriately, but the bytes between the end of the stream and the seek position's original position are initialized to undefined values.

Parameters

pv: A pointer to the buffer containing the data to be written to the stream.

cb: Indicates the number of bytes to write from the buffer into the stream.

pcbWritten: Indicates the number of bytes actually written to the stream. This parameter can be set to NIL.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from the following table, but the value returned in pcbWritten is undefined and the seek position is invalid.

See Also

IStream.Read

Example

Please see Listing 11-4 in the introduction.

Table 11-33: IStream.Write return values

Value	Description
STG_E_ACCESSDENIED	Indicates the application does not have write permission to the stream.
STG_E_CANTSAVE	Data cannot be written to the disk for reasons unrelated to access restrictions or insufficient storage space.
STG_E_INVALIDPOINTER	The destination stream object pointer is invalid.
STG_E_REVERTED	Indicates that the storage object has been reverted.
STG_E_WRITEFAULT	Indicates a failure due to a disk error (returned when the storage is opened using the STGM_SIMPLE flag).

SHAddToRecentDocs ShlObj.pas

Syntax

```
SHAddToRecentDocs(
uFlags: UINT;            {a value indicating the contents of the pv parameter}
pv: Pointer                {a pointer to a buffer or an item ID list}
);                         {this procedure does not return a value}
```

Description

This function adds or removes files to the recent documents list. This list is accessed through the Start button in the Documents menu item. Only registered documents (those that have an associated executable file) can be added to this list.

Parameters

uFlags: A value indicating what the pv parameter contains. This parameter can contain one value from the following table.

pv: Either a pointer to a null-terminated string containing the path and filename of a document or a pointer to an item identifier list uniquely identifying the document. If this parameter is NIL, the recent documents list is cleared.

See Also

SHGetFileInfo

Example

Listing 11-10: Adding a document to the recent documents list

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  {clear all documents from the recent docs list}
  SHAddToRecentDocs(SHARD_PATH, nil);
end;

procedure TForm1.FileListBox1DblClick(Sender: TObject);
var
  TheFileName: string;
begin
  {retrieve the filename of the selected document}
  TheFileName := FileListBox1.FileName;

  {add it to the recent docs list. note that the file must be registered (must have an
   associated executable) before it is added to the list}
  SHAddToRecentDocs(SHARD_PATH, PChar(TheFileName));
end;
```

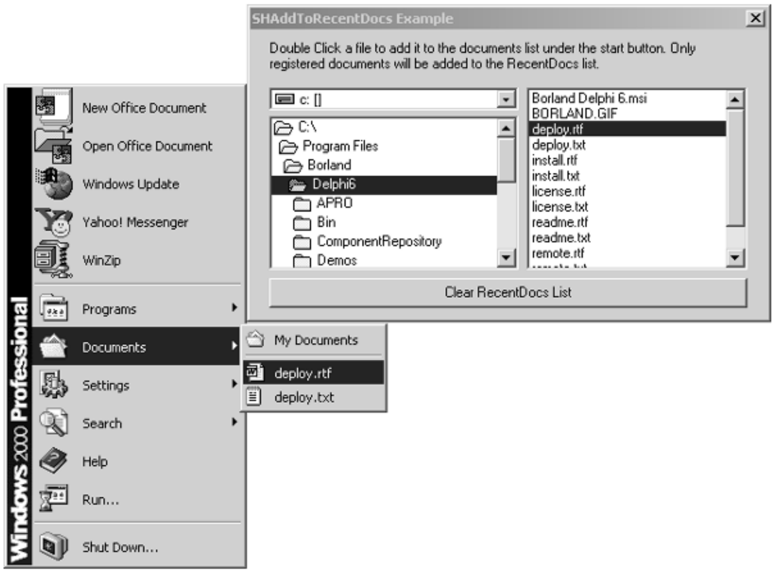


Figure 11-5: A document was added to the list

Table 11-34: SHAddToRecentDocs uFlags values

Value	Description
SHARD_PATH	The pv parameter contains the address of a null-terminated string containing the path and filename of a document.
SHARD_PIDL	The pv parameter contains the address of an item identifier list uniquely identifying the document.

SHFileOperation   ShellAPI.pas

Syntax

```
SHFileOperation(
const lpFileOp: TSHFileOpStruct   {a pointer to a file operation data structure}
): Integer;                       {returns an integer value}
```

Description

This function copies, deletes, moves, or renames files or folders.

Note: Use fully qualified path names with this function if it is used in a multithreaded application. Using relative path names is not thread-safe.

Parameters

lpFileOp: A TSHFileOpStruct that contains information about the files and the action to perform. This structure is defined as:

```
TSHFileOpStruct = record
      Wnd: HWND;                  {a handle to a window}
```

```
        wFunc: UINT;                       {a flag indicating the operation}
        pFrom: PAnsiChar;                  {a pointer to the source file names}
        pTo: PAnsiChar;                    {a pointer to the destination names}
        fFlags: FILEOP_FLAGS;              {operation control flags}
        fAnyOperationsAborted: BOOL;       {aborted operation flag}
        hNameMappings: Pointer;            {a handle to a filename mapping object}
        lpszProgressTitle: PAnsiChar;      {the progress dialog box title}
end;
```

Wnd: A handle to a window used to display the progress of the file operation.

wFunc: A flag indicating the operation to perform. This member can be one value from Table 11-35.

pFrom: A pointer to a buffer containing the filenames upon which to perform the indicated operation. If multiple filenames are specified, each must be separated with a null terminating character, and the entire buffer must end with two null terminating characters. If the filenames do not contain a path, the source directory is assumed to be the directory as reported by the GetCurrentDirectory function. Standard DOS wildcard characters (i.e., "*", "?") can be used in the filename.

pTo: A pointer to a buffer containing the name of the destination file or directory. If the fFlags member contains FOF_MULTIDESTFILES, this buffer can contain multiple destination filenames, one for each source file. Each destination filename must be separated with a null terminating character, and the entire buffer must end with two null terminating characters. If the filenames do not contain a path, the destination directory is assumed to be the directory as reported by the GetCurrentDirectory function. DOS wildcard characters cannot be used in this member. For move and copy operations, destination directories that do not exist are created when possible.

fFlags: An array of flags indicating the type of operation to perform on the specified files. This member can contain one or more values from Table 11-36.

fAnyOperationsAborted: This member receives a value of TRUE if any file operations were aborted by the user before completion. Otherwise, it receives a value of FALSE.

hNameMappings: If the fFlags member contains the FOF_WANTMAPPING-HANDLE flag, this member receives a pointer to a THandleToMappings structure containing an array of TSHNameMapping structures. These structures contain the old and new path and filename for each file that was moved, copied, or renamed. The filename mapping structure must be deleted using the SHFreeNameMappings function.

lpszProgressTitle: A null-terminated string used as the title for the progress dialog box. This member is used only if the fFlags member contains the FOF_SIMPLE- PROGRESS flag.

The THandleToMappings structure pointed to by the hNameMappings member is defined as:

```
THandleToMappings = packed record
        uNumberOfMappings: integer;       {number of name mapping structures}
        lpSHNameMapping: PSHNameMapping;      {array of name mapping structures}
end;
```

uNumberOfMappings: Indicates the number of TSHNameMapping structures pointed to by the lpSHNameMapping member.

lpSHNameMapping: Contains a pointer to the first TSHNameMapping structure in the array.

The TSHNameMapping structures pointed to by the THandleToMappings structure pointed to by the hNameMappings member are defined as:

```
TSHNameMapping = record
        pszOldPath: PAnsiChar;           {a pointer to a string}
        pszNewPath: PAnsiChar;           {a pointer to a string}
        cchOldPath: Integer;             {a string size value}
        cchNewPath: Integer;             {a string size value}
end;
```

pszOldPath: A null-terminated string specifying the original path and filename.

pszNewPath: A null-terminated string specifying the new path and filename.

cchOldPath: The number of characters in the pszOldPath member.

cchNewPath: The number of characters in the pszNewPath member.

Return Value

If the function succeeds, it returns a number greater than zero; otherwise, it returns zero.

See Also

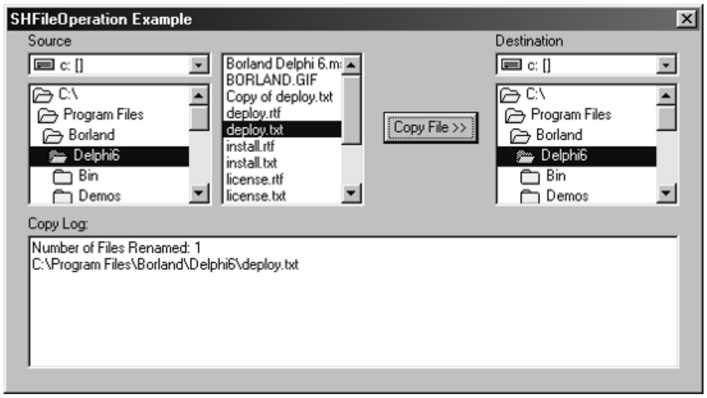
GetCurrentDirectory, SetCurrentDirectory, ShellExecute, SHFreeNameMappings

Example

Listing 11-11: Copying a file

```
type
  {this structure is not defined in Delphi}
  THandleToMappings = packed record
    uNumberOfMappings: integer;
    lpSHNameMapping: PSHNameMapping;
  end;

const
  {these SHFileOperation constants are not defined in Delphi}
  FOF_NOCOPYSECURITYATTRIBS = $0800;  // don't copy NT file Security Attributes
  FOF_NORECURSION           = $1000;  // don't recurse into directories
  FOF_NO_CONNECTED_ELEMENTS = $2000;  // don't operate on connected elements
  FOF_WANTNUKEWARNING       = $4000;  // warn if deleting

var
  Form1: TForm1;
```

```
implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var
  FileOpInfo: TSHFileOpStruct;      // holds information about the file
  FileMappings: THandleToMappings;  // holds the name mappings
  iCount: Integer;                  // loop control
  NameMapping: PSHNameMapping;      // a name mapping structure
begin
  with FileOpInfo do
  begin
    Wnd    := Form1.Handle;
    wFunc  := FO_COPY;                             // perform a copy
    pFrom  := PChar(FileListBox1.FileName+#0+#0);  // the source
    pTo    := PChar(DirectoryListBox2.Directory);  // the destination directory

    {indicate that we want files to be renamed if they already exist,
     we want a mapping handle, and no recursion}
    fFlags := FOF_RENAMEONCOLLISION or FOF_WANTMAPPINGHANDLE or FOF_NORECURSION;
  end;

  {perform the file operation}
  SHFileOperation(FileOpInfo);

  {if a mapping handle was returned, we had some files renamed, so let's
   list their original file name}
  if FileOpInfo.hNameMappings <> nil then
  begin
    {get the file mappings structure}
    FileMappings := THandleToMappings(FileOpInfo.hNameMappings^);

    {prepare to show the names}
    memLog.Lines.Clear;
    memLog.Lines.Add('Number of Files Renamed: ' +
                     IntToStr(FileMappings.uNumberOfMappings));

    {loop through the file mapping array, displaying the original path
     and file name of any files that were renamed}
    NameMapping := FileMappings.lpSHNameMapping;
    for iCount := 0 to FileMappings.uNumberOfMappings - 1 do
    begin
      memLog.Lines.Add(string(NameMapping.pszOldPath));
      Inc(NameMapping);
    end;

    {finally, free the file mapping object}
    SHFreeNameMappings(Cardinal(FileOpInfo.hNameMappings));
  end;
end;
```



```
Number of Files Renamed: 1
C:\Program Files\Borland\Delphi6\deploy.txt
```

Figure 11-6: The file was copied

Table 11-35: SHFileOperation lpFileOp.wFunc values

Value	Description
FO_COPY	Copies the files specified by the pFrom member to the location specified by the pTo member.
FO_DELETE	Deletes the files specified by the pFrom member. The pTo member is ignored.
FO_MOVE	Moves the files specified by the pFrom member to the location specified by the pTo member.
FO_RENAME	Renames the files specified by the pFrom member. The pTo member is ignored.

Table 11-36: SHFileOperation lpFileOp.fFlags values

Value	Description
FOF_ALLOWUNDO	The specified file is deleted to the recycle bin. If the pFrom member does not contain a fully qualified path, this value is ignored.
FOF_FILESONLY	The operation is performed only on files if a wildcard filename is specified (i.e., "*.pas").
FOF_MULTIDESTFILES	The pTo member contains one destination file for each source file instead of one directory to which all source files are deposited.
FOF_NOCONFIRMATION	The user is never asked for confirmation, and the operation continues as if a response of "yes to all" was indicated.
FOF_NOCONFIRMMKDIR	Automatically creates a new directory if one is needed without asking the user for confirmation.
FOF_NOCOPYSECURITYATTRIBS	**Windows NT/2000 and later**: Does not copy the file security attributes.
FOF_NOERRORUI	There is no visual indication if an error occurs.
FOF_NORECURSION	Performs the file operation on the files in the local directory only and does not continue file operations in subdirectories.

Value	Description
FOF_RENAMEONCOLLISION	The source file is automatically given a new name, such as "Copy #1 of..," in a move, copy, or rename operation if a file in the target directory already has the same name.
FOF_SILENT	Does not display a progress dialog box.
FOF_SIMPLEPROGRESS	Displays a progress dialog box but does not show filenames.
FOF_WANTMAPPINGHANDLE	The hNameMappings member receives a handle to a filename mapping object if any files were renamed. FOF_RENAMEONCOLLISION must be used in conjunction with this flag.
FOF_WANTNUKEWARNING	Displays a warning dialog box when a file is deleted.

SHFreeNameMappings ShellAPI.pas

Syntax

```
SHFreeNameMappings(
hNameMappings: THandle          {a handle to a filename mapping object}
);                              {this procedure does not return a value}
```

Description

This function frees the filename mapping object as returned by the SHFileOperation function.

Parameters

hNameMappings: A handle to the filename mapping object to free.

See Also

SHFileOperation

Example

Please see Listing 11-11 under SHFileOperation.

SHGetFileInfo ShellAPI.pas

Syntax

```
SHGetFileInfo(
pszPath: PAnsiChar;             {a pointer to a filename string}
dwFileAttributes: DWORD;        {file attribute flags}
var psfi: TSHFileInfo;          {a pointer to a TSHFileInfo structure}
cbFileInfo: UINT;               {the size of the TSHFileInfo structure}
uFlags: UINT                    {information retrieval flags}
): DWORD;                       {returns a double word value}
```

Description

This function retrieves information about a file, folder, directory, or drive root.

Parameters

pszPath: A pointer to a null-terminated string containing the path and filename of the file whose information is to be retrieved. This can be either a long filename or in the DOS 8.3 filename format. If the uFlags parameter contains the SHGFI_PIDL flag, this parameter can point to an item identifier list for the file.

dwFileAttributes: An array of flags indicating the file attribute flags. This parameter may contain one or more of the values from Table 11-37. If the uFlags parameter does not contain the SHGFI_USEFILEATTRIBUTES flag, this parameter is ignored.

psfi: A pointer to a TSHFileInfo data structure. This structure contains the requested information about the specified file. The TSHFileInfo structure is defined as:

```
TSHFileInfo = record
    hIcon: HICON;                                      {an icon handle}
    iIcon: Integer;                                    {an icon index}
    dwAttributes: DWORD;                               {attribute flags}
    szDisplayName: array [0..MAX_PATH-1] of AnsiChar;  {display name string}
    szTypeName: array [0..79] of AnsiChar;             {file type string}
end;
```

hIcon: A handle to the icon that represents the specified file.

iIcon: The index of the file's icon within the system image list.

dwAttributes: An array of flags that indicates the file's attributes. This member can be one or more of the values from Table 11-37.

szDisplayName: A null-terminated string indicating the display name of the specified file as it appears in the shell.

szTypeName: A null-terminated string describing the type of the specified file.

cbFileInfo: The size, in bytes, of the TSHFileInfo structure pointed to by the psfi parameter. This parameter should be set to SizeOf(TSHFileInfo).

uFlags: An array of flags indicating the type of information to retrieve. This parameter can be one or more of the values from Table 11-38.

Return Value

If the function succeeds, it returns a value greater than zero. Otherwise, it returns zero. See Table 11-38 for descriptions of the return value.

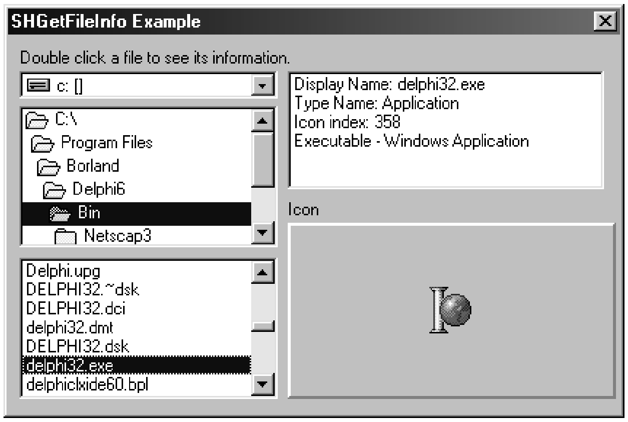
See Also

ExtractAssociatedIcon, ExtractIcon, FindExecutable

Example

Listing 11-12: Retrieving information about a file

```
const
  {Delphi does not define these SHGetFileInfo constants}
  SHGFI_ADDOVERLAYS    = $000000020;     // apply the appropriate overlays
  SHGFI_OVERLAYINDEX   = $000000040;     // get the index of the overlay
  SHGFI_ATTR_SPECIFIED = $000020000;     // get only specified attributes


procedure TForm1.FileListBox1DblClick(Sender: TObject);
var
  FileInfo: TSHFileInfo;        // holds information about a file
  TempIcon: TIcon;              // a temporary icon object
  ExeTypeInfo: DWORD;           // for determining exe type
  ExeType: string;              // holds exe type indicator
begin
  {retrieve information about the selected file}
  SHGetFileInfo(PChar(FileListBox1.Filename), 0, FileInfo, SizeOf(TSHFileInfo),
    SHGFI_DISPLAYNAME or SHGFI_ICON or SHGFI_TYPENAME);

  {see if this is an executable}
  ExeTypeInfo := SHGetFileInfo(PChar(FileListBox1.Filename), 0, FileInfo,
                               SizeOf(TSHFileInfo), SHGFI_EXETYPE);

  {display the information about the selected file}
  with ListBox1.Items do
  begin
    Clear;
    Add('Display Name: ' + FileInfo.szDisplayName);
    Add('Type Name: ' + FileInfo.szTypeName);
    Add('Icon index: ' + IntToStr(FileInfo.iIcon));

    {if this is an executable, display its type}
    if LoWord(ExeTypeInfo) <> 0 then
    begin
      ExeType := Char(Lobyte(Loword(ExeTypeInfo))) +
                 Char(Hibyte(Loword(ExeTypeInfo)));

      if (ExeType = 'NE') or (ExeType = 'PE') then
        Add('Executable - Windows Application');
      if ExeType = 'MZ' then
        Add('Executable - DOS .exe, .com, or .bat');
    end;
  end;

  {create a temporary icon object so we can
   display the file icon in the image object}
  TempIcon := TIcon.Create;
  TempIcon.Handle := FileInfo.hIcon;
  Image1.Picture.Assign(TempIcon);
  TempIcon.Free;
end;
```



Display Name: delphi32.exe
Type Name: Application
Icon index: 358
Executable - Windows Application

Figure 11-7: The file information

Table 11-37: SHGetFileInfo dwFileAttributes values

Value	Description
FILE_ATTRIBUTE_READONLY	The file is read only.
FILE_ATTRIBUTE_HIDDEN	The file is hidden.
FILE_ATTRIBUTE_SYSTEM	The file is a system file.
FILE_ATTRIBUTE_DIRECTORY	The file is a directory folder.
FILE_ATTRIBUTE_ARCHIVE	The file is an archive file.
FILE_ATTRIBUTE_NORMAL	The file does not have any attributes.
FILE_ATTRIBUTE_TEMPORARY	The file is a temporary file.
FILE_ATTRIBUTE_COMPRESSED	The file is compressed.

Table 11-38: SHGetFileInfo uFlags values

Value	Description
SHGFI_ADDOVERLAYS	Apply appropriate overlays to the icon. This flag must be used in conjunction with the SHGFI_ICON flag.
SHGFI_ATTR_SPECIFIED	Used in combination with SHGFI_ATTRIBUTES, this flag indicates that the dwAttributes member of the TSHFileInfo structure contains the specific attributes that are desired.
SHGFI_ATTRIBUTES	Retrieves the attributes of the specified file. These values are copied to the dwAttributes member of the TSHFileInfo structure pointed to by the psfi parameter.
SHGFI_DISPLAYNAME	Retrieves the display name of the specified file. This string is copied to the szDisplayName member of the TSHFileInfo structure pointed to by the psfi parameter.
FOF_RENAMEONCOLLISION	If the pszPath parameter points to an executable file, this flag returns the type of executable file. The low-order word of the return value will contain one of the following values: 0: Non-executable file NE, PE: Windows application (high-order word will not be zero) MZ: DOS executable, .com, or .bat file

Value	Description
SHGFI_EXETYPE (cont.)	PE: Win32 console application (high-order word will be zero) **Note**: This flag may not be used with any other flags.
SHGFI_ICON	Retrieves a handle to the icon that represents the specified file. The icon handle is copied to the hIcon member of the TSHFileInfo structure pointed to by the psfi parameter. The index of the icon in the system image list is copied to the iIcon member of the TSHFileInfo structure pointed to by the psfi parameter. The function returns the handle to the system image list.
SHGFI_ICONLOCATION	Retrieves the name of the file containing the icon that represents the specified file. This filename is copied to the szDisplayName member of the TSHFileInfo structure pointed to by the psfi parameter.
SHGFI_LARGEICON	Retrieves the specified file's large icon. This flag must be used in conjunction with the SHGFI_ICON flag.
SHGFI_LINKOVERLAY	Adds the link overlay graphic to the specified file's icon. This flag must be used in conjunction with the SHGFI_ICON flag.
SHGFI_OPENICON	Retrieves the specified file's open icon. This flag must be used in conjunction with the SHGFI_ICON flag.
SHGFI_OVERLAYINDEX	Returns the index of the overlay icon in the upper eight bits of the iIcon member of the TSHFileInfo structure pointed to by the psfi parameter.
SHGFI_PIDL	Indicates that the pszPath parameter points to an item identifier list instead of a path name.
SHGFI_SELECTED	The file's icon is combined with the system's highlight color. This flag must be used in conjunction with the SHGFI_ICON flag.
SHGFI_SHELLICONSIZE	Retrieves the specified file's icon modified to the size displayed by the shell. This flag must be used in conjunction with the SHGFI_ICON flag.
SHGFI_SMALLICON	Retrieves the specified file's small icon. This flag must be used in conjunction with the SHGFI_ICON flag.
SHGFI_SYSICONINDEX	Retrieves the index of the specified file's icon within the system image list. The icon index is copied to the iIcon member of the TSHFileInfo structure pointed to by the psfi parameter. The function returns the handle to the system image list.
SHGFI_TYPENAME	Retrieves a string describing the specified file's type. This string is copied to the szTypeName of the TSHFileInfo structure pointed to by the psfi parameter.
SHGFI_USEFILEATTRIBUTES	Indicates the function should retrieve information only on files that have the attributes specified by the dwFileAttributes parameter.

StgCreateDocFile ActiveX.pas

Syntax

StgCreateDocfile(	
pwcsName: POleStr;	{path and filename of structured storage file}
grfMode: Longint;	{access mode flags}
reserved: Longint;	{reserved, must be zero}
out stgOpen: IStorage	{returns a pointer to an IStorage interface}
): HResult;	{returns an OLE result}

Description

This function creates a new compound structured storage file.

Note: If the file is created in transaction mode (i.e., the grfMode parameter contains the STGM_TRANSACTED flag), changes to the file are not reflected in the file system until IStorage.Commit is called.

Parameters

pwcsName: A null-terminated string containing the path and filename of the compound structured storage file to create. If no path is given, the file will be created in the current directory. If this parameter is set to NIL, the system creates a file with a unique name.

grfMode: A series of flags indicating the access mode for the compound structured storage file. This can be one or more values from Table 11-39.

reserved: Unused; must be set to zero.

stgOpen: If the function succeeds, this parameter returns a pointer to an IStorage interface representing the created compound structured storage file.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from Table 11-40.

See Also

StgIsStorageFile, StgOpenStorage

Example

Listing 11-13: Creating a structured storage file

```
implementation

const
  {Delphi does not define these constants}
  STGM_NOSNAPSHOT   = $00200000;
  STGM_DIRECT_SWMR  = $00400000;

  STGC_CONSOLIDATE  = 8;
```

```
{$R *.dfm}

type
  {our simple data structure}
  TProprietaryInfo = record
    InfoStr: shortstring;
    InfoInt: Integer;
  end;

procedure TForm1.Button1Click(Sender: TObject);
var
  InfoStream: IStream;                // points to the stream
  RootStore: IStorage;                // points to the root storage object
  ProprietaryInfo: TProprietaryInfo;  // our data structure
  BytesWritten: Longint;              // number of bytes written
begin
  {initialize our data structure with some information}
  ProprietaryInfo.InfoStr := 'Delphi Rocks!';
  ProprietaryInfo.InfoInt := 12345;

  {create a structured storage file on the disk, opened for writing}
  StgCreateDocFile('ExampleFile.tst', STGM_CREATE or STGM_READWRITE or
                   STGM_SHARE_EXCLUSIVE, 0, RootStore);

  {create the stream to hold our data}
  RootStore.CreateStream('InfoBlock1', STGM_CREATE or STGM_READWRITE or
                         STGM_SHARE_EXCLUSIVE, 0, 0, InfoStream);

  {now write the data into the stream}
  InfoStream.Write(@ProprietaryInfo, SizeOf(TProprietaryInfo), @BytesWritten);

  {when our interface objects go out of scope, Delphi frees them automatically,
   and our data is flushed to the disk.}
  ShowMessage('Structured Storage File Created');
  Button1.Enabled := FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  InfoStream: IStream;                // points to the stream
  RootStore: IStorage;                // points to the root storage object
  ProprietaryInfo: TProprietaryInfo;  // our data structure
  BytesRead: Longint;                 // number of bytes read
begin
  {open the structured storage file that was previously created}
  StgOpenStorage('ExampleFile.tst', nil, STGM_READWRITE or
                 STGM_SHARE_EXCLUSIVE, nil, 0, RootStore);

  {open the stream within the structured storage file}
  RootStore.OpenStream('InfoBlock1', nil, STGM_READ or
                       STGM_SHARE_EXCLUSIVE, 0, InfoStream);

  {read the data from the stream into our data structure}
  InfoStream.Read(@ProprietaryInfo, SizeOf(TProprietaryInfo), @BytesRead);
```

```
    {display the data}
    Label1.Caption := ProprietaryInfo.InfoStr;
    Label2.Caption := IntToStr(ProprietaryInfo.InfoInt);
  end;
```

Table 11-39: StgCreateDocFile grfMode values

Value	Description
STGM_READ	Indicates that the object's data can be accessed but not modified. Cannot be combined with STGM_WRITE or STGM_READWRITE.
STGM_WRITE	Indicates that the object's data can be modified, but not accessed. Cannot be combined with STGM_READ or STGM_READWRITE.
STGM_READWRITE	Indicates that the object's data can be both accessed and modified. Cannot be combined with STGM_READ or STGM_WRITE.
STGM_SHARE_DENY_NONE	Indicates that other processes are not denied read or write access to the object when opened. This is the default behavior. Cannot be combined with other STGM_SHAREXXX flags.
STGM_SHARE_DENY_READ	Indicates that other processes cannot open the object with read access. Cannot be combined with other STGM_SHAREXXX flags.
STGM_SHARE_DENY_WRITE	Indicates that other processes cannot open the object with write access. Cannot be combined with other STGM_SHAREXXX flags.
STGM_SHARE_EXCLUSIVE	Indicates that other processes cannot open the object in any mode. Cannot be combined with other STGM_SHAREXXX flags.
STGM_PRIORITY	Provides exclusive access to the most recently committed version of the object and prevents other processes from committing changes to the object while it is opened in priority mode. The STGM_DIRECT and STGM_READ flags must be used with this flag, and the STGM_DELETEONRELEASE flag must not be included.
STGM_CREATE	Indicates that the specified object should be created. Any existing file with the same name is deleted. Cannot be used with STGM_CONVERT.
STGM_CONVERT	Creates the specified object and copies any existing data into a stream named CONTENTS. Cannot be used with STGM_CREATE or STGM_DELETEONRELEASE.
STGM_FAILIFTHERE	Causes the function to fail if the specified object already exists. This is the default behavior.
STGM_DIRECT	Causes any changes to be written as they occur. Cannot be combined with STGM_TRANSACTED.
STGM_TRANSACTED	Any changes to the object are buffered and are only written when the Commit method is called. Call the Revert method to discard any changes since the last call to Commit. Cannot be combined with STGM_DIRECT.
STGM_NOSCRATCH	Must be used with STGM_TRANSACTED. This causes the system to use any unused areas of the original file as a "scratch" storage space for uncommitted changes. Without this flag, a temporary file is created to store uncommitted changes.

Value	Description
STGM_NOSNAPSHOT	Any changes to the file are written to the end of the file, instead of making a temporary copy of the file. When the file is opened using this flag, no other process can open the file without also using this flag. This flag can only be used in combination with STGM_TRANSACTED and only if the STGM_SHARE_EXCLUSIVE and STGM_SHARE_DENY_WRITE flags are not specified. This can lead to very large files.
STGM_SIMPLE	Creates a simple compound structured storage file. It offers efficient performance but does not support substorages, and all streams are a minimum of 4Kb in size.
STGM_DIRECT_SWMR	Provides direct mode for single-writer, multi-reader operations.
STGM_DELETEONRELEASE	Indicates that the file should be automatically destroyed when the object is released. This flag is most often used when creating temporary files. Cannot be used simultaneously with STGM_CONVERT or STGM_PRIORITY.

Table 11-40: StgCreateDocFile return values

Value	Description
STG_E_ACCESSDENIED	Indicates the application does not have permission to access the destination file.
STG_E_FILEALREADYEXISTS	Indicates that a stream of the specified name already exists. This is returned as a result of using the STGM_FAILIFTHERE flag.
STG_E_INSUFFICIENTMEMORY	Operation failed due to insufficient available memory.
STG_E_INVALIDFLAG	The value of the grfMode parameter is not valid.
STG_E_INVALIDFUNCTION	The combination of values specified in grfMode is invalid.
STG_E_INVALIDNAME	The value specified in the pwcsName parameter is invalid.
STG_E_INVALIDPOINTER	The destination storage object pointer is invalid.
STG_E_SHAREVIOLATION	Indicates that another process has the file open and locked.
STG_E_TOOMANYOPENFILES	Indicates that there are too many open files.
STG_S_CONVERTED	Indicates that an existing storage with the same name has been converted into a stream called CONTENTS.

StgIsStorageFile ActiveX.pas

Syntax

```
StgIsStorageFile(
pwcsName: POleStr          {path and filename of structured storage file}
): HResult;                {returns an OLE result}
```

Description

This function indicates if the specified file is a compound structured storage file.

Parameters

pwcsName: A null-terminated string containing the path and filename of the compound structured storage file.

Return Value

If the specified file is a compound structured storage file, it returns S_OK; otherwise, it returns S_FALSE. The function returns STG_E_FILENOTFOUND if the specified file could not be found.

See Also

StgCreateDocFile, StgOpenStorage

Example

Please see Listing 11-4 in the introduction and other examples in this chapter.

StgOpenStorage ActiveX.pas

Syntax

```
StgOpenStorage(
pwcsName: POleStr;           {path and filename of structured storage file}
stgPriority: IStorage;       {IStorage interface open in priority mode}
grfMode: Longint;            {access mode flags}
snbExclude: TSNB;            {elements to be excluded}
reserved: Longint;           {reserved, must be zero}
out stgOpen: IStorage        {returns a pointer to an IStorage interface}
): HResult;                  {returns an OLE result}
```

Description

This function opens a compound structured storage file and returns an IStorage interface for the root storage of the specified file.

Parameters

pwcsName: A null-terminated string containing the path and filename of the compound structured storage file to open.

stgPriority: A pointer to an IStorage interface for a root storage object that has been opened in priority mode. When this function returns, the interface supplied to this parameter is no longer valid, and the application should subsequently use the interface returned in the stgOpen parameter. This parameter can be set to NIL.

grfMode: A series of flags indicating the access mode for the compound structured storage file. This can be one or more values from Table 11-41.

snbExclude: A pointer to a series of element names that are excluded when the storage file is opened (TSNB is defined as a PWideChar, so this is simply a pointer to a null-terminated string). When the storage is opened, any excluded streams are set to a length of zero and any excluded substorages have all of their elements removed. It is typically used when reading storages opened in priority mode. The application is responsible for rewriting the contents of any excluded items before changes are committed. This parameter can be set to NIL.

reserved: Reserved; must be set to zero.

stgOpen: If the function succeeds, this parameter returns a pointer to an IStorage interface representing the created compound structured storage file.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE failure result code from Table 11-42.

See Also

StgCreateDocFile, StgIsStorageFile

Example

Please see Listing 11-13 under StgCreateDocFile and other examples in this chapter.

Table 11-41: StgOpenStorage grfMode values

Value	Description
STGM_READ	Indicates that the object's data can be accessed but not modified. Cannot be combined with STGM_WRITE or STGM_READWRITE.
STGM_WRITE	Indicates that the object's data can be modified but not accessed. Cannot be combined with STGM_READ or STGM_READWRITE.
STGM_READWRITE	Indicates that the object's data can be both accessed and modified. Cannot be combined with STGM_READ or STGM_WRITE.
STGM_SHARE_DENY_NONE	Indicates that other processes are not denied read or write access to the object when opened. This is the default behavior. Cannot be combined with other STGM_SHAREXXX flags.
STGM_SHARE_DENY_READ	Indicates that other processes cannot open the object with read access. Cannot be combined with other STGM_SHAREXXX flags.
STGM_SHARE_DENY_WRITE	Indicates that other processes cannot open the object with write access. Cannot be combined with other STGM_SHAREXXX flags.
STGM_SHARE_EXCLUSIVE	Indicates that other processes cannot open the object in any mode. Cannot be combined with other STGM_SHAREXXX flags.
STGM_PRIORITY	Provides exclusive access to the most recently committed version of the object and prevents other processes from committing changes to the object while it is opened in priority mode. The STGM_DIRECT and STGM_READ flags must be used with this flag.
STGM_DIRECT	Causes any changes to be written as they occur. Cannot be combined with STGM_TRANSACTED.
STGM_TRANSACTED	Any changes to the object are buffered and are only written when the Commit method is called. Call the Revert method to discard any changes since the last call to Commit. Cannot be combined with STGM_DIRECT.
STGM_NOSCRATCH	Must be used with STGM_TRANSACTED. This causes the system to use any unused areas of the original file as a "scratch" storage space for uncommitted changes. Without this flag, a temporary file is created to store uncommitted changes.

Value	Description
STGM_NOSNAPSHOT	Any changes to the file are written to the end of the file, instead of making a temporary copy of the file. When the file is opened using this flag, no other process can open the file without also using this flag. This flag can only be used in combination with STGM_TRANSACTED and only if the STGM_SHARE_EXCLUSIVE and STGM_SHARE_DENY_WRITE flags are not specified. This can lead to very large files.
STGM_SIMPLE	Opens the simple compound structured storage file in simple mode. It offers efficient performance but does not support substorages, and all streams are a minimum of 4Kb in size.
STGM_DIRECT_SWMR	Provides direct mode for single-writer, multi-reader operations.

Table 11-42: StgOpenStorage return values

Value	Description
STG_E_ACCESSDENIED	Indicates the application does not have permission to access the destination file.
STG_E_FILEALREADYEXISTS	Indicates that the specified file exists but is not a structured storage file.
STG_E_FILENOTFOUND	Indicates that the specified file does not exist.
STG_E_INSUFFICIENTMEMORY	Operation failed due to insufficient available memory.
STG_E_INVALIDFLAG	The value of the grfMode parameter is not valid.
STG_E_INVALIDFUNCTION	The combination of values specified in grfMode is invalid.
STG_E_INVALIDNAME	The value specified in the pwcsName parameter is invalid.
STG_E_INVALIDPOINTER	The destination storage object pointer is invalid.
STG_E_NOTSIMPLEFORMAT	Indicates that the structured storage file was not created in the simple format.
STG_E_OLDDLL	Indicates that the COM objects used to open the storage file are older than the COM objects used to create it.
STG_E_OLDFORMAT	Indicates that the structured storage file is in a beta format and is not supported.
STG_E_PATHNOTFOUND	The specified file path could not be found.
STG_E_SHAREVIOLATION	Indicates that another process has the file open and locked.
STG_E_TOOMANYOPENFILES	Indicates that there are too many open files.

Chapter 12

Shell Folder Functions

The Windows user interface employs a specific metaphor to present the user with a mechanism for storing, sorting, and retrieving various documents and objects within their system: the folder. Folders are an inseparable part of the Windows Explorer interface and are used for the organization of everything from documents and files to more abstract concepts, such as remote machines. Consequently, the Windows Shell API contains many functions for manipulating and retrieving information about folders.

Note: Many of the functions documented in this chapter require constants and function prototypes that do not exist in the shipping Delphi source code at the time of publication. In order to make these examples compile, you'll need the file ShellExtra.pas, located in the root directory for this chapter on the companion CD. This will not be copied into the individual directories containing source code that require this file. Rather, it will be indirectly referenced by that source code, so make sure you duplicate the file structure on your hard drive when you copy and recompile these examples.

Browsing for Folders

Some applications need to present the user with a standard dialog box for the purpose of selecting a folder, such as specifying the starting point for a file search. The Windows shell offers the SHBrowseForFolder function that displays such a dialog box. This function has changed somewhat since its initial implementation under Windows 95. On systems prior to Windows 2000, the Browse for Folder dialog box offered basic functionality and did not make use of some of the extended shell namespace capabilities. Now, on Windows 2000 and later, the SHBrowseForFolder has extended flags that allow the function to present an updated interface. This new interface includes a resizable dialog box, drag-and-drop capability within the dialog box, shortcut menus, creation of new folders, display of folders or files, deletion of folders or files, and many other features users are accustomed to seeing when dealing with a presentation of the shell's namespace. To display the Browse for Folder dialog box using the new interface, use the following code:

Listing 12-1: The new Browse for Folder interface

```
implementation

{$R *.dfm}

uses
  ShellExtra;

procedure TForm1.Button1Click(Sender: TObject);
var
  IDList: PItemIDList;                  // an item identifier list
  BrowseInfo: TBrowseInfo;              // the browse info structure
  DispName: array[0..MAX_PATH] of Char; // display name of the selected item
  ShellMalloc: IMalloc;                 // shell's memory allocator
begin
  {setup the browse for folder information}
  BrowseInfo.hwndOwner := Self.Handle;
  BrowseInfo.pidlRoot := nil;
  BrowseInfo.pszDisplayName := DispName;
  BrowseInfo.lpszTitle := 'Select a File or Folder...';
  BrowseInfo.ulFlags := BIF_NEWDIALOGSTYLE;  // use the new user interface
  BrowseInfo.lpfn := nil;
  BrowseInfo.lParam := 0;

  {when using the new UI, we must call CoInitialize before the call to
   SHBrowseForFolder}
  CoInitialize(nil);

  {show the browse for folder dialog}
  IDList := SHBrowseForFolder(BrowseInfo);

  {every call to CoInitialize must be matched with a call to CoUninitialize}
  CoUninitialize;

  {display the name of the chosen item}
  Label2.Caption := BrowseInfo.pszDisplayName;

  {dispose of the item identifier list}
  SHGetMalloc(ShellMalloc);
  ShellMalloc.Free(IDList);
end;
```

Note: The SHBrowseForFolder function returns an item identifier list, which the caller is responsible for freeing. This is accomplished by calling SHGetMalloc to retrieve a pointer to the system's IMalloc interface and using IMalloc.Free.

Item Identifier Lists

Each object in a shell's namespace (such as files, folders, servers, workgroups, printers, etc.) is uniquely identified by an object called an item identifier. An item identifier is a variable length binary data structure whose content and format are known only to the creator of the item identifier. Item identifiers can be retrieved from a number of the file management functions.

The organization of the shell's namespace is analogous to the organization of files in a directory structure. The root of the shell's namespace is the Desktop, and every object under it can potentially contain other objects. An object's item identifier is unique and meaningful only within the context of its parent. Since container objects have an item identifier that uniquely identifies it within its parent container, any object can be uniquely identified by a list of item identifiers. Therefore, an item identifier list uniquely identifies an object within the shell's namespace by tracing a path from it to the desktop. Many of the file management and manipulation functions use item identifier lists to specify files or folders.

Item identifier lists are commonly used with shell Component Object Model (COM) objects. They are also used with several API functions that deal with the shell's namespace. While it is easy enough to identify an object within the shell's namespace that has a physical representation in the file system by using a qualified path (i.e., the StartMenu folder), some paths can be renamed, moved, or otherwise modified by the user. Therefore, it is impossible to guarantee that a specific, known directory will be in a specific place or even have a specific name. If your application uses an item identifier list to point to an object in the namespace, retrieving the physical directory by using API calls when necessary, you can avoid this pitfall.

Delphi vs. the Windows API

To deal with the shell namespace and shell folders is to deal with the Windows API itself. There are no simple objects within Delphi that wrap these functions to make them easier to use, not that any of them are overly complex. Perhaps the most complex function is the SHBrowseForFolder API function. While there is not a standard object encapsulating this function that is installed with Delphi at the time of publication, there are many variations of freeware and shareware that offer such encapsulation. However, all of these functions are relatively straightforward to use and can enrich your Delphi applications in many ways.

Shell Folder Functions

The following shell folder functions are covered in this chapter.

Table 12-1: Shell folder functions

Function	Description
SHBrowseForFolder	Creates a dialog box allowing the user to choose a shell folder.

Function	Description
SHEmptyRecycleBin	Empties the recycle bin on a specified drive.
SHGetFolderLocation	Retrieves the location of a shell namespace object as an item identifier list.
SHGetFolderPath	Retrieves a path name for a specified folder.
SHGetPathFromIDList	Retrieves a path name from an item identifier list.
SHGetSettings	Retrieves the status of various shell options.
SHGetSpecialFolderLocation	Retrieves the location of a shell namespace object as an item identifier list.
SHGetSpecialFolderPath	Retrieves a path name for a specified folder.
SHQueryRecycleBin	Retrieves the total size and number of files in the recycle bin.

SHBrowseForFolder ShlObj.pas

Syntax

```
SHBrowseForFolder(
var lpbi: TBrowseInfo        {a pointer to a TBrowseInfo data structure}
): PItemIDList;              {returns a pointer to an item identifier list}
```

Description

This function displays a dialog box allowing the user to choose a shell folder and returns an item ID list representing then selected folder. The application is responsible for freeing this item ID list by using the shell memory allocator's IMalloc.Free method (see SHGetMalloc for more details).

Note: The caller is responsible for freeing the item identifier list. This must be done via the IMalloc interface returned from SHGetMalloc.

Parameters

lpbi: A pointer to a TBrowseInfo structure. This structure holds information used to display the dialog box and receives information from the dialog box indicating the user's choice. This structure is defined as:

```
TBrowseInfo = packed record
    hwndOwner: HWND;                 {a handle to a window}
    pidlRoot: PItemIDList;           {a pointer to an item identifier list}
    pszDisplayName: PAnsiChar;       {a pointer to a string}
    lpszTitle: PAnsiChar;            {a pointer to a string}
    ulFlags: UINT;                   {control flags}
    lpfn: TFNBFFCallBack;            {the address to a callback function}
    lParam: LPARAM;                  {an application-defined value}
    iImage: Integer;                 {a system image list image index}
end;
```

hwndOwner: A handle to the window that owns the dialog box.

pidlRoot: A pointer to an item identifier list specifying the root folder from which the user starts the browse. If this member is NIL, the root of the namespace is used as the starting point.

pszDisplayName: A pointer to a buffer that receives a null-terminated string containing the display name of the selected folder. The size of this buffer is assumed to be MAX_PATH bytes.

lpszTitle: A pointer to a null-terminated string containing the text displayed in the caption of the dialog box.

ulFlags: An array of flags specifying the types of folders listed and other options. This member can be one or more of the values from Table 12-2.

lpfn: A pointer to a callback function. This function is called whenever a user action generates an event in the dialog box, such as selecting a folder. This member can be set to NIL. The callback function syntax is described below.

lParam: An application-defined value that is passed to the callback function if one is defined.

iImage: Receives an index into the system image list of the image that represents the selected folder.

Return Value

If the function succeeds, it returns a pointer to an item identifier list specifying the chosen folder. The location of the folder is relative to the root of the namespace. If the function failed, or the user chose the Cancel button, the function returns NIL.

Callback Syntax

```
BrowseCallbackProc(
hWnd: HWND;              {a handle to the dialog box window}
uMsg: UINT;              {a dialog box event message}
lParam: LPARAM;          {a message-specific value}
lpData: LPARAM           {an application-defined value}
): Integer;              {returns an integer value}
```

Description

The callback function is run whenever the user causes an event to take place in the Browse for Folder dialog box. This callback function can perform any desired task.

Parameters

hWnd: A handle to the dialog box window. The callback function can use this parameter to send a special message to the dialog box window. The available messages are listed in Table 12-3.

uMsg: A value indicating the type of event that has occurred. This parameter can be one value from Table 12-4.

lParam: A message-specific value. This value is dependent on the uMsg parameter.

lpData: The application-defined value that was passed in the lParam member of the TBrowseInfo structure.

Return Value

The callback function should always return a zero.

See Also

FindExecutable, ShellExecute, ShellExecuteEx, SHFileOperation, SHGetMalloc

Example

Listing 12-2: Browsing for a folder

```
  {the callback function used by the browse for folder dialog
   box. notice the export directive.}
  function BrowseCallback(hWnd: HWND; uMsg: UINT; lParam: LPARAM;
                          lpData: LPARAM): Integer; stdcall; export;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var
  IDList: PItemIDList;                      // an item identifier list
  BrowseInfo: TBrowseInfo;                  // the browse info structure
  PathName: array[0..MAX_PATH] of char;     // the path name
  DisplayName: array[0..MAX_PATH] of char;  // the file display name
  ShellMalloc: IMalloc;                     // shell's memory allocator
begin
  {initialize the browse information structure}
  BrowseInfo.hwndOwner      := Form1.Handle;
  BrowseInfo.pidlRoot       := nil;
  BrowseInfo.pszDisplayName := DisplayName;
  BrowseInfo.lpszTitle      := 'Choose a file or folder';
  BrowseInfo.ulFlags        := BIF_STATUSTEXT;          // display a status line
  BrowseInfo.lpfn           := @BrowseCallback;
  BrowseInfo.lParam         := 0;

  {show the browse for folder dialog box}
  IDList := SHBrowseForFolder(BrowseInfo);

  {retrieve the path from the item identifier list that was returned}
  SHGetPathFromIDList(IDList, @PathName);

  {display the pathname and display name of the selected folder}
  Label2.Caption := PathName;
  Label4.Caption := BrowseInfo.pszDisplayName;

  {dispose of the item identifier list}
  SHGetMalloc(ShellMalloc);
  ShellMalloc.Free(IDList);
```

```
end;

{this callback function is called whenever an action takes
 place inside the browse for folder dialog box}
function BrowseCallback(hWnd: HWND; uMsg: UINT; lParam: LPARAM;
                        lpData: LPARAM): Integer;
var
  PathName: array[0..MAX_PATH] of Char;  // holds the path name
begin
  {if the selection in the browse for folder dialog box has changed...}
  if uMsg=BFFM_SELCHANGED then
  begin
    {...retrieve the path name from the item identifier list}
    SHGetPathFromIDList(PItemIDList(lParam), @PathName);

    {display this path name in the status line of the dialog box}
    SendMessage(hWnd, BFFM_SETSTATUSTEXT, 0, Longint(PChar(@PathName)));

    Result := 0;
  end;
end;
```

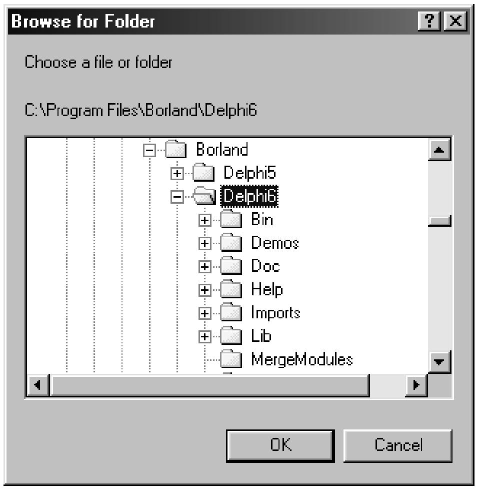


Figure 12-1: The Browse for Folder dialog box

Table 12-2: SHBrowseForFolder lpbi.uFlags values

Value	Description
BIF_BROWSEFORCOMPUTER	Allows the user to select only computers.
BIF_BROWSEFORPRINTER	Allows the user to select only printers.
BIF_BROWSEINCLUDEFILES	The browse dialog will display files as well as folders.
BIF_BROWSEINCLUDEURLS	**Windows 2000 and later**: The browse dialog will display URLs. The URL is displayed only if the folder containing the selected item supports URLs. Requires the BIF_NEWDIALOGSTYLE and BIF_BROWSE-INCLUDEFILES flags.

Value	Description
BIF_DONTGOBELOWDOMAIN	The dialog box will not contain network folders below the domain level.
BIF_EDITBOX	Includes an edit box in the dialog, allowing the user to type in the name of the desired item.
BIF_NEWDIALOGSTYLE	**Windows 2000 and later**: This flag causes the dialog box to display using an updated interface, including window sizing, drag-and-drop capability, shortcut menus, new folders, and other extended functionality. **Note**: When this flag is used, the application must call CoInitialize before the call to SHBrowseForFolder.
BIF_RETURNFSANCESTORS	Allows the user to select only file system ancestors.
BIF_RETURNONLYFSDIRS	Allows the user to select only file system directories.
BIF_SHAREABLE	**Windows 2000 or later**: Allows the dialog to display shareable resources on remote systems. Requires the BIF_NEWDIALOGSTYLE flag.
BIF_STATUSTEXT	Includes a status line in the dialog box. The callback function can send a message to the dialog box specifying what to display on this line.
BIF_UAHINT	**Windows 2000 or later**: Adds a usage hint to the dialog box. Requires the BIF_NEWDIALOGSTYLE flag.
BIF_USENEWUI	**Windows 2000 or later**: Causes the dialog box to display using an updated interface. This flag is equivalent to combining the BIF_NEWDIALOGSTYLE and BIF_EDITBOX flags. **Note**: When this flag is used, the application must call CoInitialize before the call to SHBrowseForFolder.
BIF_VALIDATE	If the BIF_EDITBOX flag is included, this flag causes a BFFM_VALIDATEFAILED message to be sent to the callback procedure if the user types an invalid name into the edit box.

Table 12-3: BrowseCallbackProc Browse for Folder dialog box messages

Value	Description
BFFM_ENABLEOK	Enables the OK button if the wParam parameter of the message contains a non-zero value. If the wParam parameter contains a zero, the OK button is disabled.
BFFM_SETSELECTION	Selects a specific folder. If the wParam parameter of the message contains TRUE, the lParam parameter must contain a pointer to a string describing the path of the folder. If the wParam parameter is FALSE, the lParam parameter must contain a pointer to an item identifier list specifying the selected folder.

Value	Description
BFFM_SETSTATUSTEXT	Sets the text of the status line in the dialog box. The lParam parameter of the message must contain a pointer to a null-terminated string for the status line. This message is only valid if the BIF_STATUSTEXT flag was specified in the ulFlags member of the TBrowseInfo structure.

Table 12-4: BrowseCallbackProc uMsg values

Value	Description
BFFM_INITIALIZED	The Browse for Folder dialog box has finished initializing. The lParam parameter contains zero.
BFFM_SELCHANGED	The user has selected a folder. The lParam parameter contains a pointer to an item identifier list specifying the chosen folder.
BFFM_VALIDATEFAILED	Indicates that the user typed an invalid name into the edit box. The lpData parameter contains a pointer to the string containing the name. Return a zero to close the dialog or a non-zero value to keep the dialog open.

SHEmptyRecycleBin ShellExtra.pas

Syntax

```
SHEmptyRecycleBin(
hWnd: HWND;                 {parent window handle for dialog boxes}
pszRootPath: PAnsiChar;     {root drive of recycle bin}
dwFlags: DWORD              {option flags}
): HResult;                 {returns OLE result code}
```

Description

This function empties the recycle bin on the drive specified by the pszRootPath parameter. Various flags specified in the dwFlags parameter control options available to the user and progress reports displayed on screen.

Note: This function requires the Internet Explorer version 4.0 Desktop Update to be installed (shell32.dll version 4.71 or higher).

Parameters

hWnd: A handle to a window used as the parent window for any dialog boxes that might be displayed by this function. This parameter can be set to zero (indicating no parent window).

pszRootPath: A null-terminated string indicating the path of the root drive containing the desired recycle bin. This string can contain a fully qualified path (i.e., C:\Windows\System) or can be set to NIL. If NIL is specified, all recycle bins on all drives are emptied. This string cannot be longer than MAX_PATH characters.

dwFlags: A combination of values controlling the behavior of this function. This value may be a combination of one or more values from the following table.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE error code.

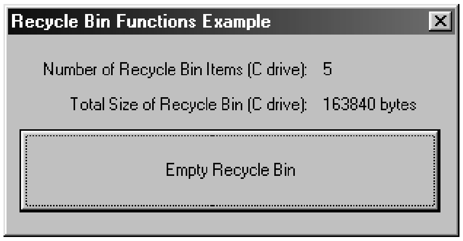
See Also

SHFileOperation, SHQueryRecycleBin

Example

Listing 12-3: Manipulating the recycle bin

```
implementation

{$R *.dfm}

uses
  ShellExtra;

procedure TForm1.CheckRecycleBin;
var
  RBInfo: TSHQueryRBInfo;   // recycle bin information structure
begin
  {initialize the query structure}
  FillChar(RBInfo, SizeOf(TSHQueryRBInfo), #0);
  RBInfo.cbSize := SizeOf(TSHQueryRBInfo);

  {retrieve recycle bin information for drive C}
  if SHQueryRecycleBin('c:\', @RBInfo) = S_OK then
  begin
    Label3.Caption := IntToStr(RBInfo.i64NumItems);
    Label4.Caption := IntToStr(RBInfo.i64Size) + ' bytes';
  end
  else
    raise Exception.Create('SHQueryRecycleBin failed!');
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  {initialize our status when the form is created}
  CheckRecycleBin;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  {empty the recycle bin on drive C}
  SHEmptyRecycleBin(Handle, 'C:\', 0);

  {now update our status}
  CheckRecycleBin;
end;
```



Number of Recycle Bin Items (C drive): 5
Total Size of Recycle Bin (C drive): 163840 bytes

Figure 12-2: Querying the recycle bin

Table 12-5: SHEmptyRecycleBin dwFlags values

Value	Description
SHERB_NOCONFIRMATION	Suppresses the display of a confirmation dialog box.
SHERB_NOPROGRESSUI	Suppresses the display of a deletion progress dialog box.
SHERB_NOSOUND	Suppresses any sounds played when the operation is completed.

SHGetFolderLocation ShellExtra.pas

Syntax

```
SHGetFolderLocation(
hWnd: HWND;               {a handle to a window}
csidl: Integer;           {a folder location flag}
hToken: THandle;          {user access token}
dwReserved: DWORD;        {reserved}
var ppidl: PItemIDList    {a pointer to an item identifier list}
): HResult;               {returns an OLE result}
```

Description

This function retrieves an item identifier list specifying the location of the special folder. Note that only those folders that are registered under the key HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Explorer\ Shell Folders will return an item identifier list specifying a file system folder that SHGetPathFromIDList can use to retrieve a physical path name.

Note: The caller is responsible for freeing the item identifier list. This must be done via the IMalloc interface returned from SHGetMalloc.

Note: This function is available only on Windows 2000 and later systems.

Parameters

hWnd: A handle to the owning window for dialog or message boxes.

csidl: A flag indicating the folder for which to retrieve the location. This parameter can be one value from the following table.

hToken: An access token used to indicate a specific user. This parameter is typically set to zero (for all Windows versions prior to Windows 2000, this value must be set to zero). A value of –1 indicates the default user.

dwReserved: Reserved.

ppidl: A pointer to an item identifier list that specifies the indicated folder's location relative to the root of the namespace.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE-defined error result.

See Also

SHBrowseForFolder, SHGetFileInfo, SHGetFolderPath, SHGetPathFromIDList, SHGetSpecialFolderLocation

Example

Listing 12-4: Retrieving the location of the Start menu folder

```
uses
  ShellExtra;

{******************************************************************************
  NOTE: The following example works only under Windows 2000 and later!!
******************************************************************************}

procedure TForm1.Button1Click(Sender: TObject);
var
  IDList: PItemIDList;                    // item id list to retrieve
  PathName: array[0..MAX_PATH] of Char;   // stores the folder path
  ShellMalloc: IMalloc;                   // shell's memory allocator
begin
  {retrieve the item id list for the start menu item}
  SHGetFolderLocation(Handle, CSIDL_STARTMENU, 0, 0, IDList);

  {retrieve the path from this id list}
  SHGetPathFromIDList(IDList, @PathName);

  {display the path}
  Label2.Caption := PathName;

  {dispose of the item identifier list}
  SHGetMalloc(ShellMalloc);
  ShellMalloc.Free(IDList);
end;
```

Table 12-6: SHGetFolderLocation csidl values

Value	Description
CSIDL_ADMINTOOLS	**Windows 2000 and later**: Retrieves the directory used to store administrative tools. This directory is specific to a user (i.e., <user name>\Start Menu\Programs\Administrative Tools).
CSIDL_ALTSTARTUP	Retrieves the directory of the user's non-localized Startup program group.
CSIDL_APPDATA	Retrieves the directory used to store application-specific data. This directory is specific to a user (i.e., Documents and Settings\<user name>\Application Data).
CSIDL_BITBUCKET	Retrieves the location of the recycle bin. This directory is not in the registry and has hidden and system attributes to prevent the user from moving or deleting it.
CSIDL_COMMON_ADMINTOOLS	**Windows 2000 and later**: Retrieves the directory used to store administrative tools available to any user (i.e., All Users\Start Menu\Programs\Administrative Tools).
CSIDL_COMMON_ALTSTARTUP	**Windows NT and later**: Retrieves the directory of the non-localized Startup program group for all users.
CSIDL_COMMON_APPDATA	**Windows 2000 and later**: Retrieves the directory used to store application-specific data for all users of the computer (i.e., Documents and Settings\All Users\Application Data).
CSIDL_COMMON_DESKTOPDIRECTORY	**Windows NT and later**: Retrieves the directory containing files and folders appearing on the desktop for all users (i.e., Documents and Settings\All Users\Desktop).
CSIDL_COMMON_DOCUMENTS	Retrieves the directory used to store document files for all users of the computer (i.e., Documents and Settings\All Users\Documents).
CSIDL_COMMON_FAVORITES	**Windows NT and later**: Retrieves the directory used to store Favorite links for all users of the computer.
CSIDL_COMMON_PROGRAMS	**Windows NT and later**: Retrieves the directory used to store the directories for the common program groups appearing in the Start menu for all users of the computer (i.e., Documents and Settings\All Users\Start Menu\Programs).
CSIDL_COMMON_STARTMENU	**Windows NT and later**: Retrieves the directory used to store the programs and directories appearing in the Start menu for all users (i.e., Documents and Settings\All Users\Start Menu).
CSIDL_COMMON_STARTUP	**Windows NT and later**: Retrieves the directory used to store the programs appearing in the Startup folder for all users (i.e., Documents and Settings\All Users\Start Menu\Programs\Startup).
CSIDL_CONTROLS	Retrieves the virtual folder containing the icons for control panel applets.
CSIDL_COOKIES	Retrieves the directory used to store Internet cookies.

Value	Description
CSIDL_DESKTOP	Retrieves the virtual folder for the root of the namespace, the Windows desktop.
CSIDL_DESKTOPDIRECTORY	Retrieves the directory used to store files and folders appearing on the desktop (i.e., Documents and Settings\<user name>\Desktop).
CSIDL_DRIVES	Retrieves the My Computer virtual folder, which contains storage devices, printers, the control panel, and may contain mapped network drives.
CSIDL_FAVORITES	Retrieves the directory used to store favorite links (i.e., Documents and Settings\<user name>\Favorites).
CSIDL_FLAG_CREATE	Used to force the creation of a directory if it does not exist. This is the only value that can be combined using a Boolean OR with any other value in this table.
CSIDL_FONTS	Retrieves the virtual folder containing fonts.
CSIDL_HISTORY	Retrieves the directory used to store Internet history items.
CSIDL_INTERNET_CACHE	Retrieves the directory used to store temporary Internet files.
CSIDL_LOCAL_APPDATA	**Windows 2000 and later**: Retrieves the directory used as a data repository for local (non-roaming) applications.
CSIDL_MYPICTURES	**Windows 2000 and later**: Retrieves the directory used as a common storage target for pictures. This directory is specific to a user (i.e., Documents and Settings\<user name>\My Documents\My Pictures).
CSIDL_NETHOOD	Retrieves the directory used to store the link files appearing as objects under the My Network Places virtual folder (i.e., Documents and Settings\<user name>\NetHood).
CSIDL_NETWORK	Retrieves the network neighborhood virtual folder representing the top level of the network hierarchy.
CSIDL_PERSONAL	Retrieves the directory used to store document files. This directory is specific to a user (i.e., Documents and Settings\<user name>\My Documents).
CSIDL_PRINTERS	Retrieves the virtual folder containing installed printers.
CSIDL_PRINTHOOD	Retrieves the directory used to store the link files appearing as objects under the Printers virtual folder (i.e., Documents and Settings\<user name>\PrintHood).
CSIDL_PROGRAM_FILES	Retrieves the directory of the Program Files folder.
CSIDL_PROGRAM_FILES_COMMON	**Windows 2000 and later**: Retrieves the directory used to store executables and other components that are shared across applications (i.e., Program Files\Common).
CSIDL_PROGRAMS	Retrieves the directory containing the directories of the program groups appearing in the Start menu (i.e., Documents and Settings\<user name>\Start Menu\Programs).

Value	Description
CSIDL_RECENT	Retrieves the directory used to store links for the user's most recently used documents (i.e., Documents and Settings\<user name>\Recent).
CSIDL_SENDTO	Retrieves the directory containing the Send To menu items (i.e., Documents and Settings\<user name>\SendTo).
CSIDL_STARTMENU	Retrieves the directory used to store the programs and directories appearing in the Start menu (i.e., Documents and Settings\<user name>\Start Menu).
CSIDL_STARTUP	Retrieves the directory used to store the programs appearing in the Startup (i.e., Documents and Settings\<user name>\Start Menu\Programs\Startup).
CSIDL_SYSTEM	Retrieves the System directory.
CSIDL_TEMPLATES	Retrieves the directory used to store template files (i.e., Documents and Settings\<user name>\Templates).
CSIDL_WINDOWS	Retrieves the Windows directory.

SHGetFolderPath SHFolder.pas

Syntax

```
SHGetFolderPath(
hwnd: HWND;                {a handle to a window}
csidl: Integer;            {a folder location flag}
hToken: THandle;           {user access token}
dwFlags: DWORD;            {path type flags}
pszPath: PAnsiChar         {null-terminated path string}
): HRESULT;                {returns an OLE result code}
```

Description

This function retrieves the physical path for the specified folder.

Note: Only folders that physically exist in the file system are valid; the function fails if a virtual folder (i.e., Printers, Dial-up Networking) is specified.

Parameters

hwnd: A handle to a window used as an owner window for any dialog boxes spawned by this function. This parameter may be zero.

csidl: A flag indicating the folder for which to retrieve the location. This parameter can be one value from Table 12-7.

hToken: An access token used to indicate a specific user. This parameter is typically set to zero (for all Windows versions prior to Windows 2000, this value must be set to zero). A value of –1 indicates the default user.

dwFlags: A flag indicating the type of path to return if the physical directory represented by the value in csidl can be moved or renamed by the user. This parameter can be one value from Table 12-8.

pszPath: A pointer to a null-terminated string that receives the path. This string should be of MAX_LENGTH length.

Return Value

If the function succeeds, it returns S_OK. S_FALSE is returned if the function succeeded but the specified folder does not exist. Otherwise, this function returns E_INVALIDARG or a standard OLE error code.

See Also

SHAddToRecentDocs, SHGetFolderLocation, SHGetPathFromIDList, SHGetSpecialFolderPath

Example

Listing 12-5: Retrieving the path for the My Documents folder

```
implementation

{$R *.dfm}

uses
  ShellExtra;

procedure TForm1.Button1Click(Sender: TObject);
var
  strPath: array[0..MAX_PATH] of Char;    // receives the path string
begin
  {retrieve the path for the My Documents folder}
  SHGetFolderPath(Handle, CSIDL_PERSONAL, 0, SHGFP_TYPE_CURRENT, strPath);

  {display the path}
  Label2.Caption := strPath;
end;
```

Table 12-7: SHGetFolderPath csidl values

Value	Description
CSIDL_ADMINTOOLS	**Windows 2000 and later**: Retrieves the directory used to store administrative tools. This directory is specific to a user (i.e., <user name>\Start Menu\Programs\Administrative Tools).
CSIDL_ALTSTARTUP	Retrieves the directory of the user's non-localized Startup program group.
CSIDL_APPDATA	Retrieves the directory used to store application-specific data. This directory is specific to a user (i.e., Documents and Settings\<user name>\Application Data).

Value	Description
CSIDL_COMMON_ADMINTOOLS	**Windows 2000 and later**: Retrieves the directory used to store administrative tools available to any user (i.e., All Users\Start Menu\Programs\Administrative Tools).
CSIDL_COMMON_ALTSTARTUP	**Windows NT and later**: Retrieves the directory of the non-localized Startup program group for all users.
CSIDL_COMMON_APPDATA	**Windows 2000 and later**: Retrieves the directory used to store application-specific data for all users of the computer (i.e., Documents and Settings\All Users\Application Data).
CSIDL_COMMON_DESKTOPDIRECTORY	**Windows NT and later**: Retrieves the directory containing files and folders appearing on the desktop for all users (i.e., Documents and Settings\All Users\Desktop).
CSIDL_COMMON_DOCUMENTS	Retrieves the directory used to store document files for all users of the computer (i.e., Documents and Settings\All Users\Documents).
CSIDL_COMMON_FAVORITES	**Windows NT and later**: Retrieves the directory used to store Favorite links for all users of the computer.
CSIDL_COMMON_PROGRAMS	**Windows NT and later**: Retrieves the directory used to store the directories for the common program groups appearing in the Start menu for all users of the computer (i.e., Documents and Settings\All Users\Start Menu\Programs).
CSIDL_COMMON_STARTMENU	**Windows NT and later**: Retrieves the directory used to store the programs and directories appearing in the Start menu for all users (i.e., Documents and Settings\All Users\Start Menu).
CSIDL_COMMON_STARTUP	**Windows NT and later**: Retrieves the directory used to store the programs appearing in the Startup folder for all users (i.e., Documents and Settings\All Users\Start Menu\Programs\Startup).
CSIDL_COOKIES	Retrieves the directory used to store Internet cookies.
CSIDL_DESKTOPDIRECTORY	Retrieves the directory used to store files and folders appearing on the desktop (i.e., Documents and Settings\<user name>\Desktop).
CSIDL_FAVORITES	Retrieves the directory used to store favorite links (i.e., Documents and Settings\<user name>\Favorites).
CSIDL_FLAG_CREATE	Used to force the creation of a directory if it does not exist. This is the only value that can be combined using a Boolean OR with any other value in this table.
CSIDL_HISTORY	Retrieves the directory used to store Internet history items.
CSIDL_INTERNET_CACHE	Retrieves the directory used to store temporary Internet files.
CSIDL_LOCAL_APPDATA	**Windows 2000 and later**: Retrieves the directory used as a data repository for local (non-roaming) applications.

Value	Description
CSIDL_MYPICTURES	**Windows 2000 and later**: Retrieves the directory used as a common storage target for pictures. This directory is specific to a user (i.e., Documents and Settings\<user name>\My Documents\My Pictures).
CSIDL_NETHOOD	Retrieves the directory used to store the link files appearing as objects under the My Network Places virtual folder (i.e., Documents and Settings\<user name>\NetHood).
CSIDL_PERSONAL	Retrieves the directory used to store document files. This directory is specific to a user (i.e., Documents and Settings\<user name>\My Documents).
CSIDL_PRINTHOOD	Retrieves the directory used to store the link files appearing as objects under the Printers virtual folder (i.e., Documents and Settings\<user name>\PrintHood).
CSIDL_PROGRAM_FILES	Retrieves the directory of the Program Files folder.
CSIDL_PROGRAM_FILES_COMMON	**Windows 2000 and later**: Retrieves the directory used to store executables and other components that are shared across applications (i.e., Program Files\Common).
CSIDL_PROGRAMS	Retrieves the directory containing the directories of the program groups appearing in the Start menu (i.e., Documents and Settings\<user name>\Start Menu\Programs).
CSIDL_RECENT	Retrieves the directory used to store links for the user's most recently used documents (i.e., Documents and Settings\<user name>\Recent).
CSIDL_SENDTO	Retrieves the directory containing the Send To menu items (i.e., Documents and Settings\<user name>\SendTo).
CSIDL_STARTMENU	Retrieves the directory used to store the programs and directories appearing in the Start menu (i.e., Documents and Settings\<user name>\Start Menu).
CSIDL_STARTUP	Retrieves the directory used to store the programs appearing in the Startup (i.e., Documents and Settings\<user name>\Start Menu\Programs\Startup).
CSIDL_SYSTEM	Retrieves the System directory.
CSIDL_TEMPLATES	Retrieves the directory used to store template files (i.e., Documents and Settings\<user name>\Templates).
CSIDL_WINDOWS	Retrieves the Windows directory.

Table 12-8: SHGetFolderPath dwFlags values

Value	Description
SHGFP_TYPE_CURRENT	Returns the folder's current path.
SHGFP_TYPE_DEFAULT	Returns the folder's default path.

SHGetPathFromIDList ShlObj.pas

Syntax

```
SHGetPathFromIDList(
pidl: PItemIDList;            {a pointer to an item identifier list}
pszPath: PChar                {a pointer to a buffer}
): BOOL;                      {returns TRUE or FALSE}
```

Description

This function retrieves a string containing the path name of the file or folder identified by the item identifier list.

Parameters

pidl: A pointer to an item identifier list that specifies a file or directory in the file system. This function will fail if the item identifier list specifies a folder that is not in the file system, such as the Printers or Control Panel folders.

pszPath: A pointer to a buffer that receives the name of the path. The size of the buffer is assumed to be MAX_PATH bytes.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE.

See Also

SHBrowseForFolder, SHGetFileInfo

Example

Please see Listing 12-2 under SHBrowseForFolder.

SHGetSettings ShlObj.pas

Syntax

```
SHGetSettings(
var lpss: TShellFlagState;    {structure receiving shell option settings}
dwMask: DWORD                 {shell option setting flags}
);                            {this procedure does not return a value}
```

Description

This function indicates if various shell options are turned on or off.

Note: When any of these settings are changed, a WM_SETTINGCHANGE message is sent with lParam pointing to the string "ShellState."

Note: This function requires the Internet Explorer version 4.0 Desktop Update to be installed (shell32.dll version 4.71 or higher).

Parameters

lpss: A TShellFlagState structure that is initialized to indicate the state of the shell option indicated by the dwMask parameter. The TShellFlagState structure is defined as:

```
TShellFlagState = packed record
      Data: Word;                  {indicates if an option is on or off}
end;
```

Data: Contains a non-zero value if the option indicated by the dwMask parameter is enabled or selected; it contains a zero if the option is disabled or unselected.

dwMask: A flag indicating which shell option to check. This parameter can be one value from the following table.

See Also

GetSystemMetrics*

Example

Listing 12-6: Retrieving various shell option settings

```
procedure TForm1.FormCreate(Sender: TObject);
var
  ShellFlagState: TShellFlagState;    // holds the shell option setting

  {simply sets the caption according to the value}
  procedure SetValue(LabelObj: TLabel; Value: Boolean);
  begin
    if Value then
      LabelObj.Caption := 'TRUE'
    else
      LabelObj.Caption := 'FALSE';
  end;

begin
  {retrieve the setting for the active desktop - view as webpage option}
  SHGetSettings(ShellFlagState, SSF_DESKTOPHTML);
  SetValue(Label6, WordBool(ShellFlagState.Data));

  {retrieve the setting for the show map network drive button option}
  SHGetSettings(ShellFlagState, SSF_MAPNETDRVBUTTON);
  SetValue(Label7, WordBool(ShellFlagState.Data));

  {retrieve the setting for the show all files option}
  SHGetSettings(ShellFlagState, SSF_SHOWALLOBJECTS);
  SetValue(Label8, WordBool(ShellFlagState.Data));

  {retrieve the setting for the show file attributes in detail view option}
  SHGetSettings(ShellFlagState, SSF_SHOWATTRIBCOL);
  SetValue(Label9, WordBool(ShellFlagState.Data));
  {retrieve the setting for the hide file extensions for known types option}
  SHGetSettings(ShellFlagState, SSF_SHOWEXTENSIONS);
  SetValue(Label10, WordBool(ShellFlagState.Data));
end;
```

Table 12-9: SHGetSettings dwMask values

Value	Description
SSF_DESKTOPHTML	Active Desktop – View as Web Page option.
SSF_DONTPRETTYPATH	Allow All Uppercase Names option.
SSF_DOUBLECLICKINWEBVIEW	Double-Click to Open an Item option.
SSF_HIDEICONS	Not used.
SSF_MAPNETDRVBUTTON	Show Map Network Drive Button in Toolbar option.
SSF_NOCONFIRMRECYCLE	Display Delete Confirmation Dialog box option (in the recycle bin).
SSF_SHOWALLOBJECTS	Show All Files option.
SSF_SHOWATTRIBCOL	Show File Attributes in Detail View option.
SSF_SHOWCOMPCOLOR	Display Compressed Files and Folders With Alternate Color option.
SSF_SHOWEXTENSIONS	Hide File Extensions for Known File Types option.
SSF_SHOWINFOTIP	Show Info Tips for Items in Folders and Desktop option.
SSF_SHOWSYSFILES	Do Not Show Hidden Files option.
SSF_WIN95CLASSIC	Classic Style option.

SHGetSpecialFolderLocation ShlObj.pas

Syntax

```
SHGetSpecialFolderLocation(
hwndOwner: HWND;                 {a handle to a window}
nFolder: Integer;                {a folder location flag}
var ppidl: PItemIDList           {a pointer to an item identifier list}
): HResult;                      {returns an OLE result}
```

Description

This function retrieves an item identifier list specifying the location of the special folder. Note that only those folders that are registered under the key HKEY_CURRENT_USER\ Software\ Microsoft\ Windows\ CurrentVersion\ Explorer\ Shell Folders will return an item identifier list specifying a file system folder that SHGetPathFromIDList can use to retrieve a physical path name.

Note: The caller is responsible for freeing the item identifier list. This must be done via the IMalloc interface returned from SHGetMalloc.

Parameters

hwndOwner: A handle to the owning window for dialog or message boxes.

nFolder: A flag indicating the folder for which to retrieve the location. This parameter can be one value from the following table.

ppidl: A pointer to an item identifier list that specifies the indicated folder's location relative to the root of the namespace.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns an OLE-defined error result.

See Also

SHBrowseForFolder, SHGetFileInfo, SHGetFolderLocation, SHGetPathFromIDList

Example

Listing 12-7: Retrieving the location of the Windows desktop directory

```
procedure TForm1.Button1Click(Sender: TObject);
var
  IDList: PItemIDList;                  // the item identifier list
  PathName: array[0..MAX_PATH] of char; // the path of the specified folder
  ShellMalloc: IMalloc;                 // shell's memory allocator
begin
  {retrieve the item identifier list specifying the
   location of the Windows desktop directory}
  SHGetSpecialFolderLocation(Form1.Handle, CSIDL_DESKTOPDIRECTORY, IDList);

  {retrieve the path name}
  SHGetPathFromIDList(IDList, @PathName);

  {display the path name}
  Label2.Caption := PathName;

  {dispose of the item identifier list}
  SHGetMalloc(ShellMalloc);
  ShellMalloc.Free(IDList);
end;
```

Table 12-10: SHGetSpecialFolderLocation nFolder values

Value	Description
CSIDL_ADMINTOOLS	**Windows 2000 and later**: Retrieves the directory used to store administrative tools. This directory is specific to a user (i.e., <user name>\Start Menu\Programs\Administrative Tools).
CSIDL_ALTSTARTUP	Retrieves the directory of the user's non-localized Startup program group.

Value	Description
CSIDL_APPDATA	Retrieves the directory used to store application-specific data. This directory is specific to a user (i.e., Documents and Settings\<user name>\Application Data).
CSIDL_BITBUCKET	Retrieves the location of the recycle bin. This directory is not in the registry and has hidden system attributes to prevent the user from moving or deleting it.
CSIDL_COMMON_ADMINTOOLS	**Windows 2000 and later**: Retrieves the directory used to store administrative tools available to any user (i.e., All Users\Start Menu\Programs\Administrative Tools).
CSIDL_COMMON_ALTSTARTUP	**Windows NT and later**: Retrieves the directory of the non-localized Startup program group for all users.
CSIDL_COMMON_APPDATA	**Windows 2000 and later**: Retrieves the directory used to store application-specific data for all users of the computer (i.e., Documents and Settings\All Users\Application Data).
CSIDL_COMMON_DESKTOPDIRECTORY	**Windows NT and later**: Retrieves the directory containing files and folders appearing on the desktop for all users (i.e., Documents and Settings\All Users\Desktop).
CSIDL_COMMON_DOCUMENTS	Retrieves the directory used to store document files for all users of the computer (i.e., Documents and Settings\All Users\Documents).
CSIDL_COMMON_FAVORITES	**Windows NT and later**: Retrieves the directory used to store favorite links for all users of the computer.
CSIDL_COMMON_PROGRAMS	**Windows NT and later**: Retrieves the directory used to store the directories for the common program groups appearing in the Start menu for all users of the computer (i.e., Documents and Settings\All Users\Start Menu\ Programs).
CSIDL_COMMON_STARTMENU	**Windows NT and later**: Retrieves the directory used to store the programs and directories appearing in the Start menu for all users (i.e., Documents and Settings\All Users\ Start Menu).
CSIDL_COMMON_STARTUP	**Windows NT and later**: Retrieves the directory used to store the programs appearing in the Startup folder for all users (i.e., Documents and Settings\All Users\Start Menu\ Programs\Startup).
CSIDL_CONTROLS	Retrieves the virtual folder containing the icons for control panel applets.
CSIDL_COOKIES	Retrieves the directory used to store Internet cookies.
CSIDL_DESKTOP	Retrieves the virtual folder for the root of the namespace, the Windows desktop.
CSIDL_DESKTOPDIRECTORY	Retrieves the directory used to store files and folders appearing on the desktop (i.e., Documents and Settings\ <user name>\Desktop).
CSIDL_DRIVES	Retrieves the My Computer virtual folder, which contains storage devices, printers, the control panel, and may contain mapped network drives.

Value	Description
CSIDL_FAVORITES	Retrieves the directory used to store favorite links (i.e., Documents and Settings\<user name>\Favorites).
CSIDL_FLAG_CREATE	Used to force the creation of a directory if it does not exist. This is the only value that can be combined using a Boolean OR with any other value in this table.
CSIDL_FONTS	Retrieves the virtual folder containing fonts.
CSIDL_HISTORY	Retrieves the directory used to store Internet history items.
CSIDL_INTERNET_CACHE	Retrieves the directory used to store temporary Internet files.
CSIDL_LOCAL_APPDATA	**Windows 2000 and later**: Retrieves the directory used as a data repository for local (non-roaming) applications.
CSIDL_MYPICTURES	**Windows 2000 and later**: Retrieves the directory used as a common storage target for pictures. This directory is specific to a user (i.e., Documents and Settings\<user name>\ My Documents\My Pictures).
CSIDL_NETHOOD	Retrieves the directory used to store the link files appearing as objects under the My Network Places virtual folder (i.e., Documents and Settings\<user name>\NetHood).
CSIDL_NETWORK	Retrieves the network neighborhood virtual folder representing the top level of the network hierarchy.
CSIDL_PERSONAL	Retrieves the directory used to store document files. This directory is specific to a user (i.e., Documents and Settings\<user name>\My Documents).
CSIDL_PRINTERS	Retrieves the virtual folder containing installed printers.
CSIDL_PRINTHOOD	Retrieves the directory used to store the link files appearing as objects under the Printers virtual folder (i.e., Documents and Settings\<user name>\PrintHood).
CSIDL_PROGRAM_FILES	Retrieves the directory of the Program Files folder.
CSIDL_PROGRAM_FILES_COMMON	**Windows 2000 and later**: Retrieves the directory used to store executables and other components that are shared across applications (i.e., Program Files\Common).
CSIDL_PROGRAMS	Retrieves the directory containing the directories of the program groups appearing in the Start menu (i.e., Documents and Settings\<user name>\Start Menu\Programs).
CSIDL_RECENT	Retrieves the directory used to store links for the user's most recently used documents (i.e., Documents and Settings\<user name>\Recent).
CSIDL_SENDTO	Retrieves the directory containing the Send To menu items (i.e., Documents and Settings\<user name>\SendTo).
CSIDL_STARTMENU	Retrieves the directory used to store the programs and directories appearing in the Start menu (i.e., Documents and Settings\<user name>\Start Menu).

Value	Description
CSIDL_STARTUP	Retrieves the directory used to store the programs appearing in the Startup (i.e., Documents and Settings\<user name>\Start Menu\Programs\Startup).
CSIDL_SYSTEM	Retrieves the System directory.
CSIDL_TEMPLATES	Retrieves the directory used to store template files (i.e., Documents and Settings\<user name>\Templates).
CSIDL_WINDOWS	Retrieves the Windows directory.

SHGetSpecialFolderPath ***ShlObj.pas***

Syntax

```
SHGetSpecialFolderPath(
hwndOwner: HWND;          {a window handle}
lpszPath: PChar;          {null-terminated path string}
nFolder: Integer;         {folder location flags}
fCreate: BOOL             {folder creation setting}
): BOOL;                  {returns TRUE or FALSE}
```

Description

This function retrieves the physical path for the specified folder.

Note: This function requires the Internet Explorer version 4.0 Desktop Update to be installed (shell32.dll version 4.71 or higher).

Note: Only folders that physically exist in the file system are valid; the function fails if a virtual folder (i.e., Printers, Dial-up Networking) is specified. This function has been superceded by the SHGetFolderPath function on Windows 2000 and later, but it may be used on earlier systems if ShFolder.dll is present.

Parameters

hwndOwner: A handle to a window used as an owner window for any dialog boxes spawned by this function. This parameter may be zero.

lpszPath: A pointer to a null-terminated string that receives the path. This string should be of MAX_LENGTH length.

nFolder: A flag indicating the folder for which to retrieve the location. This parameter can be one value from the following table.

fCreate: A Boolean value indicating whether the folder should be created if it does not already exist. If the indicated folder does not exist and this value is set to TRUE, the folder will be created.

Return Value

This function returns TRUE if the path was retrieved, and FALSE otherwise.

See Also

SHAddToRecentDocs, SHGetFolderLocation, SHGetFolderPath, SHGetPathFromIDList

Example

Listing 12-8: Retrieving the location of the Windows desktop directory

```
procedure TForm1.Button1Click(Sender: TObject);
var
  PathName: array[0..MAX_PATH] of char; // the path name of the specified folder
begin
  {retrieve the location of the Windows desktop directory}
  SHGetSpecialFolderPath(Handle, PathName, CSIDL_DESKTOPDIRECTORY, FALSE);

  {display the path name}
  Label2.Caption := PathName;
end;
```

Table 12-11 SHGetSpecialFolderPath nFolder values

Value	Description
CSIDL_ADMINTOOLS	**Windows 2000 and later**: Retrieves the directory used to store administrative tools. This directory is specific to a user (i.e., <user name>\Start Menu\Programs\Administrative Tools).
CSIDL_ALTSTARTUP	Retrieves the directory of the user's non-localized Startup program group.
CSIDL_APPDATA	Retrieves the directory used to store application-specific data. This directory is specific to a user (i.e., Documents and Settings\<user name>\Application Data).
CSIDL_COMMON_ADMINTOOLS	**Windows 2000 and later**: Retrieves the directory used to store administrative tools available to any user (i.e., All Users\Start Menu\Programs\Administrative Tools).
CSIDL_COMMON_ALTSTARTUP	**Windows NT and later**: Retrieves the directory of the non-localized Startup program group for all users.
CSIDL_COMMON_APPDATA	**Windows 2000 and later**: Retrieves the directory used to store application-specific data for all users of the computer (i.e., Documents and Settings\All Users\Application Data).
CSIDL_COMMON_DESKTOPDIRECTORY	**Windows NT and later**: Retrieves the directory containing files and folders appearing on the desktop for all users (i.e., Documents and Settings\All Users\Desktop).
CSIDL_COMMON_DOCUMENTS	Retrieves the directory used to store document files for all users of the computer (i.e., Documents and Settings\All Users\Documents).
CSIDL_COMMON_FAVORITES	**Windows NT and later**: Retrieves the directory used to store favorite links for all users of the computer.

Value	Description
CSIDL_COMMON_PROGRAMS	**Windows NT and later**: Retrieves the directory used to store the directories for the common program groups appearing in the Start menu for all users of the computer (i.e., Documents and Settings\All Users\Start Menu\Programs).
CSIDL_COMMON_STARTMENU	**Windows NT and later**: Retrieves the directory used to store the programs and directories appearing in the Start menu for all users (i.e., Documents and Settings\All Users\Start Menu).
CSIDL_COMMON_STARTUP	**Windows NT and later**: Retrieves the directory used to store the programs appearing in the Startup folder for all users (i.e., Documents and Settings\All Users\Start Menu\Programs\Startup).
CSIDL_COOKIES	Retrieves the directory used to store Internet cookies.
CSIDL_DESKTOPDIRECTORY	Retrieves the directory used to store files and folders appearing on the desktop (i.e., Documents and Settings\<user name>\Desktop).
CSIDL_FAVORITES	Retrieves the directory used to store favorite links (i.e., Documents and Settings\<user name>\Favorites).
CSIDL_HISTORY	Retrieves the directory used to store Internet history items.
CSIDL_INTERNET_CACHE	Retrieves the directory used to store temporary Internet files.
CSIDL_LOCAL_APPDATA	**Windows 2000 and later**: Retrieves the directory used as a data repository for local (non-roaming) applications.
CSIDL_MYPICTURES	**Windows 2000 and later**: Retrieves the directory used as a common storage target for pictures. This directory is specific to a user (i.e., Documents and Settings\<user name>\My Documents\My Pictures).
CSIDL_NETHOOD	Retrieves the directory used to store the link files appearing as objects under the My Network Places virtual folder (i.e., Documents and Settings\<user name>\NetHood).
CSIDL_PERSONAL	Retrieves the directory used to store document files. This directory is specific to a user (i.e., Documents and Settings\<user name>\My Documents).
CSIDL_PRINTHOOD	Retrieves the directory used to store the link files appearing as objects under the Printers virtual folder (i.e., Documents and Settings\<user name>\PrintHood).
CSIDL_PROGRAM_FILES	Retrieves the directory of the Program Files folder.
CSIDL_PROGRAM_FILES_COMMON	**Windows 2000 and later**: Retrieves the directory used to store executables and other components that are shared across applications (i.e., Program Files\Common).
CSIDL_PROGRAMS	Retrieves the directory containing the directories of the program groups appearing in the Start menu (i.e., Documents and Settings\<user name>\Start Menu\Programs).

Value	Description
CSIDL_RECENT	Retrieves the directory used to store links for the user's most recently used documents (i.e., Documents and Settings\ <user name>\Recent).
CSIDL_SENDTO	Retrieves the directory containing the Send To menu items (i.e., Documents and Settings\<user name>\SendTo).
CSIDL_STARTMENU	Retrieves the directory used to store the programs and directories appearing in the Start menu (i.e., Documents and Settings\<user name>\Start Menu).
CSIDL_STARTUP	Retrieves the directory used to store the programs appearing in the Startup (i.e., Documents and Settings\<user name>\Start Menu\Programs\Startup).
CSIDL_SYSTEM	Retrieves the System directory.
CSIDL_TEMPLATES	Retrieves the directory used to store template files (i.e., Documents and Settings\<user name>\Templates).
CSIDL_WINDOWS	Retrieves the Windows directory.

SHQueryRecycleBin *ShellExtra.pas*

Syntax

```
SHQueryRecycleBin(
pszRootPath: PChar;                 {a null-terminated string indicating a drive}
pSHQueryRBInfo: PSHQueryRBInfo      {a pointer to a TSHQueryRBInfo structure}
): HResult;                         {returns an OLE result}
```

Description

This function retrieves the total size of and the number of items in the recycle bin on the specified drive.

Note: This function requires the Internet Explorer version 4.0 Desktop Update to be installed (shell32.dll version 4.71 or higher).

Parameters

pszRootPath: A null-terminated string containing the path of the root drive containing the recycle bin whose status is desired. At a minimum, this string requires a drive letter, but it can contain folder and subfolder names. If this parameter is set to NIL, information for all recycle bins on all drives is retrieved.

Note: Under Windows 2000 and above, this parameter must contain a valid path and cannot be set to NIL.

pSHQueryRBInfo: An address of a TSHQueryRBInfo structure that receives the recycle bin information. This structure is defined as:

```
TSHQueryRBInfo = packed record
        cbSize: DWORD;              {size of the structure, in bytes}
        i64Size: Int64;             {the total size of all recycle bin items}
        i64NumItems: Int64;         {the number of items}
end;
```

cbSize: Indicates the size of the TSHQueryRBInfo structure, in bytes. This parameter must be set to SizeOf(TSHQueryRBInfo) before the function is called.

i64Size: Indicates the total size of all objects in the specified recycle bin, in bytes.

i64NumItems: Indicates the total number of items in the specified recycle bin.

Return Value

This function returns S_OK if successful; otherwise, it returns an OLE error code.

See Also

SHEmptyRecycleBin, SHFileOperation

Example

Please see Listing 12-3 under SHEmptyRecycleBin.

Chapter 13

Shell Extension Functions

The Windows API provides application programmers with a plethora of functions that can greatly extend application functionality. Shell extensions make it possible to do the reverse: allow programmers to provide extended functionality to Windows. By using shell extensions, programmers can give the Windows shell access to new functionality that can enhance the user experience above and beyond what might be available in a standard application.

Shell extensions are typically part of an application suite that provide extended functionality outside of and beyond that offered by the applications themselves. For example, WinZip uses shell extensions that allow users to drag and drop files directly onto zipped archives. Other applications may use a shell extension that provides feedback on the state of files associated with that application. The functionality provided by shell extensions is as varied as applications and can provide the user with interesting and powerful ways to interact with an application or application files.

It is important to note that this chapter differs from others in this book in that direct API functions are not discussed. This chapter discusses various COM interfaces and how those interfaces are implemented to provide the functionality for a shell extension. The reader is assumed to have basic knowledge of how COM objects work and are implemented. An in-depth discussion of COM objects is beyond the scope of this book.

Note: The unit references for each interface method indicate the unit in which the interface is defined, not where the interface method is implemented.

Shell Extension Basics

Implementing a shell extension is a relatively straightforward process, where getting the shell to see the extension and load it at the appropriate time is concerned. What the shell extension actually provides in the way of extended functionality can be as simple or complicated as the programmer desires, and some extensions require a little more work than others. However, in general, implementing a shell extension is accomplished by following three simple steps:

1. Create a new COM object to contain the logic of the shell extension.
2. Implement specific methods for the appropriate interfaces.
3. Register the shell extension.

Creating the COM Object

To begin, create a new COM object that implements certain specific interfaces that are used by the shell for a particular shell extension. Shell extensions are implemented as in-process COM servers that Explorer loads when necessary. For the examples in this book, COM automation servers are used. To create the COM object, first select File | New from Delphi and click Other to bring up the New Items dialog box. Then, select the ActiveX tab, select ActiveX Library, and click OK. This will create the DLL in which the in-process COM server is implemented.

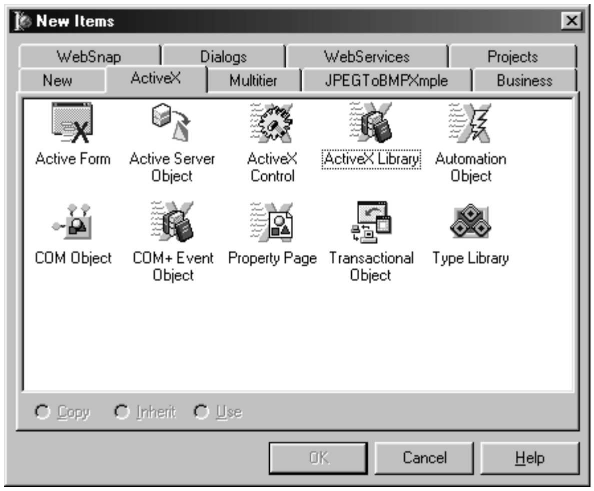
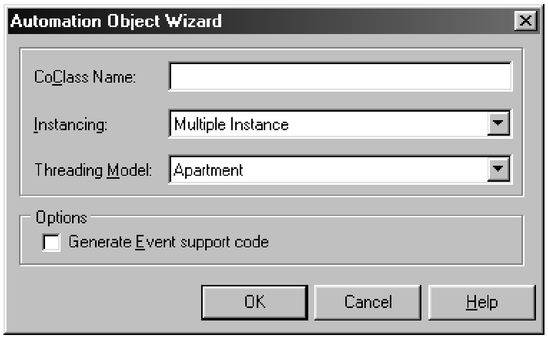


Figure 13-1: The New Items dialog box

Once the DLL is created, the COM object itself must be defined. Select File | New again, click Other, and go back to the ActiveX tab. Select Automation Object, and click OK. This will open the Automation Object Wizard dialog box. Fill in the CoClass Name box with the name of the COM object, and click OK. This will create the basic COM object structure that is ready for implementation.



CoClass Name:
Instancing: Multiple Instance
Threading Model: Apartment
☐ Generate Event support code

Figure 13-2: The Automation Object Wizard dialog box

Note: These steps were based on Delphi 6 at the time of publication. Earlier or later versions of Delphi may require a different method, and will likely feature dialog boxes that look different from those depicted in the screenshots.

Implementing Interface Methods

Now that the COM object has been created, the next step is to declare the shell extension interfaces that are implemented, and implement their methods. To declare the interfaces, put them in the class definition of the object generated by the Automation Object Wizard. Each interface's methods must be declared and implemented by this object, which will vary widely depending on the shell extension created.

Registering the Shell Extension

Once the COM object is fully implemented, it must be registered with the system. Registering a COM object is simply a matter of clicking Run | Register ActiveX Server within Delphi. However, all shell extensions require additional entries in the registry to inform Explorer that they exist and where they can be located. Different shell extensions require different types of registration, and some require even additional steps when registered under Windows NT/2000.

All COM objects are created through a class factory object, as seen in the Initialization section of the COM object's unit. A typical method for providing the additional registration required by shell extensions is to create a descendant of this class factory and override the UpdateRegistry method. For some shell extensions, the ApproveShell-Extension method of the class factory is also overridden to provide the additional registry manipulation required under Windows NT/2000.

Some shell extensions are not immediately loaded and used by Explorer after installation. Often, the user must logout and log back in or reboot the system before the shell extension is used. Typically, installation programs that are installing a shell extension should ask the user to reboot to insure that Explorer sees the new extension.

URL Search Hook Shell Extensions

Web browsers use the URL search hook shell extension to provide translation of a URL address for a protocol that it does not recognize. For example, this would be useful for applications that wish to browse to Internet addresses using a proprietary URL address format or for providing shortcuts to frequently visited addresses.

When an address is entered into the browser, the browser attempts to determine the protocol used. If it does not recognize the protocol (i.e., "http://" is not present at the beginning of the address), it begins loading URL search hook extensions. It loads each registered extension, passing the entered address and receiving a translated address, until the address is fully translated into a protocol the browser understands.

Implementing URL Search Hook Shell Extensions

COM objects must implement the IURLSearchHook interface to become a URL search hook shell extension. This interface contains only one method: Translate. This method receives the URL address provided to the browser, translates the address, and returns an Internet address to which the browser can navigate.

Registering URL Search Hook Shell Extensions

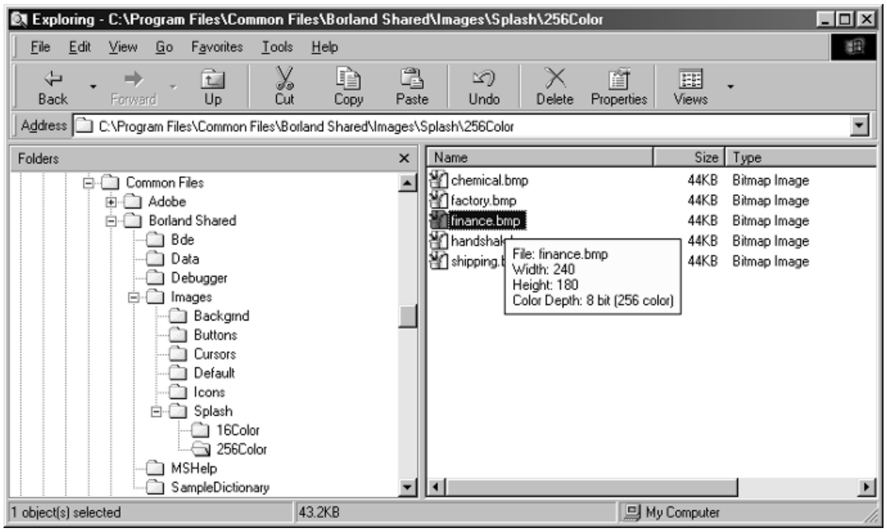
URL search hook shell extensions must be registered under the HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\URLSearchHooks key. Simply write the class identifier of the COM object as the value name, with no associated value.

Listing 13-1: Implementing IURLSearchHook

```
unit URLSearchHookXampleU;

{$WARN SYMBOL_PLATFORM OFF}

interface

uses
  ComObj, ActiveX, URLSearchHookXmpl_TLB, StdVcl, ShlObj, Windows;

type
  TURLSearchHookXmpl = class(TAutoObject, IURLSearchHookXmpl, IURLSearchHook)
  protected
    {IURLSearchHook Methods}
    function Translate(lpwszSearchURL: PWideChar; cchBufferSize: DWORD):
             HResult; stdcall;
  end;

  {the new class factory}
  TURLSearchHookXmplFactory = class(TAutoObjectFactory)
  public
    procedure UpdateRegistry(Register: Boolean); override;
  end;

implementation

uses
  ComServ, Registry, SysUtils;

{ TURLSearchHookXmpl }

{provides the translation of unknown URL protocols}
function TURLSearchHookXmpl.Translate(lpwszSearchURL: PWideChar;
  cchBufferSize: DWORD): HResult;
var
  TransAddr: string;      // holds the translated address
begin
  {initialize values}
  Result := E_FAIL;
  TransAddr := '';
  {provide translation for some 2-character shortcuts to frequently
   visited web sites}
  if UpperCase(lpwszSearchURL) = 'MS' then
    TransAddr := 'http://msdn.microsoft.com'
  else
  if UpperCase(lpwszSearchURL) = 'BL' then
    TransAddr := 'http://www.borland.com';

  {indicate if the translation succeeded}
  if TransAddr  '' then
  begin
    StringToWideChar(TransAddr, lpwszSearchUrl, (Length(TransAddr) + 1) * 2);
    Result := S_OK;
  end;
end;

{ TURLSearchHookXmplFactory }

{provides additional registry manipulation for this shell extension}
procedure TURLSearchHookXmplFactory.UpdateRegistry(Register: Boolean);
var
  TempReg: TRegistry;
begin
  {perform normal registration}
  inherited UpdateRegistry(Register);

  {if this server is being registered, write a value into the
   appropriate key to expose it to Explorer}
  if Register then
    CreateRegKey('Software\Microsoft\Internet Explorer\URLSearchHooks',
                 GUIDToString(ClassID), '', HKEY_CURRENT_USER)
  else
  begin
    {upon unregistering, delete the value for this server but leave other
     server values intact}
    TempReg := TRegistry.Create;
    try
      TempReg.RootKey := HKEY_CURRENT_USER;
      TempReg.OpenKey('Software\Microsoft\Internet Explorer\URLSearchHooks',
                      TRUE);
      TempReg.DeleteValue(GUIDToString(ClassID))
    finally
      TempReg.Free;
    end;
  end;
end;

initialization
  TURLSearchHookXmplFactory.Create(ComServer, TURLSearchHookXmpl,
                                   Class_URLSearchHookXmpl,
                                   ciMultiInstance, tmApartment);
end.
```

Infotip Shell Extensions

Under Windows 98 with Internet Explorer 5 (or Windows 2000 and later), Explorer uses Infotip shell extensions to provide additional information to the user when the mouse is hovered over a file. This information is displayed like a pop-up tooltip (or in the status bar under Windows 2000.



Name	Size	Type
chemical.bmp	44KB	Bitmap Image
factory.bmp	44KB	Bitmap Image
finance.bmp	44KB	Bitmap Image
handshak	44KB	Bitmap Image
shipping.b	44KB	Bitmap Image

File: finance.bmp
Width: 240
Height: 180
Color Depth: 8 bit (256 color)

Figure 13-3: The bitmap infotip in action

When the mouse cursor hovers over a file in Explorer, Explorer checks the registry for the extension of the file to determine if an infotip extension is registered. If so, it loads the extension and calls the GetInfoTip method, which in turn sends back a text string that Explorer displays.

Implementing Infotip Shell Extensions

COM objects must implement the IQueryInfo, IPersistFile, and IPersist interfaces to become an infotip shell extension (IPersist must be implemented because it is the ancestor of IPersistFile). Only the Load method of IPersistFile is of actual interest; this method is passed the name of the file whose information tip is to be displayed. All other methods of IPersistFile and IPersist can return E_NOTIMPL.

The GetInfoTip method of IQueryInfo is called after IPersistFile.Load. GetInfoTip assembles and returns the string that is displayed in Explorer as the tip for the file. The GetInfoTip method receives a pointer to a wide char that receives this string, but the memory for this string must be set by the shell extension using the shell's memory allocater. Use the SHGetMalloc API function to retrieve an IMalloc interface for the shell's memory allocater, and use the Alloc method to allocate memory for the string. This must be done to allow the shell to dispose of the string when it is no longer needed.

Note: IQueryInfo declares a method called GetInfoFlags. This method is not currently used, but it must be implemented and must return the value E_NOTIMPL.

Registering Infotip Shell Extensions

Infotip extensions are registered under the HKEY_CLASSES_ROOT\<file type>\ShellEx key. A key whose name must match the interface identifier for the IQueryInfo interface must be written under this key, and its default value must be set to the class identifier of the COM object. For example, the listing below implements an infotip extension for bitmap files. The registry entry for this COM server is:

```
HKEY_CLASSES_ROOT
  .bmp
    ShellEx
      {00021500-0000-0000-C000-000000000046}   (IQueryInterface interface id)
        Default = {DD952428-1E30-11D5-8978-0050DA8DB54F} (extension class id)
```

Additionally, Windows NT/2000 requires an entry for the COM object's class identifier into the HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved key.

Listing 13-2: A bitmap infotip shell extension

```
unit BMPInfoTipU;

interface

uses
  ComObj, ActiveX, BMPInfoTip_TLB, StdVcl, ShlObj, Windows;

type
  TBMPInfoTip = class(TAutoObject, IBMPInfoTip, IQueryInfo, IPersistFile,
                      IPersist)
  protected
    pMalloc: IMalloc;     // pointer to the shell's memory allocater
    FBMPFile: string;     // the bitmap file

    {IQueryInfo methods}
    function GetInfoTip(dwFlags: DWORD; var ppwszTip: PWideChar): HResult; stdcall;
    function GetInfoFlags(out pdwFlags: DWORD): HResult; stdcall;

    {IPersistFile methods}
    function IsDirty: HResult; stdcall;
    function Load(pszFileName: POleStr; dwMode: Longint): HResult; stdcall;
    function Save(pszFileName: POleStr; fRemember: BOOL): HResult; stdcall;
    function SaveCompleted(pszFileName: POleStr): HResult; stdcall;
    function GetCurFile(out pszFileName: POleStr): HResult; stdcall;

    {IPersist methods}
    function GetClassID(out classID: TCLSID): HResult; stdcall;
  public
    {TBMPInfoTip methods}
    procedure Initialize; override;
    destructor Destroy; override;
  end;

  {the new class factory}
  TBMPInfoTipFactory = class(TAutoObjectFactory)
  protected
    procedure ApproveShellExtension(Register: Boolean; const ClsID: string);
    function GetProgID: string; override;
  public
    procedure UpdateRegistry(Register: Boolean); override;
  end;


implementation

uses ComServ, Graphics, SysUtils, Registry;

{ TBMPInfoTip }

{ - IQueryInfo methods - }

function TBMPInfoTip.GetInfoTip(dwFlags: DWORD;
  var ppwszTip: PWideChar): HResult;
var
  FileInfo: string;
  BitmapFile: TBitmap;
begin
  {initialize values}
  Result := E_FAIL;
  FileInfo := 'File: ' + ExtractFileName(FBMPFile) + #13#10;
  BitmapFile := TBitmap.Create;

  try
    {load the bitmap}
    BitmapFile.LoadFromFile(FBMPFile);

    {retrieve the width and height}
    FileInfo := FileInfo + 'Width: ' + IntToStr(BitmapFile.Width) + #13#10;
    FileInfo := FileInfo + 'Height: ' + IntToStr(BitmapFile.Height) + #13#10;

    {retrieve the color depth}
    FileInfo := FileInfo + 'Color Depth: ';
    case BitmapFile.PixelFormat of
      pfDevice  : FileInfo := FileInfo + 'Device';
      pf1bit    : FileInfo := FileInfo + '1 bit (monochrome)';
      pf4bit    : FileInfo := FileInfo + '4 bit (16 color)';
      pf8bit    : FileInfo := FileInfo + '8 bit (256 color)';
      pf15bit   : FileInfo := FileInfo + '15 bit';
      pf16bit   : FileInfo := FileInfo + '16 bit (65,536 color)';
      pf24bit   : FileInfo := FileInfo + '24 bit (true color)';
      pf32bit   : FileInfo := FileInfo + '32 bit (true color + alpha)';
      pfCustom  : FileInfo := FileInfo + 'Custom';
    end;
  {allocate memory for the infotip string}
  ppwszTip := pMalloc.Alloc(SizeOf(WideChar) * (Length(FileInfo) + 1));

  {copy the infotip into the string}
  if (ppwszTip  nil) then
    ppwszTip := StringToWideChar(FileInfo, ppwszTip, SizeOf(WideChar) *
                                 Length(FileInfo) + 1);
  finally
    BitmapFile.Free;
  end;

  Result := S_OK;
end;

function TBMPInfoTip.GetInfoFlags(out pdwFlags: DWORD): HResult;
begin
  {this method is declared, but is not used}
  Result := E_NOTIMPL;
end;


{ - IPersistFile methods - }

function TBMPInfoTip.Load(pszFileName: POleStr; dwMode: Integer): HResult;
begin
  {this method passes the file name to the shell extension}
  FBMPFile := pszFileName;
  Result := S_OK;
end;

function TBMPInfoTip.IsDirty: HResult;
begin
  Result := E_NOTIMPL;
end;

function TBMPInfoTip.Save(pszFileName: POleStr; fRemember: BOOL): HResult;
begin
  Result := E_NOTIMPL;
end;

function TBMPInfoTip.SaveCompleted(pszFileName: POleStr): HResult;
begin
  Result := E_NOTIMPL;
end;

function TBMPInfoTip.GetCurFile(out pszFileName: POleStr): HResult;
begin
  Result := E_NOTIMPL;
end;

{ - IPersist methods - }

function TBMPInfoTip.GetClassID(out classID: TCLSID): HResult;
begin
  Result := E_NOTIMPL;
end;
```

```
{ - TBMPInfoTip methods - }

procedure TBMPInfoTip.Initialize;
begin
  inherited;

  {retrieve a pointer to the shell's memory allocater}
  if Failed(ShGetMalloc(pMalloc)) then
    pMalloc := nil;
end;

destructor TBMPInfoTip.Destroy;
begin
  inherited;
  pMalloc := nil;
end;

{ TBMPInfoTipFactory }

{provides additional registry manipulation for Windows NT/2000}
procedure TBMPInfoTipFactory.ApproveShellExtension(Register: Boolean;
const ClsID: string);
var
  TempReg: TRegistry;
begin
  TempReg := TRegistry.Create;

  try
    TempReg.RootKey := HKEY_LOCAL_MACHINE;

    {open the appropriate key}
    if not TempReg.OpenKey('SOFTWARE\Microsoft\Windows\CurrentVersion\Shell
                           Extensions\Approved', True) then
      Exit;

    {register the extension appropriately}
    if Register then
      TempReg.WriteString(ClsID, Description)
    else
      TempReg.DeleteValue(ClsID);
  finally
    TempReg.Free;
  end;
end;

function TBMPInfoTipFactory.GetProgID: string;
begin
  {the ProgID is not needed for shell extensions}
  Result := '';
end;

procedure TBMPInfoTipFactory.UpdateRegistry(Register: Boolean);
begin
  {perform normal registration}
  inherited UpdateRegistry(Register);
  {perform registration for Windows NT/2000}
  ApproveShellExtension(Register, GUIDToString(ClassID));

  {write the appropriate value for registration}
  if Register then
    CreateRegKey('.bmp\shellex\' + SID_IQueryInfo, '', GUIDToString(ClassID))
  else
    DeleteRegKey('.bmp\shellex\' + SID_IQueryInfo);
end;

initialization
  TBMPInfoTipFactory.Create(ComServer, TBMPInfoTip, CLASS_BMPInfoTip,
                            ciMultiInstance, tmApartment);

end.
```

Copy Hook Shell Extensions

Explorer uses copy hook shell extensions to determine if a shell folder or printer object can be moved, copied, renamed, or deleted. Technically, without a copy hook shell extension, these operations always succeed, but a copy hook shell extension can provide additional restraints and either permit or deny the processing of the operation. Unfortunately, copy hook shell extensions work only on shell folders or printer objects; they do not work on file objects.

When the user attempts to move, copy, rename, or delete a shell folder or printer, Explorer begins loading registered copy hook shell extensions. It queries each one until all copy hook shell extensions indicate that processing can continue or until one of them indicates that the process should not be allowed. Copy hook shell extensions merely indicate if the operation can continue or should be halted; they do not perform the operation themselves nor are they notified if the operation actually succeeded.

Implementing Copy Hook Shell Extensions

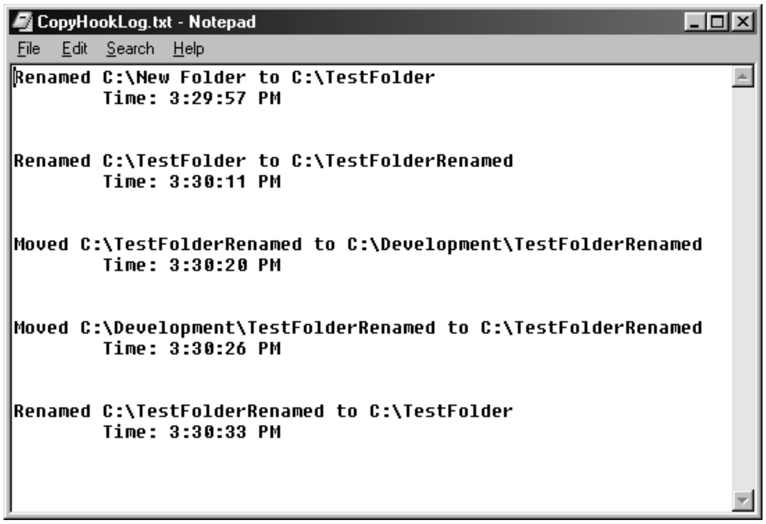
COM objects must implement the ICopyHook interface to become a copy hook shell extension. The only method in this interface, CopyCallback, is called when the user tries to copy, rename, move, or delete a file folder or printer. CopyCallback receives information about the source and destination folder (or printer) names and the nature of the operation. Based on this information and the functionality provided by the handler, CopyCallback returns a value to indicate if the operation should continue or is not allowed.

Registering Copy Hook Shell Extensions

Copy hook shell extensions are registered under the HKEY_CLASSES_ROOT\Directory\ShellEx\CopyHookHandlers key. A key whose name is set to the class name of the COM object must be written under this key, and its default value must be set to the class identifier of the COM object. For printers, the registry key is the same, except that instead of HKEY_CLASSES_ROOT\Directory, it must be located under HKEY_CLASSES_ROOT\Printers. Additionally, Windows NT/2000 requires an entry for the COM object's class identifier into the HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved key.

Listing 13-3: Logging directory modifications

```
unit CopyHookXampleU;

{$WARN SYMBOL_PLATFORM OFF}

interface

uses
  Windows, ComObj, ActiveX, CopyHookXample_TLB, StdVcl, ShlObj;

type
  TCopyHookXmpl = class(TAutoObject, ICopyHookXmpl, ICopyHook)
  protected
    {ICopyHook Methods}
    function CopyCallback(Wnd: HWND; wFunc, wFlags: UINT; pszSrcFile: PAnsiChar;
      dwSrcAttribs: DWORD; pszDestFile: PAnsiChar; dwDestAttribs: DWORD): UINT; stdcall;
  end;

  {the new class factory}
  TCopyHookXmplFactory = class(TAutoObjectFactory)
  protected
    procedure ApproveShellExtension(Register: Boolean; const ClsID: string);
    function GetProgID: string; override;
  public
    procedure UpdateRegistry(Register: Boolean); override;
  end;

implementation

uses
  ComServ, Registry, SysUtils, ShellAPI, Classes;

{ TCopyHookXmpl }

function TCopyHookXmpl.CopyCallback(Wnd: HWND; wFunc, wFlags: UINT;
  pszSrcFile: PAnsiChar; dwSrcAttribs: DWORD; pszDestFile: PAnsiChar;
  dwDestAttribs: DWORD): UINT;
var
  LogFile: TextFile;
begin
  {open the log file}
  AssignFile(LogFile, 'c:\CopyHookLog.txt');
  if FileExists('c:\CopyHookLog.txt') then
    Append(LogFile)
  else
    Rewrite(LogFile);

  {record the operation into the log file}
  case wFunc of
    FO_MOVE    : WriteLn(LogFile, 'Moved ' + pszSrcFile + ' to ' + pszDestFile +
                         #13#10#9 + 'Time: ' + TimeToStr(Now) + #13#10#13#10);
    FO_COPY    : WriteLn(LogFile, 'Copied ' + pszSrcFile + ' to ' + pszDestFile +
                         #13#10#9 + 'Time: ' + TimeToStr(Now) + #13#10#13#10);
    FO_DELETE  : WriteLn(LogFile, 'Deleted ' + pszSrcFile +
                         #13#10#9 + 'Time: ' + TimeToStr(Now) + #13#10#13#10);
    FO_RENAME  : WriteLn(LogFile, 'Renamed ' + pszSrcFile + ' to ' + pszDestFile +
                         #13#10#9 + 'Time: ' + TimeToStr(Now) + #13#10#13#10);
  end;

  {write and close the logfile}
  CloseFile(LogFile);

  Result := IDYES;
end;

{ TCopyHookXmplFactory }

{provides additional registry manipulation for Windows NT/2000}
procedure TCopyHookXmplFactory.ApproveShellExtension(Register: Boolean;
  const ClsID: string);
var
  TempReg: TRegistry;
begin
  TempReg := TRegistry.Create;

  try
    TempReg.RootKey := HKEY_LOCAL_MACHINE;

    {open the appropriate key}
    if not TempReg.OpenKey('SOFTWARE\Microsoft\Windows\CurrentVersion\Shell
                           Extensions\Approved', True) then
      Exit;

    {register the extension appropriately}
    if Register then
      TempReg.WriteString(ClsID, Description)
    else
      TempReg.DeleteValue(ClsID);
  finally
    TempReg.Free;
  end;
end;

function TCopyHookXmplFactory.GetProgID: string;
begin
  {the ProgID is not needed for shell extensions}
  Result := '';
end;

{provides additional registry manipulation for this shell extension}
procedure TCopyHookXmplFactory.UpdateRegistry(Register: Boolean);
var
  TempReg: TRegistry;
begin
  {perform normal registration}
  inherited UpdateRegistry(Register);

  {perform registration for Windows NT/2000}
  ApproveShellExtension(Register, GUIDToString(ClassID));

  {register or unregister the shell extension}
  if Register then
    CreateRegKey('directory\shellex\CopyHookHandlers\' + ClassName, '',
                 GUIDToString(ClassID))
  else
    DeleteRegKey('directory\shellex\CopyHookHandlers\' + ClassName);
end;


initialization
  TCopyHookXmplFactory.Create(ComServer, TCopyHookXmpl, CLASS_CopyHookXmpl,
                              ciMultiInstance, tmApartment);

end.
```



```
Renamed C:\New Folder to C:\TestFolder
        Time: 3:29:57 PM


Renamed C:\TestFolder to C:\TestFolderRenamed
        Time: 3:30:11 PM


Moved C:\TestFolderRenamed to C:\Development\TestFolderRenamed
        Time: 3:30:20 PM


Moved C:\Development\TestFolderRenamed to C:\TestFolderRenamed
        Time: 3:30:26 PM


Renamed C:\TestFolderRenamed to C:\TestFolder
        Time: 3:30:33 PM
```

Figure 13-4: The copy hook shell extension log

Shell Execute Hook Shell Extensions

Shell execute hook extensions are used when the ShellExecute or ShellExecuteEx API functions are called. This occurs every time a file is double-clicked in Explorer or when the Run dialog box is used. The shell execute hook extension can completely process the request (preventing the call to ShellExecute or ShellExecuteEx), or it can perform any desired processing and allow ShellExecute or ShellExecuteEx to continue processing the request.

When ShellExecute or ShellExecuteEx is called, Explorer begins loading registered shell execute hook extensions. It sends the request to each one, allowing the extension to perform its processing. This continues until all extensions have processed the request and allowed it to pass on to ShellExecute or ShellExecuteEx, or one of the extensions processes the request completely.

Implementing Shell Execute Hook Shell Extensions

COM objects must implement the IShellExecuteHook interface to become a shell execute hook extension. This interface has one method, Execute, that is called every time ShellExecute or ShellExecuteEx is called. This method is called first, allowing it to suppress the final call to ShellExecute or ShellExecuteEx if necessary. This method receives a pointer to a TShellExecuteInfo structure containing information about the file being opened or executable being launched. If the extension handles the request and launches the requested application, the hInstApp member of this structure should be set to the instance handle of the application. If the extension did not launch the requested application but wants to suppress the call to ShellExecute or ShellExecuteEx, the hInstApp member should be set to a value greater than 32 to prevent the shell from displaying an error message.

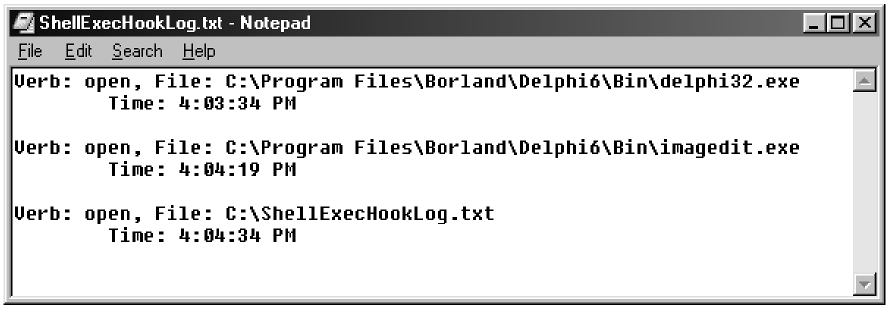
Registering Shell Execute Hook Shell Extensions

Shell execute hook extensions must be registered under the HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Explorer\ShellExecuteHooks key. Simply write the class identifier of the COM object as the value name, with no associated value. Additionally, Windows NT/2000 requires an entry for the COM object's class identifier into the HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved key.

Listing 13-4: Logging file execute operations

```
unit ShellExecHookXampleU;

{$WARN SYMBOL_PLATFORM OFF}

interface

uses
  ComObj, ActiveX, ShellExecHookXample_TLB, StdVcl, ShlObj, ShellAPI;

type
  TShellExecHookXmpl = class(TAutoObject, IShellExecHookXmpl, IShellExecuteHook)
  protected
    {IShellExecuteHook Methods}
    function Execute(var ShellExecuteInfo: TShellExecuteInfo): HResult;stdcall;
  end;

  {the new class factory}
  TShellHookExecXmplFactory = class(TAutoObjectFactory)
  protected
    procedure ApproveShellExtension(Register: Boolean; const ClsID: string);
    function GetProgID: string; override;
  public
    procedure UpdateRegistry(Register: Boolean); override;
  end;
```

```
implementation

uses
  ComServ, Registry, SysUtils, Windows;

{ TShellExecHookXmpl }

function TShellExecHookXmpl.Execute(var ShellExecuteInfo: TShellExecuteInfo): HResult;
var
  LogFile: TextFile;
begin
  {open the log file}
  AssignFile(LogFile, 'c:\ShellExecHookLog.txt');
  if FileExists('c:\ShellExecHookLog.txt') then
    Append(LogFile)
  else
    Rewrite(LogFile);

  {write the operation to the log file}
  WriteLn(LogFile, 'Verb: ' + ShellExecuteInfo.lpVerb + ', File: ' +
          ShellExecuteInfo.lpFile + #13#10#9 + 'Time: ' + TimeToStr(Now) +
          #13#10);

  {close the log file}
  CloseFile(LogFile);

  {indicate that the request was processed, but that default processing
   should continue}
  Result := S_FALSE;
end;

{ TShellHookExecXmpl }

{provides additional registry manipulation for Windows NT/2000}
procedure TShellHookExecXmplFactory.ApproveShellExtension(Register: Boolean;
  const ClsID: string);
var
  TempReg: TRegistry;
begin
  TempReg := TRegistry.Create;

  try
    TempReg.RootKey := HKEY_LOCAL_MACHINE;

    {open the appropriate key}
    if not TempReg.OpenKey('SOFTWARE\Microsoft\Windows\CurrentVersion\Shell
                           Extensions\Approved', True) then
      Exit;

    {register the extension appropriately}
    if Register then
      TempReg.WriteString(ClsID, Description)
    else
      TempReg.DeleteValue(ClsID);
  finally
    TempReg.Free;
  end;
end;

function TShellHookExecXmplFactory.GetProgID: string;
begin
  {the ProgID is not needed for shell extensions}
  Result := '';
end;

{provides additional registry manipulation for this shell extension}
procedure TShellHookExecXmplFactory.UpdateRegistry(Register: Boolean);
var
  TempReg: TRegistry;
begin
  {perform normal registration}
  inherited UpdateRegistry(Register);

  {perform registration for Windows NT/2000}
  ApproveShellExtension(Register, GUIDToString(ClassID));

  {if this server is being registered, write a value into the
   appropriate key to expose it to Explorer}
  if Register then
    CreateRegKey('Software\Microsoft\Windows\CurrentVersion\Explorer\
                 ShellExecuteHooks', GUIDToString(ClassID), '',
                 HKEY_LOCAL_MACHINE)
  else
  begin
    {upon unregistering, delete the value for this server but leave other
     server values intact}
    TempReg := TRegistry.Create;
    try
      TempReg.RootKey := HKEY_LOCAL_MACHINE;
      TempReg.OpenKey('Software\Microsoft\Windows\CurrentVersion\Explorer\
                      ShellExecuteHooks', TRUE);
      TempReg.DeleteValue(GUIDToString(ClassID))
    finally
      TempReg.Free;
    end;
  end;
end;


initialization
  TShellHookExecXmplFactory.Create(ComServer, TShellExecHookXmpl,
                                   Class_ShellExecHookXmpl,
                                   ciMultiInstance, tmApartment);

end.
```



```
Verb: open, File: C:\Program Files\Borland\Delphi6\Bin\delphi32.exe
        Time: 4:03:34 PM

Verb: open, File: C:\Program Files\Borland\Delphi6\Bin\imagedit.exe
        Time: 4:04:19 PM

Verb: open, File: C:\ShellExecHookLog.txt
        Time: 4:04:34 PM
```

Figure 13-5: The file execute log

Context Menu Handler Shell Extensions

When the user right-clicks an item on the desktop or from Explorer, the system displays a pop-up menu for the item. Context menu shell extensions add items to this menu and allow the user quick access to specific functionality related to the selected item. Any number of menu items can be added for the item from a single context menu shell extension.

When the item is right-clicked, Explorer determines what default items are displayed in the context menu and then begins loading registered context menu shell extensions. As it loads each one, it queries the extension to retrieve the menu items to add to the context menu. A single context menu extension can add one or more menu items as necessary. This continues until all context menu shell extensions registered for the item have had a chance to add their menu items. When the user selects a menu item added by a context menu extension, this context menu extension is called to perform the specific functionality represented by the selected menu item.

Implementing Context Menu Handler Shell Extensions

COM objects must implement the IShellExtInit and IContextMenu interfaces to become a context menu shell extension. IShellExtInit defines only one method, Initialize, but unfortunately, TComObject also defines this method, so a method resolution clause must be used. The IShellExtInit.Initialize method is called first when the context menu extension is loaded. Initialize receives a data object and other parameters that contain information about the selected file, and any initialization of the extension should occur at this time.

The IContextMenu interface defines three methods: QueryContextMenu, GetCommandString, and InvokeCommand. QueryContextMenu is called first and is passed the handle of the menu into which the menu items are inserted, how the menu was spawned, the index of the first menu item, and a range of valid menu item identifiers. This method should use these values to insert appropriate menu items at the indicated spot using the InsertMenu or InsertMenuItem API functions. GetCommandString is called when the inserted menu item receives focus, and it simply returns a help string for the menu item that is displayed in the Explorer status bar. Finally, when the user selects the menu item, the InvokeCommand method is called, which carries out the functionality on the selected file represented by the chosen menu item.

Registering Context Menu Handler Shell Extensions

Context menu shell extensions are registered under the HKEY_CLASSES_ROOT\<file type>\ShellEx\ContextMenuHandlers key. To find the appropriate file type, look under HKEY_CLASSES_ROOT for the extension of the file type associated with the context menu. Its default value will be the file type under which the context menu is registered. A key whose name is set to the class name of the COM object must be written under this key, and its default value must be set to the class identifier of the COM object. Additionally, Windows NT/2000 requires an entry for the COM object's class identifier into the HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved key.

Listing 13-5: Context menu for converting JPEG images to bitmap

```
unit JPEGToBMPXmplU;

{$WARN SYMBOL_PLATFORM OFF}

interface

uses
  ComObj, ActiveX, JPEGToBMPXmple_TLB, StdVcl, ShlObj, Windows;

type
  TIJPEGToBMPContextMenu = class(TAutoObject, IIJPEGToBMPContextMenu,
                               IShellExtInit, IContextMenu)
  private
    FFileName: string;
    FMenuItemIndex: UINT;
  protected
    { IShellExtInit Methods }
    function IShellExtInit.Initialize = ShellExtInitialize;
    function ShellExtInitialize(pidlFolder: PItemIDList; lpdobj: IDataObject;
      hKeyProgID: HKEY): HResult; stdcall;

    { IContextMenu Methods }
    function QueryContextMenu(Menu: HMENU;
      indexMenu, idCmdFirst, idCmdLast, uFlags: UINT): HResult; stdcall;
    function InvokeCommand(var lpici: TCMInvokeCommandInfo): HResult; stdcall;
    function GetCommandString(idCmd, uType: UINT; pwReserved: PUINT;
      pszName: LPSTR; cchMax: UINT): HResult; stdcall;
  end;

  {the new class factory}
  TJPEGToBMPObjectFactory = class(TAutoObjectFactory)
  protected
    procedure ApproveShellExtension(Register: Boolean; const ClsID: string);
    function GetProgID: string; override;
  public
    procedure UpdateRegistry(Register: Boolean); override;
  end;
implementation

uses
```

```
  ComServ, JPeg, Graphics, ShellAPI, SysUtils, Registry;

{provides jpeg to bitmap conversion logic}
procedure ConvertToBmp(JpegImg: string);
var
  TempJpg: TJpegImage;
  TempBMP: TBitmap;
begin
  {load the jpeg}
  TempJpg := TJpegImage.Create;
  TempJpg.LoadFromFile(JpegImg);

  {create the bitmap}
  TempBMP := TBitmap.Create;

  {set bitmap dimensions to match the jpeg}
  TempBMP.Width := TempJpg.Width;
  TempBMP.Height := TempJpg.Height;

  {draw the jpeg to the bitmap}
  TempBMP.Canvas.Draw(0, 0, TempJpg);

  {save the bitmap}
  TempBMP.SaveToFile(ChangeFileExt(JpegImg, '.bmp'));

  {cleanup}
  TempBMP.Free;
  TempJpg.Free;
end;


{ TIJPEGToBMPContextMenu }

{ - IShellExtInit Methods - }

function TIJPEGToBMPContextMenu.ShellExtInitialize(pidlFolder: PItemIDList;
  lpdobj: IDataObject; hKeyProgID: HKEY): HResult;
var
  DataFormat: TFormatEtc;
  StrgMedium: TStgMedium;
begin
  Result := E_FAIL;

  {if no object was provided, exit}
  if lpdobj = nil then
    Exit;

  {prepare to retrieve information about the object}
  DataFormat.cfFormat := CF_HDROP;
  DataFormat.ptd := nil;
  DataFormat.dwAspect :=DVASPECT_CONTENT;
  DataFormat.lindex := -1;
  DataFormat.tymed := TYMED_HGLOBAL;
  if lpdobj.GetData(DataFormat, StrgMedium)  S_OK then
    Exit;
  {valid only if one file is selected}
  if DragQueryFile(StrgMedium.hGlobal, $FFFFFFFF, nil, 0) = 1 then
  begin
    {retrieve the file name}
    SetLength(FFileName, MAX_PATH);
    DragQueryFile(StrgMedium.hGlobal, 0, PChar(FFileName), MAX_PATH);
  end;

  {only one file selected, ready to go}
  ReleaseStgMedium(StrgMedium);
  Result := NOERROR;
end;

{ - IContextMenu Methods - }

{called to add items to the menu}
function TIJPEGToBMPContextMenu.QueryContextMenu(Menu: HMENU; indexMenu,
  idCmdFirst, idCmdLast, uFlags: UINT): HResult;
begin
  {only adding one menu item, so generate the result code accordingly}
  Result := MakeResult(SEVERITY_SUCCESS, 0, 1);

  {store the menu item index}
  FMenuItemIndex := indexMenu;

  {specify what the menu says, depending on where it was spawned}
  if (uFlags = CMF_NORMAL) then                       // from the desktop
    InsertMenu(Menu, indexMenu, MF_STRING or MF_BYPOSITION, idCmdFirst,
               'Convert JPEG to Bitmap (from desktop)')
  else
  if (uFlags and CMF_VERBSONLY) = CMF_VERBSONLY then  // from a shortcut
    InsertMenu(Menu, indexMenu, MF_STRING or MF_BYPOSITION, idCmdFirst,
               'Convert JPEG to Bitmap (from shortcut)')
  else
  if (uFlags and CMF_EXPLORE) = CMF_EXPLORE then      // from explorer
    InsertMenu(Menu, indexMenu, MF_STRING or MF_BYPOSITION, idCmdFirst,
               'Convert JPEG to Bitmap (from Explorer)')
  else
    {fail for any other value}
    Result := E_FAIL;
end;

{called to get the help string}
function TIJPEGToBMPContextMenu.GetCommandString(idCmd, uType: UINT;
  pwReserved: PUINT; pszName: LPSTR; cchMax: UINT): HResult;
begin
  Result := E_INVALIDARG;

  {indicates the help string displayed in the bottom of the explorer when
   the menu item is selected}
  if (idCmd = FMenuItemIndex) and (uType = GCS_HELPTEXT) then
  begin
    StrLCopy(pszName, 'Convert the selected JPEG image to a bitmap', cchMax);
    Result := NOERROR;
  end;
end;
```

```
{called to perform the functionality represented by the menu item}
function TIJPEGToBMPContextMenu.InvokeCommand(var lpici:
                                              TCMInvokeCommandInfo): HResult;
begin
  Result := E_FAIL;

  {if a verb string is specified, don't process}
  if HiWord(integer(lpici.lpVerb))  0 then
    Exit;

  {if the index matches the index for the menu, convert the jpeg to a bitmap}
  if LoWord(integer(lpici.lpVerb)) = FMenuItemIndex then
  begin
    ConvertToBmp(FFileName);
    Result := NOERROR;
  end;
end;

{ TJPEGToBMPObjectFactory }

{provides additional registry manipulation for Windows NT/2000}
procedure TJPEGToBMPObjectFactory.ApproveShellExtension(Register: Boolean;
  const ClsID: string);
var
  TempReg: TRegistry;
begin
  TempReg := TRegistry.Create;

  try
    TempReg.RootKey := HKEY_LOCAL_MACHINE;

    {open the appropriate key}
    if not TempReg.OpenKey('SOFTWARE\Microsoft\Windows\CurrentVersion\Shell
                           Extensions\Approved', True) then
      Exit;

    {register the extension appropriately}
    if Register then
      TempReg.WriteString(ClsID, Description)
    else
      TempReg.DeleteValue(ClsID);
  finally
    TempReg.Free;
  end;
end;

function TJPEGToBMPObjectFactory.GetProgID: string;
begin
  {the ProgID is not needed for shell extensions}
  Result := '';
end;

procedure TJPEGToBMPObjectFactory.UpdateRegistry(Register: Boolean);
begin
  {perform normal registration}
  inherited UpdateRegistry(Register);
  {perform registration for Windows NT/2000}
  ApproveShellExtension(Register, GUIDToString(ClassID));

  {if this server is being registered, write a value into the
   appropriate key to expose it to Explorer}
  if Register then
    CreateRegKey('jpegfile\shellex\ContextMenuHandlers\' + ClassName, '',
                 GUIDToString(ClassID), HKEY_CLASSES_ROOT)
  else
    DeleteRegKey('jpegfile\shellex\ContextMenuHandlers\' + ClassName);
end;

initialization
  TJPEGToBMPObjectFactory.Create(ComServer, TIJPEGToBMPContextMenu,
                                 Class_IJPEGToBMPContextMenu,
                                 ciMultiInstance, tmApartment);
end.
```

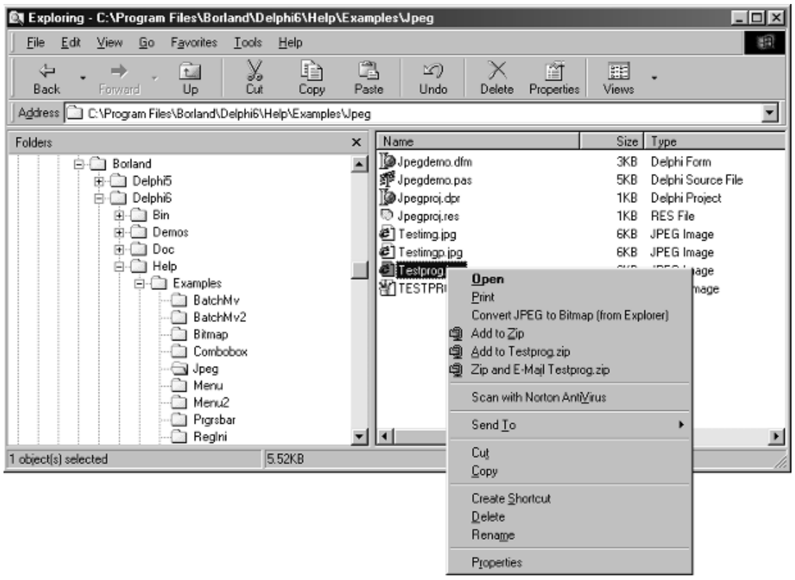

Exploring - C:\Program Files\Borland\Delphi6\Help\Examples\Jpeg
File Edit View Go Favorites Tools Help
Back Forward Up Cut Copy Paste Undo Delete Properties Views
Address C:\Program Files\Borland\Delphi6\Help\Examples\Jpeg
Folders
Borland
Delphi5
Delphi6
Bin
Demos
Doc
Help
Examples
BatchMv
BatchMv2
Bitmap
Combobox
Jpeg
Menu
Menu2
Prgrsbar
RegIni
Name Size Type
Jpegdemo.dfm 3KB Delphi Form
Jpegdemo.pas 5KB Delphi Source File
Jpegproj.dpr 1KB Delphi Project
Jpegproj.res 1KB RES File
Testimg.jpg 6KB JPEG Image
Testimgp.jpg 6KB JPEG Image
Testprog
TESTPR
Open
Print
Convert JPEG to Bitmap (from Explorer)
Add to Zip
Add to Testprog.zip
Zip and E-Mail Testprog.zip
Scan with Norton AntiVirus
Send To
Cut
Copy
Create Shortcut
Delete
Rename
Properties
1 object(s) selected
5.52KB

Figure 13-6: The context menu extension from Explorer

Icon Handler Shell Extensions

Explorer uses icon handler shell extensions to display a non-default icon for a file or folder object. Typically, this type of shell extension is used to provide icons for a file on a per-instance basis. For example, such extensions could be used to display a different icon for a specific file type based on its read-only attribute; a read-only file would display a different icon than a normal file, even though the files are the same type.

When Explorer needs to draw the icon for a specific file type, it checks the registry for the file type associated with the file extension to determine if an icon handler extension is registered. If so, it loads the extension and retrieves either a handle to the appropriate icon or a location of a file containing the icon and the icon's index within that file.

Implementing Icon Handler Shell Extensions

COM objects must implement the IExtractIcon, IPersistFile, and IPersist interfaces to become an icon handler shell extension (IPersist must be implemented because it is the ancestor of IPersistFile). Only the Load method of IPersistFile is of actual interest; this method is passed the name of the file whose icon is to be extracted. All other methods of IPersistFile and IPersist can return E_NOTIMPL.

IPersistFile.Load is called first and receives the name of the file whose icon is to be determined. Explorer then calls the GetIconLocation method. This method should return the name of a file containing the icon resource to use and the index of the icon within the file. It also returns a combination of flags that instruct Explorer on how this information is used. Typically, it indicates that the icon should be used on a per-instance basis and that it shouldn't be cached by the shell. However, in some implementations, the extension may need to pass back an actual handle to a specific icon. In this case, the GIL_NOTFILENAME flag is passed back, indicating that the specified filename and index do not identify a valid icon location. When this happens, Explorer calls the Extract method, which returns a valid handle to a large and small icon.

Registering Icon Handler Shell Extensions

Icon handler shell extensions are registered under the HKEY_CLASSES_ROOT\<file type>\ShellEx\IconHandler key. To find the appropriate file type, look under HKEY_CLASSES_ROOT for the extension of the file type associated with the icon handler. Its default value will be the file type under which the icon handler is registered. The default value of the HKEY_CLASSES_ROOT\<file type>\ShellEx\IconHandler key must be set to the class identifier of the COM object. For icon handlers that provide per-instance icons, the default value of HKEY_CLASSES_ROOT\<file type>\DefaultIcon must be set to "%1." Additionally, Windows NT/2000 requires an entry for the COM object's class identifier into the HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved key.

Listing 13-6: Displaying different icons for read-only .pas files

```
unit ExtractIconXampleU;

{$WARN SYMBOL_PLATFORM OFF}

interface

uses
  ComObj, ActiveX, ExtractIconXample_TLB, StdVcl, ShlObj, Windows;

type
  TIExtractIconXmpl = class(TAutoObject, IIExtractIconXmpl, IPersistFile,
                          IExtractIcon)
  private
    FFileName: string;
  protected
    {IExtractIcon methods}
    function GetIconLocation(uFlags: UINT; szIconFile: PAnsiChar; cchMax: UINT;
      out piIndex: Integer; out pwFlags: UINT): HResult; stdcall;
    function Extract(pszFile: PAnsiChar; nIconIndex: UINT;
      out phiconLarge, phiconSmall: HICON; nIconSize: UINT): HResult; stdcall;

    {IPersistFile methods}
    function IsDirty: HResult; stdcall;
    function Load(pszFileName: POleStr; dwMode: Longint): HResult; stdcall;
    function Save(pszFileName: POleStr; fRemember: BOOL): HResult; stdcall;
    function SaveCompleted(pszFileName: POleStr): HResult; stdcall;
    function GetCurFile(out pszFileName: POleStr): HResult; stdcall;

    {IPersist methods}
    function GetClassID(out classID: TCLSID): HResult; stdcall;
  end;

  {the new class factory}
  TExtractIconXmplFactory = class(TAutoObjectFactory)
  protected
    procedure ApproveShellExtension(Register: Boolean; const ClsID: string);
    function GetProgID: string; override;
  public
    procedure UpdateRegistry(Register: Boolean); override;
  end;

implementation

uses
  ComServ, SysUtils, Registry;

{ - IPersistFile methods - }

function TIExtractIconXmpl.Load(pszFileName: POleStr; dwMode: Integer): HResult;
begin
  {save the filename}
  FFileName := pszFileName;
  Result := S_OK;
end;

function TIExtractIconXmpl.IsDirty: HResult;
begin
  Result := E_NOTIMPL;
end;

function TIExtractIconXmpl.Save(pszFileName: POleStr; fRemember: BOOL): HResult;
begin
  Result := E_NOTIMPL;
end;

function TIExtractIconXmpl.SaveCompleted(pszFileName: POleStr): HResult;
begin
  Result := E_NOTIMPL;
end;

function TIExtractIconXmpl.GetCurFile(out pszFileName: POleStr): HResult;
begin
  Result := E_NOTIMPL;
```

```
end;

{ - IPersist methods - }

function TIExtractIconXmpl.GetClassID(out classID: TCLSID): HResult;
begin
  Result := E_NOTIMPL;
end;

{ - IExtractIcon methods - }

{called to extract icon handles}
function TIExtractIconXmpl.Extract(pszFile: PAnsiChar; nIconIndex: UINT;
  out phiconLarge, phiconSmall: HICON; nIconSize: UINT): HResult;
begin
  Result := NOERROR;
  phiconSmall := 0;       // no icon for the small icon

  {load the appropriate icon}
  case nIconIndex of
    0 : phiconLarge := LoadIcon(hInstance, 'PASNORM');
    1 : phiconLarge := LoadIcon(hInstance, 'PASREADONLY');
  end;
end;

{called to get icon locations}
function TIExtractIconXmpl.GetIconLocation(uFlags: UINT;
  szIconFile: PAnsiChar; cchMax: UINT; out piIndex: Integer;
  out pwFlags: UINT): HResult;
begin
  Result := NOERROR;

  {indicate the name of this COM server dll, but this is not used}
  StrLCopy(szIconFile, PChar(ComServer.ServerFileName), cchMax);

  {indicate the icon to use based on the readonly attribute}
  piIndex := 0;
  if FileIsReadOnly(FFileName) then
    piIndex := 1;

  {indicate that this is a per-instance icon that shouldn't be cached,
   and that the system should call Extract to get icon handles}
  pwFlags := pwFlags or GIL_PERINSTANCE or GIL_DONTCACHE or GIL_NOTFILENAME;
end;

{ TExtractIconXmplFactory }

{provides additional registry manipulation for Windows NT/2000}
procedure TExtractIconXmplFactory.ApproveShellExtension(Register: Boolean;
  const ClsID: string);
var
  TempReg: TRegistry;
begin
  TempReg := TRegistry.Create;

  try
```

```
    TempReg.RootKey := HKEY_LOCAL_MACHINE;

    {open the appropriate key}
    if not TempReg.OpenKey('SOFTWARE\Microsoft\Windows\CurrentVersion\Shell
                           Extensions\Approved', True) then
      Exit;

    {register the extension appropriately}
    if Register then
      TempReg.WriteString(ClsID, Description)
    else
      TempReg.DeleteValue(ClsID);
  finally
    TempReg.Free;
  end;
end;

function TExtractIconXmplFactory.GetProgID: string;
begin
  {the ProgID is not needed for shell extensions}
  Result := '';
end;

procedure TExtractIconXmplFactory.UpdateRegistry(Register: Boolean);
var
  TempReg: TRegistry;
begin
  {perform normal registration}
  inherited UpdateRegistry(Register);

  {perform registration for Windows NT/2000}
  ApproveShellExtension(Register, GUIDToString(ClassID));

  {if this server is being registered, write a value into the
   appropriate key to expose it to Explorer}
  TempReg := TRegistry.Create;
  TempReg.RootKey := HKEY_CLASSES_ROOT;

  if Register then
  begin
    CreateRegKey('DelphiUnit\ShellEx\IconHandler', '', GUIDToString(ClassID),
                 HKEY_CLASSES_ROOT);

    {make a backup of the original default icon location}
    if TempReg.OpenKey('DelphiUnit\DefaultIcon', FALSE) then
      TempReg.WriteString('DefaultIconBackup', TempReg.ReadString(''));

    CreateRegKey('DelphiUnit\DefaultIcon', '', '%1', HKEY_CLASSES_ROOT);
  end
  else
  begin
    DeleteRegKey('DelphiUnit\ShellEx\IconHandler');

    {restore the original default icon location, if it was backed up}
    if TempReg.OpenKey('DelphiUnit\DefaultIcon', FALSE) then
      if TempReg.ValueExists('DefaultIconBackup') then
```

```
        TempReg.WriteString('', TempReg.ReadString('DefaultIconBackup'));
  end;

  TempReg.Free;
end;

initialization
  TExtractIconXmplFactory.Create(ComServer, TIExtractIconXmpl, Class_IExtractIconXmpl,
    ciMultiInstance, tmApartment);
end.
```

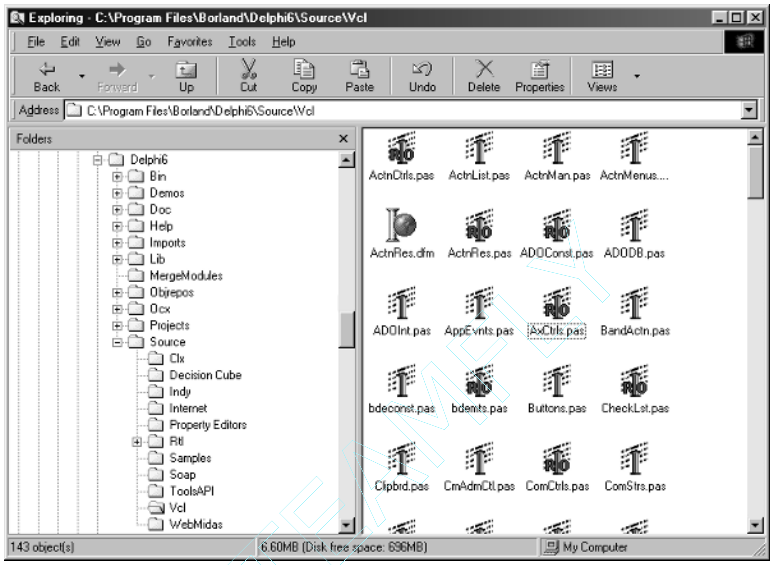


Figure 13-7: Different icons for read-only files

Drag-Drop Handler Shell Extensions

A drag-drop handler shell extension allows files to be dragged and dropped on registered file types. A classic example is WinZip. WinZip registers a drag-drop handler shell extension that, when files are dragged over a .zip file, allows those files to be added to the .zip file they are dropped upon.

When a file or folder object is dragged over another file object, Explorer checks the registry for the file type associated with the file extension to determine if a drag-drop handler extension is registered. If so, it loads the drag-drop handler extension and calls various methods to determine if a file will be accepted by the target and provides feedback as to what type of action the user is attempting.

Implementing Drag-Drop Handler Shell Extensions

COM objects must implement the IDropTarget, IPersistFile, and IPersist interfaces to become a drag-drop handler shell extension (IPersist must be implemented because it is the ancestor of IPersistFile). Only the Load method of IPersistFile is of actual interest;

this method is passed the name of the file over which another file is being dragged. All other methods of IPersistFile and IPersist can return E_NOTIMPL.

IPersistFile.Load is called first and receives the name of the file identified as the drop target. Explorer then calls the DragEnter method. This method should determine if the dragged file is valid for this drop target and return a flag that changes the mouse cursor to indicate what type of operation is being attempted, such as a copy or move operation. IDropTarget.DragOver is called every time the mouse is moved over the drop target while still dragging a file, and IDropTarget.DragLeave is called when the mouse leaves the drop target area. Finally, when the user releases the mouse button over the drop target, IDropTarget.Drop is called to perform the functionality provided by the shell extension based on the attempted drag-drop operation.

Registering Drag-Drop Handler Shell Extensions

Drag-drop handler shell extensions are registered under the HKEY_CLASSES_ROOT\<file type>\ShellEx\DropHandler key. To find the appropriate file type, look under HKEY_CLASSES_ROOT for the extension of the file type associated with the drag-drop handler. Its default value will be the file type under which the drag-drop handler is registered. The default value of the HKEY_CLASSES_ROOT\<file type>\ShellEx\DropHandler key must be set to the class identifier of the COM object. Additionally, Windows NT/2000 requires an entry for the COM object's class identifier into the HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved key.

Listing 13-7: Adding images to an HTML file via drag-drop

```
unit DropTargetXampleU;

{$WARN SYMBOL_PLATFORM OFF}

interface

uses
  ComObj, ActiveX, DropTargetXample_TLB, StdVcl, Windows;

type
  TIDropTargetXmple = class(TAutoObject, IIDropTargetXmple, IDropTarget,
                            IPersistFile, IPersist)
  private
    FFileName: string;
  protected
    {IDropTarget Methods}
    function DragEnter(const dataObj: IDataObject; grfKeyState: Longint;
      pt: TPoint; var dwEffect: Longint): HResult; stdcall;
    function DragOver(grfKeyState: Longint; pt: TPoint;
      var dwEffect: Longint): HResult; stdcall;
    function DragLeave: HResult; stdcall;
    function Drop(const dataObj: IDataObject; grfKeyState: Longint; pt: TPoint;
      var dwEffect: Longint): HResult; stdcall;

    {IPersistFile methods}
    function IsDirty: HResult; stdcall;
    function Load(pszFileName: POleStr; dwMode: Longint): HResult; stdcall;
    function Save(pszFileName: POleStr; fRemember: BOOL): HResult; stdcall;
    function SaveCompleted(pszFileName: POleStr): HResult; stdcall;
    function GetCurFile(out pszFileName: POleStr): HResult; stdcall;

    {IPersist methods}
    function GetClassID(out classID: TCLSID): HResult; stdcall;
  end;

  {the new class factory}
  TDropTargetXmplFactory = class(TAutoObjectFactory)
  protected
    procedure ApproveShellExtension(Register: Boolean; const ClsID: string);
    function GetProgID: string; override;
  public
    procedure UpdateRegistry(Register: Boolean); override;
  end;

implementation

uses
  ComServ, SysUtils, ShellAPI, Registry, Dialogs, Classes;

{ - IPersistFile methods - }

function TIDropTargetXmple.Load(pszFileName: POleStr; dwMode: Integer): HResult;
begin
  {store the filename of the drop target}
  FFileName := pszFileName;
  Result := S_OK;
end;

function TIDropTargetXmple.IsDirty: HResult;
begin
  Result := E_NOTIMPL;
end;

function TIDropTargetXmple.Save(pszFileName: POleStr; fRemember: BOOL): HResult;
begin
  Result := E_NOTIMPL;
end;

function TIDropTargetXmple.SaveCompleted(pszFileName: POleStr): HResult;
begin
  Result := E_NOTIMPL;
end;

function TIDropTargetXmple.GetCurFile(out pszFileName: POleStr): HResult;
begin
  Result := E_NOTIMPL;
end;


{ - IPersist methods - }
```

```
function TIDropTargetXmple.GetClassID(out classID: TCLSID): HResult;
begin
  Result := E_NOTIMPL;
end;


{ TIDropTargetXmple }

{called when the cursor enters the drop target area dragging a file}
function TIDropTargetXmple.DragEnter(const dataObj: IDataObject;
  grfKeyState: Integer; pt: TPoint; var dwEffect: Integer): HResult;
var
  DataFormat: TFormatEtc;       // data object format information
  StrgMedium: TStgMedium;       // data object storage medium
  FileName: string;             // filename of dragged object
begin
  {initialize}
  Result := E_ABORT;
  dwEffect := DROPEFFECT_NONE;

  {if no object was provided, exit}
  if dataObj = nil then
    Exit;

  {prepare to retrieve information about the object}
  DataFormat.cfFormat := CF_HDROP;
  DataFormat.ptd := nil;
  DataFormat.dwAspect :=DVASPECT_CONTENT;
  DataFormat.lindex := -1;
  DataFormat.tymed := TYMED_HGLOBAL;

  if dataObj.GetData(DataFormat, StrgMedium)  S_OK then
    Exit;

  {valid only if one file is selected}
  if DragQueryFile(StrgMedium.hGlobal, $FFFFFFFF, nil, 0) = 1 then
  begin
    {retrieve the filename of the dragged object}
    SetLength(FileName, MAX_PATH);
    DragQueryFile(StrgMedium.hGlobal, 0, PChar(FileName), MAX_PATH);
    DragFinish(StrgMedium.hGlobal);
  end
  else
    Exit;

  {if this file is a jpeg or bitmap file, allow the operation}
  if (Trim(ExtractFileExt(PChar(FileName))) = '.jpg') or
     (Trim(ExtractFileExt(PChar(FileName))) = '.jpeg') or
     (Trim(ExtractFileExt(PChar(FileName))) = '.bmp') then
    dwEffect := DROPEFFECT_COPY;

  {clean up}
  ReleaseStgMedium(StrgMedium);

  Result := S_OK;
end;
```

```
{called when the mouse cursor leaves the drop target area}
function TIDropTargetXmple.DragLeave: HResult;
begin
  Result := S_OK;
end;

{called every time the mouse cursor moves over the drop target area}
function TIDropTargetXmple.DragOver(grfKeyState: Integer; pt: TPoint;
  var dwEffect: Integer): HResult;
begin
  {indicate the copy effect}
  dwEffect := DROPEFFECT_COPY;
  Result := S_OK;
end;

{called when the file is dropped on the drop target}
function TIDropTargetXmple.Drop(const dataObj: IDataObject;
  grfKeyState: Integer; pt: TPoint; var dwEffect: Integer): HResult;
var
  DataFormat: TFormatEtc;       // data object format information
  StrgMedium: TStgMedium;       // data object storage medium
  FileName: string;             // filename of dragged object
  Msg, FileStr: string;
  HtmlFile: TStringList;
begin
  {initialize}
  Result := E_ABORT;

  {if no object was provided, exit}
  if dataObj = nil then
    Exit;

  {prepare to retrieve information about the object}
  DataFormat.cfFormat := CF_HDROP;
  DataFormat.ptd := nil;
  DataFormat.dwAspect :=DVASPECT_CONTENT;
  DataFormat.lindex := -1;
  DataFormat.tymed := TYMED_HGLOBAL;

  if dataObj.GetData(DataFormat, StrgMedium)  S_OK then
    Exit;

  {valid only if one file is selected}
  if DragQueryFile(StrgMedium.hGlobal, $FFFFFFFF, nil, 0) = 1 then
  begin
    {get the name of the dropped file}
    SetLength(FileName, MAX_PATH);
    DragQueryFile(StrgMedium.hGlobal, 0, PChar(FileName), MAX_PATH);
    DragFinish(StrgMedium.hGlobal);
  end
  else
    Exit;

  {retrieve the caption for the image, defaulting to the image filename}
  Msg := Trim(FileName);
  if not InputQuery('Image Caption', 'Please enter a caption for the image',
                    Msg) then
    Exit;

  {make sure the html template file exists}
  if not FileExists(ExtractFilePath(ComServer.ServerFileName) +
                    'ImgsCollectionTemplate.html') then
    Exit;

  HtmlFile := TStringList.Create;
  try
    {load the html template file and retrieve its text}
    HtmlFile.LoadFromFile(ExtractFilePath(ComServer.ServerFileName) +
                          'ImgsCollectionTemplate.html');
    FileStr := HtmlFile.Text;

    {insert an img tag into the html file right before the </body> tag}
    Insert('<img align="middle" src="' + Trim(PChar(FileName)) + '"><p>' + Msg+
           '<p>'#13#10, FileStr, Pos('</body>', FileStr) - 1);

    {set the contents of the html file and save}
    HtmlFile.Text := FileStr;
    HtmlFile.SaveToFile(ExtractFilePath(ComServer.ServerFileName) +
                        'ImgsCollectionTemplate.html');
  finally
    HtmlFile.Free;
  end;

  Result := S_OK;
end;

{ TDropTargetXmplFactory }

{provides additional registry manipulation for Windows NT/2000}
procedure TDropTargetXmplFactory.ApproveShellExtension(Register: Boolean;
  const ClsID: string);
var
  TempReg: TRegistry;
begin
  TempReg := TRegistry.Create;

  try
    TempReg.RootKey := HKEY_LOCAL_MACHINE;

    {open the appropriate key}
    if not TempReg.OpenKey('SOFTWARE\Microsoft\Windows\CurrentVersion\Shell
                          Extensions\Approved', True) then
      Exit;

    {register the extension appropriately}
    if Register then
      TempReg.WriteString(ClsID, Description)
    else
      TempReg.DeleteValue(ClsID);
  finally
    TempReg.Free;
  end;
```

```
end;

function TDropTargetXmplFactory.GetProgID: string;
begin
  {the ProgID is not needed for shell extensions}
  Result := '';
end;

procedure TDropTargetXmplFactory.UpdateRegistry(Register: Boolean);
begin
  {perform normal registration}
  inherited UpdateRegistry(Register);

  {perform registration for Windows NT/2000}
  ApproveShellExtension(Register, GUIDToString(ClassID));

  {write the appropriate value for registration}
  if Register then
    CreateRegKey('htmlfile\ShellEx\DropHandler', '', GUIDToString(ClassID),
                 HKEY_CLASSES_ROOT)
  else
    DeleteRegKey('htmlfile\ShellEx\DropHandler');
end;

initialization
  TDropTargetXmplFactory.Create(ComServer, TIDropTargetXmple,
                                Class_IDropTargetXmple,
                                ciMultiInstance, tmApartment);
end.
```

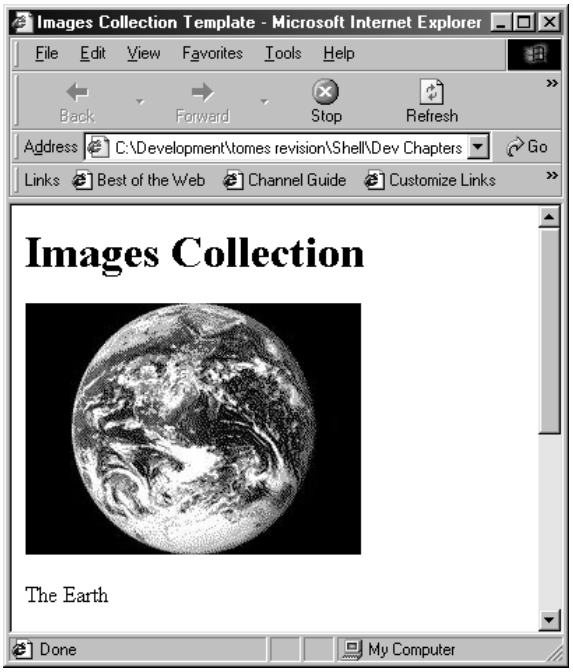

Images Collection Template - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh
Address C:\Development\tomes revision\Shell\Dev Chapters Go
Links Best of the Web Channel Guide Customize Links
Images Collection
The Earth
Done
My Computer

Figure 13-8: The HTML page after accepting a dropped file

Delphi vs. the Windows API

While several Delphi units define the interfaces and constants for the shell extensions discussed in this chapter, there are no VCL components or other objects that implement these interfaces. Delphi does ship with good examples of a copy hook extension and a context menu extension under the Demos\ActiveX directory, and there are plenty of third-party applications that wrap the complexities of shell extension implementation to speed development. However, the implementation of shell extensions is relatively straightforward, and the trickiest part of getting shell extensions to work is getting them registered correctly.

Shell Extension Functions

The following shell extension functions are covered in this chapter.

Table 13-1: Shell extension functions

Function	Description
IContextMenu.GetCommandString	Retrieves the help string or verb for the command.
IContextMenu.InvokeCommand	Processes the menu item.
IContextMenu.QueryContextMenu	Retrieves menu items to add to the context menu.
ICopyHook.CopyCallBack	Approves or rejects the moving, renaming, copying, or deleting of a folder or printer.
IDropTarget.DragEnter	Indicates if the drag operation can be dropped and what action is taken.
IDropTarget.DragLeave	Called when the mouse cursor leaves the drop target.
IDropTarget.DragOver	Provides feedback when the mouse cursor is over a drop target.
IDropTarget.Drop	Performs the drop operation.
IExtractIcon.Extract	Retrieves icon handles.
IExtractIcon.GetIconLocation	Retrieves the location and index of an icon.
IQueryInfo.GetInfoFlags	Not used, but must return E_NOTIMPL.
IQueryInfo.GetInfoTip	Retrieves an information tip.
IShellExecuteHook.Execute	Provides additional processing when ShellExecute or ShellExecuteEx is called.
IShellExtInit.Initialize	Used by Explorer to initialize certain shell extensions.
IURLSearchHook.Translate	Performs URL address translation.

IContextMenu.GetCommandString ShlObj.ps

Syntax

```
GetCommandString(
idCmd: UINT;                {menu command identifier index}
uType: UINT;                {information flags}
pwReserved: PUINT;          {reserved, should be ignored}
pszName: LPSTR;             {the desired string}
cchMax: UINT                {the size of the pszName buffer}
): HResult;                 {returns an OLE result}
```

Description

This method is called to retrieve information about the command specified by the idCmd parameter, such as the help string or the command verb. Based on the uType flags, handlers should return the appropriate information in the pszName parameter.

Parameters

idCmd: Specifies the index of the command item within the menu.

uType: A flag indicating the type of information the shell is trying to retrieve. This can be one value from the following table.

pwReserved: This parameter is reserved and should be ignored by handlers.

pszName: A null-terminated string that should be set to the desired information.

cchMax: The size of the null-terminated string buffer pointed to by the pszName parameter.

Return Value

This method should return NOERROR if successful or an OLE error result otherwise.

See Also

IContextMenu.InvokeCommand, IContextMenu.QueryContextMenu

Example

Please see Listing 13-5 in the introduction.

Table 13-2: IContextMenu.GetCommandString uType values

Value	Description
GCS_HELPTEXT	The pszName parameter should be set to the help text for the command.
GCS_VALIDATE	The shell is not requesting string information, only validation that the indicated command is available. Return NOERROR if the indicated command exists or S_FALSE if it does not.
GCS_VERB	Indicates that pszName is set to a common verb, such as “open,” “print,” etc.

IContextMenu.InvokeCommand ShlObj.pas

Syntax

```
InvokeCommand(
var lpici: TCMInvokeCommandInfo     {a TCMInvokeCommandInfo structure}
): HResult;                         {returns an OLE result}
```

Description

Called when the user selects one of the commands added to the context menu, this method instructs the handler to perform the functionality provided by the menu command.

Parameters

lpici: A TCMInvokeCommandInfo structure containing information about the menu command selected by the user. The TCMInvokeCommandInfo structure is defined as:

```
TCMInvokeCommandInfo = record
      cbSize: DWORD;                 {size of the structure}
      fMask: DWORD;                  {behavior flags}
      hwnd: HWND;                    {owning window handle}
      lpVerb: LPCSTR;                {verb or menu index}
      lpParameters: LPCSTR;          {NIL}
      lpDirectory: LPCSTR;           {NIL}
      nShow: Integer;                {window show flags}
      dwHotKey: DWORD;               {application hot key}
      hIcon: THandle;                {application icon}
end;
```

cbSize: Indicates the size of the TCMInvokeCommand structure, in bytes.

fMask: A combination of flags that indicate additional behavior. This may be zero or a combination of values from Table 13-3.

hwnd: Specifies a handle to the window owning the context menu.

lpVerb: If the user selected the menu item from the context menu, the high-order word of this value is set to zero, and the low-order word contains the index of the selected menu item. Otherwise, this value points to a null-terminated string indicating the command verb (i.e., "open"), typically indicating that the command was activated by another application.

lpParameters: This member is always NIL for context menu shell handlers.

lpDirectory: This member is always NIL for context menu shell handlers.

nShow: A flag indicating how the handler should display a window if it starts an application. This can be one value from Table 13-4.

dwHotKey: Indicates an optional hot key that should be associated with any application launched by the command.

hIcon: Indicates an optional icon that should be used for any application launched by the command.

Return Value

This method should return NOERROR if successful or an OLE error result otherwise.

See Also

IContextMenu.GetCommandString, IContextMenu.QueryContextMenu

Example

Please see Listing 13-5 in the introduction.

Table 13-3: IContextMenu.InvokeCommand TCMInvokeCommand.fMask values

Value	Description
CMIC_MASK_HOTKEY	Indicates that the dwHotKey member contains valid data.
CMIC_MASK_ICON	Indicates that the hIcon member contains valid data.
CMIC_MASK_FLAG_NO_UI	Prevents the display of user interface elements while processing the command.
CMIC_MASK_NO_CONSOLE	Prevents the creation of new consoles.
CMIC_MASK_ASYNCOK	Indicates that the handler should wait for the DDE conversation to terminate before returning.

Table 13-4: IContextMenu.InvokeCommand TCMInvokeCommand nShow values

Value	Description
SW_HIDE	The window is hidden and another window is activated.
SW_MINIMIZE	The window is minimized and the next top-level window in the system window list is activated.
SW_RESTORE	The window is activated and displayed in its original size and position.
SW_SHOW	The window is activated and displayed in its current size and position.
SW_SHOWDEFAULT	The window is shown based on the wShowWindow member of the TStartupInfo structure passed to the CreateProcess function by the program that started the application. This is used to set the initial show state of an application's main window. This flag should be used when showing the window for the first time if the application could be run from a shortcut. This flag will cause the window to be shown using the Run settings under the shortcut properties.
SW_SHOWMAXIMIZED	The window is activated and displayed in a maximized state.
SW_SHOWMINIMIZED	The window is activated and displayed as an icon.
SW_SHOWMINNOACTIVE	The window is displayed as an icon. The active window remains active.
SW_SHOWNA	The window is displayed in its current state. The active window remains active.
SW_SHOWNOACTIVE	The window is displayed in its most recent state. The active window remains active.
SW_SHOWNORMAL	This is the same as SW_RESTORE.

IContextMenu.QueryContextMenu ShlObj.pas

Syntax

QueryContextMenu(	
Menu: HMENU;	{the menu handle}
indexMenu: UINT;	{first menu item index}
idCmdFirst: UINT;	{minimum menu item identifier}
idCmdLast: UINT;	{maximum menu item identifier}
uFlags: UINT	{behavior flags}
): HResult;	{returns an OLE result}

Description

This method retrieves the menu items to add to the context menu. Menu items should be added to the menu using the InsertMenu or InsertMenuItem API functions.

Parameters

Menu: A handle to the menu into which menu items are inserted.

indexMenu: Indicates the position within the menu at which the first menu item should be inserted. This is a zero-based position.

idCmdFirst: Indicates the minimum value to use for menu item identifiers.

idCmdLast: Indicates the maximum value to use for menu item identifiers.

uFlags: Flags indicating how the menu was called. This can be one value from the following table.

Note: There are other values defined for this parameter, but the listed values are those that are valid for context menu handlers.

Return Value

If the method succeeds, it should return an OLE result code created by using the following line of code:

```
MakeResult(SEVERITY_SUCCESS, 0, <largest menu item identifier>-idCmdFirst+1);
```

Otherwise, it returns an OLE error code.

See Also

IContextMenu.GetCommandString, IContextMenu.InvokeCommand

Example

Please see Listing 13-5 in the introduction.

Table 13-5: IContextMenu.QueryContextMenu uFlags values

Value	Description
CMF_NORMAL	Indicates normal operation (typically called from the desktop).
CMF_VERBSONLY	Indicates that the menu was called from a shortcut.
CMF_EXPLORE	Indicates that the menu was called from Explorer.

ICopyHook.CopyCallback *ShlObj.pas*

Syntax

```
CopyCallback(
Wnd: HWND;                      {a window handle}
wFunc: UINT,                    {operation flags}
wFlags: UINT;                   {control flags}
pszSrcFile: PAnsiChar;          {source folder name}
dwSrcAttribs: DWORD;            {source folder attributes}
pszDestFile: PAnsiChar;         {destination folder name}
dwDestAttribs: DWORD            {destination folder attributes}
): UINT;                        {returns on OLE result}
```

Description

This method determines if a folder or printer can be moved, copied, renamed, or deleted.

Parameters

Wnd: A handle to a window that should be used as the parent window for any user interface displayed (i.e., error message boxes, etc.).

wFunc: A flag indicating the operation to perform. This parameter may be one value from Table 13-6.

wFlags: A combination of flags that indicate specific behavior for the operation. This parameter may be one or more values from Table 13-7.

pszSrcFile: A null-terminated string containing the name of the source folder or printer.

dwSrcAttribs: A combination of flags indicating the attributes of the source folder. This may be one or more values from Table 13-8.

pszDestFile: A null-terminated string containing the name of the destination folder or printer.

dwDestAttribs: A combination of flags indicating the attributes of the destination folder. This may be one or more values from Table 13-8.

Return Value

This method returns one value from Table 13-9.

See Also

CreateDirectory, CreateDirectoryEx, RemoveDirectory

Example

Please see Listing 13-3 in the introduction.

Table 13-6: ICopyHook.CopyCallback wFunc values

Value	Description
FO_COPY	Copies the files specified by the pszSrcFile parameter to the location specified by the pszDestFile parameter.
FO_DELETE	Deletes the files specified by the pszSrcFile parameter. The pszDestFile parameter is ignored.
FO_MOVE	Moves the files specified by the pszSrcFile parameter to the location specified by the pszDestFile parameter.
FO_RENAME	Renames the files specified by the pszSrcFile parameter. The pszDestFile parameter is ignored.

Table 13-7: ICopyHook.CopyCallback wFlags values

Value	Description
FOF_ALLOWUNDO	The specified file is deleted to the recycle bin. If the pszSrcFile parameter does not contain a fully qualified path, this value is ignored.
FOF_FILESONLY	The operation is performed only on files if a wildcard filename is specified (i.e., "*.pas").
FOF_MULTIDESTFILES	The pszDestFile parameter contains one destination file for each source file instead of one directory to which all source files are deposited.
FOF_NOCONFIRMATION	The user is never asked for confirmation, and the operation continues as if a response of "yes to all" was indicated.
FOF_NOCONFIRMMKDIR	Automatically creates a new directory if one is needed without asking the user for confirmation.
FOF_NOCOPYSECURITYATTRIBS	**Windows NT/2000 and later**: Does not copy the file security attributes.
FOF_NOERRORUI	There is no visual indication if an error occurs.
FOF_NORECURSION	Performs the file operation on the files in the local directory only and does not continue file operations in subdirectories.
FOF_RENAMEONCOLLISION	The source file is automatically given a new name, such as "Copy #1 of..," in a move, copy, or rename operation if a file in the target directory already has the same name.
FOF_SILENT	Does not display a progress dialog box.
FOF_SIMPLEPROGRESS	Displays a progress dialog box but does not show filenames.
FOF_WANTMAPPINGHANDLE	The hNameMappings member receives a handle to a filename mapping object if any files were renamed. FOF_RENAMEONCOLLISION must be used in conjunction with this flag.
FOF_WANTNUKEWARNING	Displays a warning dialog box when a file is deleted.

Table 13-8: ICopyHook.CopyCallback dwSrcAttribs and dwDestAttribs values

Value	Description
FILE_ATTRIBUTE_READONLY	The object is read only.
FILE_ATTRIBUTE_HIDDEN	The object is hidden.
FILE_ATTRIBUTE_SYSTEM	The object is a system file.
FILE_ATTRIBUTE_DIRECTORY	The object is a directory folder.
FILE_ATTRIBUTE_ARCHIVE	The object is an archive file.
FILE_ATTRIBUTE_NORMAL	The object does not have any attributes.
FILE_ATTRIBUTE_TEMPORARY	The object is a temporary file.
FILE_ATTRIBUTE_COMPRESSED	The object is compressed.

Table 13-9: ICopyHook.CopyCallback return values

Value	Description
IDYES	Allow the operation to continue.
IDNO	Reject the operation on this folder or printer but allow any other approved operations.
IDCANCEL	Reject the operation and cancel all other operations.

IDropTarget.DragEnter *ActiveX.pas*

Syntax

```
DragEnter(
const dataObj: IDataObject;        {an IDataObject interface for source data}
grfKeyState: Longint;              {current keyboard state}
pt: TPoint;                        {mouse cursor coordinates}
var dwEffect: Longint              {drag-drop effect flags}
): HResult;                        {returns an OLE result code}
```

Description

This method is called when the mouse cursor dragging an object enters a potential drop target. It returns information indicating if the drop target is legal and the potential operation that will be performed.

Parameters

dataObj: A pointer to an IDataObject interface containing information and data about the object being dragged. Use this interface to retrieve information about the dragged item and determine if the object can be accepted.

Note: While the example demonstrates how the IDataObject is used to retrieve information about the dragged object, the IDataObject interface is not discussed fully in this text.

grfKeyState: A combination of flags that indicate if specific keys on the keyboard are currently depressed. This parameter may be one or more values from Table 13-10.

pt: A TPoint structure indicating the position of the mouse cursor, in screen coordinates (pixels).

dwEffect: A flag indicating the result of the drop operation. This parameter may be one value from Table 13-11.

Return Value

This method should return NOERROR if successful or an OLE error result code otherwise.

See Also

IDropTarget.DragLeave, IDropTarget.DragOver, IDropTarget.Drop

Example

Please see Listing 13-7 in the introduction.

Table 13-10: IDropTarget.DragEnter grfKeyState values

Value	Description
MK_CONTROL	The Control key is depressed.
MK_SHIFT	The Shift key is depressed.
MK_ALT	The Alt key is depressed.
MK_LBUTTON	The left mouse button is depressed.
MK_MBUTTON	The middle mouse button is depressed.
MK_RBUTTON	The right mouse button is depressed.

Table 13-11: IDropTarget.DragEnter dwEffect values

Value	Description
DROPEFFECT_LINK	Indicates a link operation (typically used when the Ctrl and Shift keys are depressed).
DROPEFFECT_COPY	Indicates a copy operation (typically used when the Ctrl key is depressed).
DROPEFFECT_MOVE	Indicates a move operation.
DROPEFFECT_NONE	Indicates no operation will be performed (typically used when the method fails).

IDropTarget.DragLeave ActiveX.pas

Syntax

```
DragLeave: HResult;           {returns an OLE result code}
```

Description

This method is called when the mouse leaves the drop target. Any drag operation feedback should be removed.

Return Value

This method should return NOERROR if successful or an OLE error code otherwise.

See Also

IDropTarget.DragEnter, IDropTarget.DragOver, IDropTarget.Drop

Example

Please see Listing 13-7 in the introduction.

IDropTarget.DragOver ActiveX.pas

Syntax

```
DragOver(
grfKeyState: Longint;         {current keyboard state}
pt: TPoint;                   {mouse cursor coordinates}
var dwEffect: Longint         {drag-drop effect flags}
): HResult;                   {returns an OLE result code}
```

Description

This method is called when the mouse cursor moves over a potential drop target. This method is called frequently, and any code in its implementation should be optimized and should not perform any time-consuming process.

Parameters

grfKeyState: A combination of flags that indicate if specific keys on the keyboard are currently depressed. This parameter may be one or more values from Table 13-12.

pt: A TPoint structure indicating the position of the mouse cursor, in screen coordinates (pixels).

dwEffect: A flag indicating the result of the drop operation. This parameter may be one value from Table 13-13.

Return Value

This method should return NOERROR if successful or an OLE error code otherwise.

See Also

IDropTarget.DragEnter, IDropTarget.DragLeave, IDropTarget.Drop

Example

Please see Listing 13-7 in the introduction.

Table 13-12: IDropTarget.DragOver grfKeyState values

Value	Description
MK_CONTROL	The Control key is depressed.
MK_SHIFT	The Shift key is depressed.
MK_ALT	The Alt key is depressed.
MK_LBUTTON	The left mouse button is depressed.
MK_MBUTTON	The middle mouse button is depressed.
MK_RBUTTON	The right mouse button is depressed.

Table 13-13: IDropTarget.DragOver dwEffect values

Value	Description
DROPEFFECT_LINK	Indicates a link operation (typically used when the Ctrl and Shift keys are depressed).
DROPEFFECT_COPY	Indicates a copy operation (typically used when the Ctrl key is depressed).
DROPEFFECT_MOVE	Indicates a move operation.
DROPEFFECT_NONE	Indicates no operation will be performed (typically used when the method fails).

IDropTarget.Drop ActiveX.pas

Syntax

```
Drop(
const dataObj: IDataObject;      {an IDataObject interface for source data}
grfKeyState: Longint;            {current keyboard state}
pt: TPoint;                      {mouse cursor coordinates}
var dwEffect: Longint            {drag-drop effect flags}
): HResult;                      {returns an OLE result code}
```

Description

This method is called when the dragged object is dropped over the drop target. It should perform the drop operation indicated by the dwEffect parameter.

Parameters

dataObj: A pointer to an IDataObject interface containing information and data about the object being dragged. Use this interface to retrieve information about the dragged item and perform the drop operation.

Note: While the example demonstrates how the IDataObject is used to retrieve information about the dragged object, the IDataObject interface is not discussed fully in this text.

grfKeyState: A combination of flags that indicate if specific keys on the keyboard are currently depressed. This parameter may be one or more values from Table 13-14.

pt: A TPoint structure indicating the position of the mouse cursor, in screen coordinates (pixels).

dwEffect: A flag indicating the drop operation to perform. This parameter may be one value from Table 13-15.

Return Value

This method should return NOERROR if successful or an OLE error result code otherwise.

See Also

IDropTarget.DragEnter, IDropTarget.DragLeave, IDropTarget.DragOver

Example

Please see Listing 13-7 in the introduction.

Table 13-14: IDropTarget.Drop grfKeyState values

Value	Description
MK_CONTROL	The Control key is depressed.
MK_SHIFT	The Shift key is depressed.
MK_ALT	The Alt key is depressed.
MK_LBUTTON	The left mouse button is depressed.
MK_MBUTTON	The middle mouse button is depressed.
MK_RBUTTON	The right mouse button is depressed.

Table 13-15: IDropTarget.Drop dwEffect values

Value	Description
DROPEFFECT_LINK	Indicates a link operation (typically used when the Ctrl and Shift keys are depressed).
DROPEFFECT_COPY	Indicates a copy operation (typically used when the Ctrl key is depressed).
DROPEFFECT_MOVE	Indicates a move operation.
DROPEFFECT_NONE	Indicates no operation will be performed (typically used when the method fails).

IExtractIcon.Extract ShlObj.pas

Syntax

```
Extract(
pszFile: PAnsiChar;              {icon location string}
nIconIndex: UINT;                {icon index}
out phiconLarge: HICON;          {the large icon handle}
out phiconSmall: HICON;          {the small icon handle}
nIconSize: UINT                  {size of icon}
): HResult;                      {returns an OLE result code}
```

Description

This method retrieves handles to a large and small icon for the file.

Parameters

pszFile: A null-terminated string indicating the location of the icon. This is the same value as that returned by IExtractIcon.GetIconLocation.

nIconIndex: The index of the icon. This is the same value as that returned by IExtractIcon.GetIconLocation.

phiconLarge: This parameter is set to the handle of the large icon that is loaded by this method.

phiconSmall: This parameter is set to the handle of the small icon that is loaded by this method.

nIconSize: Indicates the desired size of the icon, in pixels. The low-order word contains the size of the large icon, and the high-order word contains the size of the small icon.

Note: Icons are always square, so this size indicates both width and height.

Return Value

This method returns NOERROR if icons were extracted; otherwise, it returns S_FALSE to indicate that the shell should use a default icon.

See Also

IExtractIcon.GetIconLocation, LoadIcon

Example

Please see Listing 13-6 in the introduction.

IExtractIcon.GetIconLocation ShlObj.pas

Syntax

```
GetIconLocation(
uFlags: UINT;                    {icon flags}
szIconFile: PAnsiChar;          {icon location}
cchMax: UINT;                   {location buffer size}
out piIndex: Integer;           {icon index}
out pwFlags: UINT               {icon information flags}
): HResult;                     {returns an OLE result code}
```

Description

This function retrieves the location of a file containing the desired icon and the index of the icon within that file.

Parameters

uFlags: A combination of flags indicating the icon to retrieve. This parameter may be one or more values from Table 13-16.

szIconFile: A null-terminated string buffer that receives the path and filename of the file containing the icon to extract.

cchMax: Indicates the size of the buffer pointed to by the szIconFile parameter.

piIndex: This parameter receives the index of the icon within the file specified in the szIconFile parameter.

pwFlags: This parameter receives a combination of flags that indicate additional information about the indicated icon. This parameter may be one or more values from Table 13-17.

Return Value

This method returns NOERROR if successful or S_FALSE if the shell should use a default icon.

See Also

IExtractIcon.Extract, LoadIcon

Example

Please see Listing 13-6 in the introduction.

Table 13-16: IExtractIcon.GetIconLocation uFlags values

Value	Description
GIL_DEFAULTICON	Indicates a default icon.
GIL_FORSHELL	Indicates the icon is to be displayed in a shell folder.
GIL_FORSHORTCUT	Indicates the icon is to be used for a shortcut (do not apply a shortcut overlay).
GIL_OPENICON	Indicates the icon to be displayed should specify an open state. If this flag is not included, the icon should be in a closed state (typically used for folders).

Table 13-17: IExtractIcon.GetIconLocation pwFlags values

Value	Description
GIL_DONTCACHE	Instructs the shell not to cache the icon.
GIL_NOTFILENAME	Indicates that the specified icon location and icon index do not specify a valid icon. This causes the system to call IExtract-Icon.Extract.
GIL_PERINSTANCE	Indicates each file associated with the icon handler can have its own icon.

IQueryInfo.GetInfoFlags ShlObj.pas

Syntax

```
GetInfoFlags(
out pdwFlags: DWORD       {unused}
): HResult;               {returns an OLE result code}
```

Description

This method is not currently used but must return E_NOTIMPL in implementations.

Parameters

pdwFlags: This parameter is not used and should be ignored.

Return Value

This method must return E_NOTIMPL.

See Also

IQueryInfo.GetInfoTip

Example

Please see Listing 13-2 in the introduction.

IQueryInfo.GetInfoTip ShlObj.pas

Syntax

```
GetInfoTip(
dwFlags: DWORD;           {unused}
var ppwszTip: PWideChar   {the information tip}
): HResult;               {returns an OLE result code}
```

Description

This method retrieves the text to use for the information tip of the item.

Parameters

dwFlags: This parameter is not used and should be ignored.

ppwszTip: A null-terminated string indicating the information tip to display. The memory for this string must be allocated by using the IMalloc interface returned by the SHGetMalloc function.

Return Value

This method returns NOERROR if it succeeds; otherwise, it returns an OLE error code.

See Also

SHGetMalloc

Example

Please see Listing 13-2 in the introduction.

IShellExecuteHook.Execute ShlObj.pas

Syntax

```
Execute(
var ShellExecuteInfo: TShellExecuteInfo        {a TShellExecuteInfo structure}
): HResult;                                    {returns an OLE result code}
```

Description

This method is called any time the ShellExecute or ShellExecuteEx functions are called. This happens when a file is double-clicked in Explorer or when the Run dialog box is used.

Parameters

ShellExecuteInfo: A TShellExecuteInfo structure containing information about the action to perform on a specific file. The TShellExecuteInfo structure is defined as:

```
TShellExecuteInfo = record
      cbSize: DWORD;                {size of the structure in bytes}
      fMask: ULONG;                 {flags indicating how to use other members}
      Wnd: HWND;                    {a handle to a parent window}
      lpVerb: PAnsiChar;            {a pointer to a string describing the action}
      lpFile: PAnsiChar;            {a pointer to a filename or folder name}
      lpParameters: PAnsiChar;      {a pointer to executable file parameters}
      lpDirectory: PAnsiChar;       {a pointer to the default directory name}
      nShow: Integer;               {file display flags}
      hInstApp: HINST;              {a handle to an application instance}
```

These fields are optional:

```
      lpIDList: Pointer;            {a pointer to an item identifier list}
      lpClass: PAnsiChar;           {a pointer to the name of a file class or GUID}
      hkeyClass: HKEY;              {a handle to the file class registry key}
      dwHotKey: DWORD;              {the hot key associated with the application}
      hIcon: THandle;               {a handle to an icon for the file class}
      hProcess: THandle;            {a process handle for the newly launched
```

```
    application}
end;
```

cbSize: The size of the TShellExecuteInfo structure, in bytes. This member is set to SizeOf(TShellExecuteInfo).

fMask: A series of flags that indicate if the optional members of the structure are used. This member can be one or more values from Table 13-18.

Wnd: A handle to a parent window that receives message boxes if an error occurs.

lpVerb: A pointer to a null-terminated string specifying the action to perform on the file or folder indicated by the lpFile member. Table 13-19 lists the standard actions that can be performed on a file or folder. However, these actions are not limited to those listed in the table. This member is dependent on the actions registered for the document or application in the registry, and new actions can be created through the Options menu in the Windows Explorer.

Note: On Windows prior to Windows 2000, if this member is NIL, the default verb is used if it is valid and available in the registry for the indicated file; otherwise, the "open" verb is used by default. On Windows 2000 and later systems, if this member is NIL, it attempts the steps listed above, and if neither verb is available, it uses the first verb listed for this filename in the registry.

lpFile: A pointer to a null-terminated string containing the name of a document, executable file, or folder.

lpParameters: If the lpFile member indicates an executable file, this member contains a pointer to a null-terminated string specifying the command line parameters to pass to the executable file. The parameters are separated by spaces. If the lpFile member specifies a document or folder, this parameter is NIL.

lpDirectory: A pointer to a null-terminated string containing the path to the default directory. If this parameter is NIL, the current directory is used as the working directory.

nShow: A flag that determines how the executable file indicated by the lpFile member is to be displayed when it is launched. This parameter can be one value from Table 13-20.

hInstApp: If the handler executes the application, this member must be set to the hInstance handle of the new process.

These fields are optional:

lpIDList: A pointer to an item identifier list that uniquely identifies the executable file to launch. This member is ignored if the fMask member does not contain SEE_MASK_IDLIST.

lpClass: A pointer to a null-terminated string containing the name of a file class or globally unique identifier (GUID). This member is ignored if the fMask member does not contain SEE_MASK_CLASSNAME.

hkeyClass: A handle to the registry key for the file class. This member is ignored if the fMask member does not contain SEE_MASK_CLASSKEY.

dwHotKey: The hot key to associate with the launched executable file. The low-order word contains the virtual key code, and the high-order word contains a modifier flag. The modifier flag can be one or more values from Table 13-21. This member is ignored if the fMask member does not contain SEE_MASK_HOTKEY.

hIcon: A handle to an icon to use for the file class. This member is ignored if the fMask member does not contain SEE_MASK_ICON. Alternatively, if the fMask member contains SEE_MASK_HMONITOR, this member should contain a handle to the monitor upon which the application or document is displayed.

hProcess: If the method succeeds, this member should be set to the process handle of the application that was started. This member is set to zero if the fMask member does not contain SEE_MASK_NOCLOSEPROCESS or if no new process was launched.

Return Value

This method returns NOERROR to indicate that the hook completely processed the request; this prevents ShellExecute or ShellExecuteEx from performing any other processing. Return S_FALSE to let ShellExecute or ShellExecuteEx continue processing the request.

Note: In the event that NOERROR is returned, the hInstApp member of the TShellExecuteInfo structure should contain a value greater than 32 to prevent the system from displaying an error message.

See Also

ShellExecute, ShellExecuteEx

Example

Please see Listing 13-4 in the introduction.

Table 13-18: IShellExecuteHook.Execute ShellExecuteInfo.fMask values

Value	Description
SEE_MASK_CLASSKEY	Use the class key specified by the hkeyClass member. This flag overrides the SEE_MASK_CLASSNAME flag.
SEE_MASK_CLASSNAME	Use the class name specified by the lpClass member.
SEE_MASK_CONNECTNETDRV	The lpFile member specifies a Universal Naming Convention path.
SEE_MASK_DOENVSUBST	Expand any environment variables included in the lpFile or lpDirectory members.
SEE_MASK_FLAG_DDEWAIT	If a DDE conversation is started, wait for it to end before returning.
SEE_MASK_FLAG_NO_UI	Do not display error message boxes if errors occur.

Value	Description
SEE_MASK_HMONITOR	Indicates a specific monitor on a multimonitor system. Use the hIcon member to indicate the monitor. This flag cannot be combined with the SEE_MASK_ICON flag.
SEE_MASK_HOTKEY	Use the hotkey specified by the dwHotKey member.
SEE_MASK_ICON	Use the icon specified by the hIcon member. This flag cannot be combined with the SEE_MASK_HMONITOR flag.
SEE_MASK_IDLIST	Use the item identifier list specified by the lpIDList member.
SEE_MASK_INVOKEIDLIST	Use the item identifier list specified by the lpIDList member. If the lpIDList member is NIL, the function creates an item identifier list and launches the application. This flag overrides the SEE_MASK_IDLIST flag.
SEE_MASK_NOCLOSEPROCESS	Causes the hProcess member to receive a handle to the process started. The process continues to run after the ShellExecuteEx function ends.
SEE_MASK_NO_CONSOLE	For console applications, this creates a new console for the new process instead of inheriting the parent's console.
SEE_MASK_UNICODE	Indicates a Unicode application.

Table 13-19: IShellExecuteHook.Execute ShellExecuteInfo.lpVerb values

Value	Description
"find"	Initiates a search starting from the directory specified in the lpDirectory member.
"edit"	Opens the application associated with the file specified by the lpFile member and passes the file to it for editing.
"open"	Opens the file or folder or launches the executable file identified by the lpFile member.
"print"	Prints the document identified by the lpFile member. If the lpFile member identifies an executable file, it is launched as if a value of "open" had been specified.
"explore"	Opens a Windows Explorer window onto the folder identified by the lpFile parameter.

Table 13-20: IShellExecuteHook.Execute ShellExecuteInfo.nShow values

Value	Description
SW_HIDE	The window is hidden and another window is activated.
SW_MAXIMIZE	Maximizes the window.
SW_MINIMIZE	The window is minimized and the next top-level window in the relative z-order is activated.
SW_RESTORE	The window is activated and displayed in its original size and position.
SW_SHOW	The window is activated and displayed in its current size and position.

Value	Description
SW_SHOWDEFAULT	The window is shown based on the wShowWindow member of the TStartupInfo structure passed to the CreateProcess function by the program that started the application. This is used to set the initial show state of an application's main window. This flag should be used when showing the window for the first time if the application could be run from a shortcut. This flag will cause the window to be shown using the Run settings under the shortcut properties.
SW_SHOWMAXIMIZED	The window is activated and displayed in a maximized state.
SW_SHOWMINIMIZED	The window is activated and displayed as an icon.
SW_SHOWMINNOACTIVE	The window is displayed as an icon. The active window remains active.
SW_SHOWNA	The window is displayed in its current state. The active window remains active.
SW_SHOWNOACTIVE	The window is displayed in its most recent state. The active window remains active.
SW_SHOWNORMAL	This is the same as SW_RESTORE.

Table 13-21: IShellExecuteHook.Execute ShellExecuteInfo.dwHotKey modifier flag values

Value	Description
HOTKEYF_ALT	The Alt key must be held down.
HOTKEYF_CONTROL	The Ctrl key must be held down.
HOTKEYF_SHIFT	The Shift key must be held down.

IShellExtInit.Initialize ShlObj.pas

Syntax

```
Initialize(
pidlFolder: PItemIDList;     {item ID list}
lpdobj: IDataObject;         {an IDataObject interface}
hKeyProgID: HKEY             {registry key}
): HResult;                  {returns an OLE result code}
```

Description

This method is called by Explorer when initializing context menu or property sheet shell extensions. These handlers should perform any internal initialization in the implementation of this method.

Parameters

pidlFolder: For property sheet extensions, this parameter is NIL. For context menu extensions, this parameter is a pointer to an item identifier list for the folder containing the selected file object whose context menu is displayed.

lpdobj: A pointer to an IDataObject interface containing information and data about the selected file objects.

Note: While the example demonstrates how the IDataObject is used to retrieve information about the dragged object, the IDataObject interface is not discussed fully in this text.

hKeyProgID: Indicates the registry key for the file class of the selected object.

Return Value

This method returns NOERROR if it succeeds; otherwise, it returns an OLE error code.

See Also

IContextMenu.GetCommandString, IContextMenu.InvokeCommand, IContextMenu.QueryContextMenu

Example

Please see Listing 13-5 in the introduction.

IURLSearchHook.Translate   ShlObj.pas

Syntax

```
Translate(
lpwszSearchURL: PWideChar;      {URL address buffer}
cchBufferSize: DWORD            {size of the buffer}
): HResult;                     {returns an OLE result code}
```

Description

This method performs translation of Internet addresses entered into Explorer and is called when the browser cannot determine the protocol of the URL address. It is typically used to provide translation for custom URL protocols.

Parameters

lpwszSearchURL: This points to a null-terminated wide string indicating the URL address entered into the browser. This method should set this parameter to the translated URL address.

cchBufferSize: The size of the buffer pointed to by the lpwszSearchURL.

Return Value

This method returns one value from the following table.

See Also

ShellExecute, ShellExecuteEx

Example

Please see Listing 13-1 in the introduction.

Table 13-22: IURLSearchHook Translate return values

Value	Description
S_OK	Indicates that the URL address was fully translated, the lpwszSearchURL parameter contains a URL address to which the browser can navigate, and no other URL translation hooks are called.
S_FALSE	Indicates that the URL address was partially translated. The lpwszSearchURL parameter contains the partially translated URL, and the system will continue to call any other registered URL search hooks.
E_FAIL	Indicates that the URL address was not translated and the lpwszSearchURL parameter was not modified. The system will continue to call any other registered URL search hooks.

Chapter 14

Specialized Shell Functions

The Windows Shell API contains a plethora of functions that give the developer access to a wide variety of shell functionality. Some of these functions are very specific, giving the developer power to create applications that interact with certain parts of the shell in very specific ways. Certain functions allow the developer to extend the Control Panel by adding control panel applets. You can also create new appbars, windows that live on the side of the screen similar to the Windows taskbar. Even adding an icon to the taskbar tray area is possible with these functions. In this chapter, we'll examine many of these functions and more.

Control Panel Applications

Control panel applications are simply DLLs with an extension of ".cpl" that are called by the control panel process. This DLL may contain one or more applets, which appear within the control panel as individual icons. Typically, control panel applets take the role of configuration utilities for system-level functions and applications, such as the Date/Time or Desktop Themes applets. Other control panel applets may be used as configuration utilities for a family or group of applications, like Delphi's BDE Administrator. Whatever their function, control panel applets provide a useful metaphor for the developer to present the user with a tool in a common location for the purposes of configuration, maintenance, etc.

The CplApplet Function

Windows communicates with the control panel application DLL by sending messages to a specific exported function. The DLL must export only one function, and it must be called CplApplet. The definition of CplApplet is:

```
function CplApplet(hwndCPl: THandle; uMsg: DWORD;
                   lParam1, lParam2: Longint): Longint; stdcall;
```

No matter how many applets are presented by the control panel application DLL, all communication between Windows and the control panel application takes place through CplApplet. The first parameter, hwndCPl, is a handle to the control panel application window and can be used to establish ownership of dialog boxes and other windows that the control panel applet will display. The uMsg parameter contains the message identifier, and the last two parameters will vary depending on the specific message sent.

Control Panel Messages

The messages Windows sends to the CplApplet function occur in a specific order. Fortunately, the order in which the messages are sent is well defined, as is the method by which each message should be responded to. Windows begins sending messages to the control panel application when the control panel is first displayed and again when the user double-clicks its icon from within the control panel.

The very first message sent to CplApplet after Windows loads the DLL is CPL_INIT. At this time, the application should perform any application-wide initialization, such as allocating memory or initializing global variables. The CplApplet function should return a non-zero value to indicate that initialization was successful and processing should continue. If, for some reason, the application could not complete initialization, the CplApplet function should return zero. This causes Windows to terminate execution and unload the control panel application DLL.

The next message received by CplApplet is CPL_GETCOUNT. CplApplet should simply return the number of applets implemented by the control panel application.

After CPL_GETCOUNT, Windows sends CplApplet two separate messages: CPL_INQUIRE and CPL_NEWINQUIRE. These messages are sent a number of times equal to the value returned from the CPL_GETCOUNT message; in essence, these messages are sent once for each applet implemented by the control panel. The purpose of these messages is to retrieve information about the applet, which is displayed to the user in the control panel. For each message, the lParam1 parameter contains the zero-based index of the applet. The lParam2 parameter contains a pointer to either a TCplInfo or TNewCplInfo data structure, depending on the message sent (TCplInto for the CPL_INQUIRE message or TNewCplInfo for the CPL_NEWINQUIRE). The CplApplet function must fill out these structures with information such as the applet's icon, name, and a descriptive string (see the CplApplet function description for more information). Responding to CPL_INQUIRE is the typical method of presenting this information; most control panel applications should ignore the CPL_NEWINQUIRE message for better performance. Return a non-zero value to indicate that the messages were processed.

Note: The CPL_NEWINQUIRE message is sent before the CPL_INQUIRE message. If CplApplet returns a non-zero value when processing CPL_NEWINQUIRE, the CPL_INQUIRE message will not be sent (for this specific applet).

When the user double-clicks the control panel application icon, CplApplet receives a CPL_DBLCLICK message. This signals the applet to display its window. Any applet specific initialization can occur at this time. The lParam1 parameter contains the applet number, and the lParam2 parameter contains an application-defined value. Show the applet's window, and return a non-zero value to indicate the message was handled.

When the control panel is unloading the control panel application DLL, CplApplet receives the CPL_STOP message once for each applet. Again, lParam1 contains the applet number. Any applet-specific cleanup should occur at this time.

Finally, after all of the CPL_STOP messages have been sent, CplApplet receives one final message before the DLL is unloaded: CPL_EXIT. This allows the application to clean up any global initialization (such as deallocating memory).

Writing Control Panel Applications

To write a control panel application, begin by creating a new DLL project. Make sure that you specify a new extension of .cpl under project options (or include the {$E CPL} compiler directive somewhere). Once this is done, it's a simple matter of writing and exporting the CplApplet function. The following example demonstrates a simple control panel application implementing one applet. So that the mechanics of writing a control panel application are not confused with applet functionality, this example applet doesn't do anything other than display a dialog box.

Note: In order for the control panel applet to show up in the control panel, it must be copied to the Windows system directory (or the system32 directory for Windows NT/2000 and later systems).

Listing 14-1: A simple control panel application

The DLL project file:

```
library CplAppletXmple;
uses
  SysUtils,
  Classes,
  CplAppletMainU in 'CplAppletMainU.pas' {CplAppletMain};

{$E CPL}    // force an extension of .cpl

{$R extraresources.RES}

exports CPlApplet name 'CPlApplet';

begin
end.
```

The primary unit of the example application:

```
unit CplAppletMainU;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls, ShellCtrls,
  Cpl;  {Cpl is required for message constants and data structures}

type
```

```
  TCplAppletMain = class(TForm)
    Label1: TLabel;
    ShellTreeView1: TShellTreeView;
    ShellComboBox1: TShellComboBox;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  {the control panel applet entry point definition}
  function CplApplet(hwndCPl: THandle; uMsg: DWORD;
                     lParam1, lParam2: Longint): Longint; stdcall;


implementation

{$R *.dfm}

var
  CplAppletMainForm: TCplAppletMain;

{control panel applet entry point}
function CplApplet(hwndCPl: THandle; uMsg: DWORD;
                       lParam1, lParam2: Longint): Longint; stdcall;
begin
  case uMsg of
    {Sent when the control panel application is loaded. Any initialization
     (such as allocating memory) should occur at this time. The application
     should return a non-zero value to indicate that processing should
     continue.}
    CPL_INIT      : begin
                      Result := 1;  // tell it to keep processing
                    end;
    {Sent to retrieve the number of applets supported by the control panel DLL.
     The application should return the number of applets supported. lParam1 and
     lParam2 are not defined.}
    CPL_GETCOUNT  : begin
                      Result := 1;  // we only have one applet
                    end;
    {Sent to retrieve information about each applet within the control panel
     application DLL. This message is sent to the DLL as many times as was
     returned by the CPL_GETCOUNT message (essentially once per applet).
     Return a non-zero value to indicate that this message was handled.}
    CPL_INQUIRE   : begin
                      {these are resource ids; see the extraresources.rc
                       file}
                      PCPLInfo(lParam2)^.idIcon := 2;
                      PCPLInfo(lParam2)^.idName := 1;
                      PCPLInfo(lParam2)^.idInfo := 2;

                      {no application specific data}
                      PCPLInfo(lParam2)^.lData := 0;

                      Result := 1;  // continue processing
                    end;
    {Sent when the control panel application's icon is double-clicked, this
     indicates that the control panel should display its dialog box. Return a
     non-zero value to indicate that this message was handled.}
    CPL_DBLCLK    : begin
                      {create and show the dialog box}
                      CplAppletMainForm := TCplAppletMain.Create(nil);
                      CplAppletMainForm.ShowModal;
                      CplAppletMainForm.Free;

                      Result := 1;  // continue processing
                    end;
    {Sent when the control panel application is closing. This message is sent
     to the DLL as many times as was returned by the CPL_GETCOUNT message
     (essentially once per applet). Any applet specific cleanup
     (i.e., deallocating memory) should be performed at this time. Return a
     non-zero value to indicate this message was handled.}
    CPL_STOP      : begin
                      {nothing to cleanup}
                      Result := 1;
                    end;
    {Sent when the control panel application is closing, the control panel will
     send this message to the applet just before the call to FreeLibrary to
     unload the control panel application DLL. Any non-applet specific cleanup
     (such as deallocation of memory) should occur at this time. Return a
     non-zero value to indicate this message was processed.}
    CPL_EXIT      : begin
                      {nothing to cleanup}
                      Result := 1;
                    end;
    {Sent to retrieve information about each applet within the control panel
     application DLL. This message is sent to the DLL as many times as was
     returned by the CPL_GETCOUNT message (essentially once per applet).
     This message is typically ignored.}
    CPL_NEWINQUIRE : begin
                       Result := 0;
                     end;
    {Sent to indicate that the control panel should display its dialog box.
     However, this message may carry additional information regarding the
     behavior of its execution.}
    CPL_STARTWPARMS : begin
                        Result := 0;
                      end;
  else
    {Any other undocumented messages are not handled.}
    Result := 0;
  end;
end;

end.
```

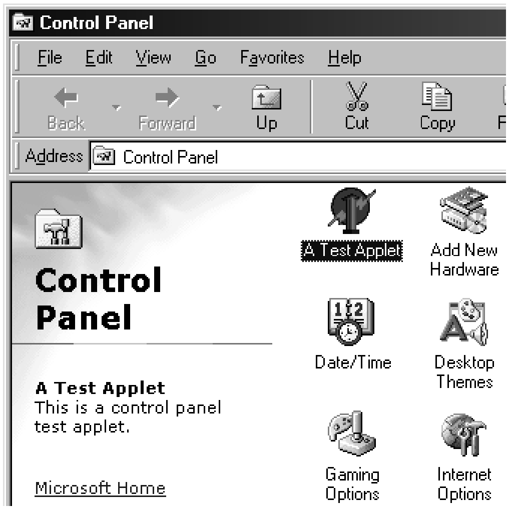


Figure 14-1: The test applet in the control panel

Application Bars

The Windows 95 and NT shells introduce a new user interface item known as an application bar. An appbar is a window that is associated with a particular edge of the screen. The space occupied by the appbar is reserved for its own use, and the system prevents other windows from using this area. There are several popular applications that ship with appbars, most of which provide the user with an alternative form of file management than that offered by the Start menu. The Windows taskbar is a special type of appbar.

The SHAppBarMessage function provides the interface to the Windows shell for registering an appbar and controlling its position. The application communicates with the shell through this function by sending it application bar messages. An appbar is registered by using the SHAppBarMessage function to send the system an ABM_NEW message. When an application creates an appbar, it should use the ABM_QUERYPOS message to retrieve an approved area for the appbar to reside. The ABM_SETPOS message is then used to inform the system that the appbar is occupying the specified rectangular area of the screen. The MoveWindow function is used to physically move the appbar window into the approved area. Once the appbar is in position, it receives appbar notification messages through the application-defined message identifier to inform it of events that might affect its appearance. These events include such things as a change in the state of the Windows taskbar or the launching or shutdown of a full screen application.

The appbar gives the Delphi developer an alternative to using a top-level window as the primary user interface. The following example demonstrates how Delphi can create a Windows application bar.

Listing 14-2: Creating an appbar using Delphi

```
unit AppBarMessageU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ShellAPI;

const
  {the application-defined appbar message identifier}
  WM_DELAPPBAR = WM_USER+1;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    procedure FormActivate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
  private
    { Private declarations }
    procedure CreateParams(var Params: TCreateParams); override;
  public
    { Public declarations }
    {the appbar message handler}
    procedure WMDelAppBar(var Msg: TMessage); message WM_DELAPPBAR;
  end;

var
  Form1: TForm1;
  {the TAppBarData structure must be global to the unit}
  AppBarInfo: TAppBarData;

implementation

{$R *.DFM}

{we must override the CreateParams method to insure the appropriate styles are used}
procedure TForm1.CreateParams(var Params: TCreateParams);
begin
  inherited CreateParams(Params);
  {the appbar must be a popup toolwindow to function properly}
  Params.ExStyle := WS_EX_TOOLWINDOW;
  Params.Style := WS_POPUP or WS_CLIPCHILDREN;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  {provide the TAppBarData structure with the handle to the appbar window}
  AppBarInfo.hWnd := Form1.Handle;
  {register the new appbar}
  SHAppBarMessage(ABM_NEW, AppBarInfo);
  {ask the system for an approved position}
  SHAppBarMessage(ABM_QUERYPOS, AppBarInfo);

  {adjust the new position to account for the appbar window height}
  AppBarInfo.rc.Bottom := AppBarInfo.rc.Top+50;

  {inform the system of the new appbar position}
  SHAppBarMessage(ABM_SETPOS, AppBarInfo);

  {physically move the appbar window into position}
  MoveWindow(AppBarInfo.hWnd, AppBarInfo.rc.Left, AppBarInfo.rc.Top,
             AppBarInfo.rc.Right-AppBarInfo.rc.Left,
             AppBarInfo.rc.Bottom-AppBarInfo.rc.Top, TRUE);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  {provide the TAppBarData structure with the required information}
  AppBarInfo.cbSize := SizeOf(TAppBarData);
  AppBarInfo.hWnd := Form1.Handle;
  AppBarInfo.lParam := 0;

  {unregister the appbar}
  SHAppBarMessage(ABM_REMOVE, AppBarInfo);
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
  {activate the appbar}
  SHAppBarMessage(ABM_ACTIVATE, AppBarInfo);
end;

procedure TForm1.FormPaint(Sender: TObject);
var
  Loop: Integer;
begin
  {this will fill the appbar with a gradient from yellow to red}
  for Loop := 0 to (Width div 20) do
  begin
    Canvas.Brush.Color := RGB(255,255-((255 div 20)*Loop),0);
    Canvas.Brush.Style := bsSolid;
    Canvas.FillRect(Rect((Width div 20)*Loop,0,((Width div 20)*Loop)+
                    (Width div 20),Height));
  end;

  {paint a caption on the appbar}
  Canvas.Font.Name   := 'Arial';
  Canvas.Font.Size   := 20;
  Canvas.Font.Color  := clBlue;
  Canvas.Font.Style  := [fsBold];
  Canvas.Brush.Style := bsClear;
  Canvas.TextOut(10,10,'Delphi App Bar');
end;

{this message handler is called whenever an event has
 occurred that could affect the appbar}
```

```
procedure TForm1.WMDelAppBar(var Msg: TMessage);
begin
  {the wParam parameter of the message contains the
   appbar notification message identifier}
  case Msg.wParam of
    ABN_FULLSCREENAPP: ShowMessage('FullScreenApp notification message
                                   received.');
    ABN_POSCHANGED: ShowMessage('PosChanged notification message received.');
    ABN_STATECHANGE: ShowMessage('StateChange notification message
                                   received.');
    ABN_WINDOWARRANGE: ShowMessage('WindowArrange notification message
                                   received.');
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Close;
end;

initialization
  {initialize the TAppBarData structure with the required information}
  AppBarInfo.uEdge := ABE_TOP;
  AppBarInfo.rc := Rect(0,0,GetSystemMetrics(SM_CXSCREEN),50);
  AppBarInfo.cbSize := SizeOf(TAppBarData);
  AppBarInfo.uCallbackMessage := WM_DELAPPBAR;

end.
```

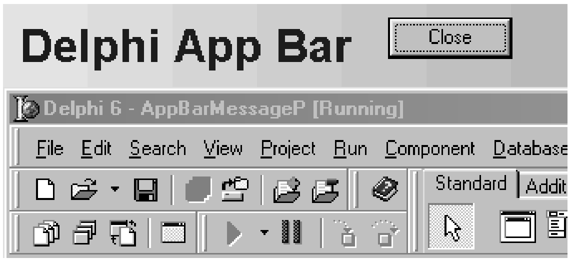

Delphi App Bar
Close
Delphi 6 - AppBarMessageP [Running]
File Edit Search View Project Run Component Database
Standard
Addit

Figure 14-2: The Delphi appbar in action

Tray Icon Applications

The taskbar tray notification area is a perfect spot for displaying a user interface hook for applications that run in the background. It is common practice now to add an icon to this notification area, displaying a dialog box or menu when the user clicks on the icon. Some examples of common applications that take this approach are the Task Scheduler, Volume Control, and anti-virus scanners. This is all accomplished through the use of the Shell_NotifyIcon API function.

Creating a Tray Icon Application

Creating a tray icon application is no different than creating any other type of application, with the exception that you will use Shell_NotifyIcon to actually add the icon to the notification tray. You will also need to provide a method handler for the message sent from the icon (more on this below).

To create a tray icon application, start by designing your application and implementing whatever functionality the application provides to the user. Then, in the Create method, you will fill out a TNotifyIconData data structure with some specific information about the application. This data structure requires a handle to the window that receives notification messages from the icon. You will also need to provide a unique identifier for the icon. This is required because an application can add multiple icons to the tray notification area; the identifiers let the application know which icon was activated by or manipulated by the user. Additionally, you need to provide an application-defined notification message. This is the message that is sent to the application when the icon is clicked and can be easily defined by using the WM_USER constant. Finally, by providing the handle to the application's icon (or whatever icon you wish to display), the text of the tooltip that is displayed when the user hovers the mouse over the icon, and some flags that indicate which members of the data structure are defined, you call the Shell_NotifyIcon function to display the icon in the tray notification area.

Messages

When the user interacts with the icon, the system sends the application-defined message to the window identified in the TNotifyIconData structure. The wParam member of the message holds the identifier of the icon with which the user interacted. The lParam member contains the actual message associated with the event.

Typically, these messages include various mouse and keyboard messages, such as WM_MOUSEMOVE, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_LBUTTONDBLCLICK, WM_KEYDOWN, or WM_KEYUP. However, under Windows 2000 and later, additional messages are sent depending on how the icon was activated. If the icon is selected by the keyboard and activated by pressing the Spacebar or Enter key, the system sends a NIN_KEYSELECT message. If the icon is selected by the mouse and activated by pressing the Spacebar or Enter key, the system sends a NIN_SELECT message. Additionally, if the tray icon is using balloon tooltips, several different messages are sent when the balloon tooltip is displayed or hidden.

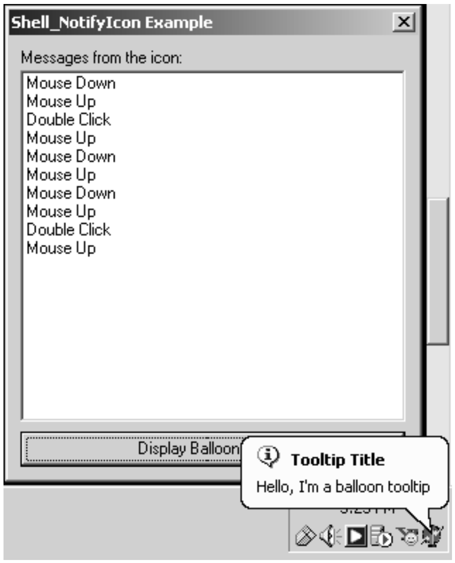
Balloon Tooltips

Windows 2000 and later introduced a new type of tooltip that the tray icon can display. This tooltip looks like a cartoon-style conversation balloon with a stem that points to the icon. In order to use this functionality, the system must have version 5.0 or later of shell32.dll (standard on Windows 2000 and later systems). To use a balloon tooltip, initialize the TNotifyIconData structure as outlined above. Some additional flags are necessary, and the szInfo, uTimeout, szInfoTitle, and dwInfoFlags members must be initialized. The szInfo member is set to the text that is displayed within the balloon.

The szInfoTitle is used to provide a title for the balloon tooltip. The uTimeout value, measured in milliseconds, indicates how long the balloon tooltip should be displayed before it hides itself. Finally, the dwInfoFlags member contains a flag indicating an icon to display in the balloon tooltip. Once all of this is done, use the NIM_MODIFY command to display the balloon tooltip. The following example demonstrates how to add a tray notification icon using balloon tooltips.

Listing 14-3: A tray notification icon using balloon tooltips

```
const
  WM_TRAYICONCLICKED = WM_USER+1; // the application-defined notification message

type
  TForm1 = class(TForm)
    ListBox1: TListBox;
    Button1: TButton;
    Label1: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    {the message handler for the tray icon notification message}
    procedure WMTrayIconClicked(var Msg: TMessage); message WM_TRAYICONCLICKED;
  public
    procedure DisplayBalloonTooltip;
  end;

var
  Form1: TForm1;
  IconData: TNotifyIconData;  // the tray notification icon data structure

const
  DELTRAYICON = 1;                // the tray icon ID

implementation

{$R *.DFM}

{NOTE: This example must be run under Windows 2000 or later to see the
 balloon tooltip}

procedure TForm1.FormCreate(Sender: TObject);
begin
  {initialize the tray notification icon structure}
  with IconData do
  begin
    cbSize           := SizeOf(TNotifyIconData);
    Wnd              := Form1.Handle;
    uID              := DELTRAYICON;
    uFlags           := NIF_ICON or NIF_MESSAGE or NIF_TIP;
    uCallbackMessage := WM_TRAYICONCLICKED;
    hIcon            := Application.Icon.Handle;
    szTip            := 'Delphi TrayIcon';
    dwState          := 0;
    dwStateMask      := 0;
  end;

  {notify the system that we are adding a tray notification icon}
  Shell_NotifyIcon(NIM_ADD, @IconData);

  {display the balloon tooltip when we start}
  DisplayBalloonTooltip;
end;
 procedure TForm1.WMTrayIconClicked(var Msg: TMessage);
begin
  {the tray icon has received a message, so display it}
  case Msg.lParam of
    WM_LBUTTONDBLCLK      : Listbox1.Items.Add('Double Click');
    WM_LBUTTONDOWN        : Listbox1.Items.Add('Mouse Down');
    WM_LBUTTONUP          : Listbox1.Items.Add('Mouse Up');
  end;
End;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  {remove the icon from the system}
  Shell_NotifyIcon(NIM_DELETE, @IconData);
end;

procedure TForm1.DisplayBalloonTooltip;
begin
  {initialize the structure to include the necessary information for
   displaying balloon tooltips}
  IconData.uFlags           := IconData.uFlags or NIF_INFO;
  IconData.szInfo           := 'Hello, I''m a balloon tooltip';
  IconData.uTimeout         := 5000;
  IconData.szInfoTitle      := 'Tooltip Title';
  IconData.dwInfoFlags      := NIIF_INFO;

  {display the balloon tooltip}
  Shell_NotifyIcon(NIM_MODIFY, @IconData);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  {display the balloon tooltip on command}
  DisplayBalloonTooltip;
end;
```



Mouse Down
Mouse Up
Double Click
Mouse Up
Mouse Down
Mouse Up
Mouse Down
Mouse Up
Double Click
Mouse Up

Figure 14-3: The balloon tooltip

Delphi vs. the Windows API

Delphi has a great template available in the repository for creating control panel applets. This template hides much of the complexity of creating control panel applets, and it is very easy to extend the control panel using this template. However, there is no functionality built in to Delphi's control panel applet template to allow dynamically changing the icon, title, and description. If you need this functionality from a control panel applet, you will have to use the Windows API.

However, most of the functions discussed in this chapter have no Delphi equivalent. If you want to create an appbar or add a tray notification icon, you will have to use the Windows API. Some of these functions are quite complex, but they are straightforward in their implementation and allow the developer to take advantage of certain Windows features that would otherwise be out of reach.

Specialized Shell Functions

The following specialized shell functions are covered in this chapter.

Table 14-1: Specialized shell functions

Function	Description
CplApplet	Entry point for control panel applications.
IMalloc.Alloc	Allocates a block of memory.
IMalloc.DidAlloc	Indicates if the block of memory was allocated by the calling instance of IMalloc.
IMalloc.Free	Frees an allocated block of memory.
IMalloc.GetSize	Retrieves the size of a block of memory.

Function	Description
IMalloc.HeapMinimize	Defragments the system memory heap.
IMalloc.Realloc	Reallocates a previously allocated block of memory.
SHAppBarMessage	Registers and controls an application bar.
SHChangeNotify	Indicates that the application has performed an action that may affect the shell.
ShellAbout	Displays the shell About dialog box.
ShellExecute	Launches an executable file.
ShellExecuteEx	Launches an executable file. This function provides more options than ShellExecute.
Shell_NotifyIcon	Registers a tray notification icon.
SHGetMalloc	Retrieves a pointer to the shell's IMalloc interface.

CplApplet Cpl.pas

Syntax

```
CplApplet(
hwndCPl: THandle;          {window handle of controlling application}
uMsg: DWORD;               {message identifier}
lParam1: Longint;          {message data}
lParam2: Longint           {message data}
): Longint;                {returns a message-dependent value}
```

Description

CplApplet is not an API function, but a user-defined entry point used by control panel applications. This is the function Windows calls within the control panel DLL, and so it is implemented by individual control panel application libraries.

Note: While CplApplet is not defined in the Cpl.pas unit (as it is not an API function per se), Cpl.pas is required in the Uses clause as it defines the constants and structures used by and referred to within calls to this function.

Parameters

hwndCPl: A window handle for the main window of the controlling application, intended to be used as the parent window for dialog boxes.

uMsg: The message sent to the control panel application. This can be one value from Table 14-2.

lParam1: Message-specific information. See Table 14-2.

lParam2: Message-specific information. See Table 14-2. With some specific messages, this may be a pointer to either a TCPLInfo structure or a TNewCPLInfo structure. These structures are defined as:

```
TCPLInfo = packed record
     idIcon: Integer;                    {icon resource identifier}
     idName: Integer;                    {string resource identifier}
     idInfo: Integer;                    {string resource identifier}
     lData : Longint;                    {application-defined data}
end;
```

idIcon: A resource identifier indicating the icon to display for the applet. This resource should be compiled into the DLL.

idName: A resource identifier indicating the name of the applet. This resource should be compiled into the DLL.

idInfo: A resource identifier indicating the description of the applet. This resource should be compiled into the DLL.

lData: This member can contain any longint value and is designed to provide application-specific data to an applet.

Note: Instead of actual resource identifiers, an application can use the constant CPL_DYNAMIC_RES for the idIcon, idName, and idInfo members. This indicates that these values can change and is useful for control panel applications that need to change their icon, name, or description based upon the current status of the system. If any of these members are set to CPL_DYNAMIC_RES, the control panel application is loaded every time information from any of these members is required, and a CPL_NEWINQUIRE message is sent to perform the query. Although this allows a control panel application to provide dynamic information based on changing system states, it will cause a performance decrease and users will notice significantly slower display times when opening the control panel.

```
TNewCPLInfo = packed record
     dwSize: DWORD;                          {size of structure}
     dwFlags: DWORD;                         {ignored}
     dwHelpContext: DWORD;                   {ignored}
     lData: Longint;                         {application-defined data}
     hIcon: HICON;                           {icon identifier}
     szName: array[0..31] of AnsiChar;       {applet name}
     szInfo: array[0..63] of AnsiChar;       {applet description}
     szHelpFile: array[0..127] of AnsiChar;  {ignored}
end;
```

dwSize: This member must be set to SizeOf(TNewCPLInfo).

dwFlags: Ignored, not used.

dwHelpContext: Ignored, not used.

lData: This member can contain any longint value and is designed to provide application-specific data to an applet.

hIcon: The identifier of an icon displayed for this applet.

szName: A null-terminated string containing the name of the applet.

szInfo: A null-terminated string containing the description of the applet. This is displayed in the control panel when the applet's icon is selected.

szHelpFile: Ignored, not used.

Return Value

This function returns a value that is dependent on the specific message. See the following table for more information.

See Also

DLLMain*

Example

Please see Listing 14-1 in the introduction.

Table 14-2: CplApplet uMsg values

Value	Description
CPL_DBLCLK	Sent when the control panel application's icon is double-clicked, this indicates that the control panel should display its dialog box. lParam1 contains the applet number, and lParam2 contains the user-defined lData value (see the CPL_INQUIRE message). Return a non-zero value to indicate this message was processed.
CPL_EXIT	Sent when the control panel application is closing. The control panel will send this message to the applet just before the call to FreeLibrary to unload the control panel application DLL. Any non-applet-specific cleanup (such as deallocation of memory) should occur at this time. lParam1 and lParam2 are not defined. Return a non-zero value to indicate this message was processed.
CPL_GETCOUNT	Sent to retrieve the number of applets supported by the control panel DLL. The application should return the number of applets supported. lParam1 and lParam2 are not defined.
CPL_INIT	Sent when the control panel application is loaded. Any initialization (such as allocating memory) should occur at this time. The application should return a non-zero value to indicate that processing should continue. If a zero is returned, the control panel applet DLL is released. lParam1 and lParam2 are not defined.
CPL_INQUIRE	Sent to retrieve information about each applet within the control panel application DLL. This message is sent to the DLL as many times as was returned by the CPL_GETCOUNT message (essentially, once per applet). lParam1 contains the applet number for the query (this is zero-based). lParam2 contains a pointer to a TCPLInfo structure. The idIcon field should be set to the resource ID for an icon to display (this resource should be linked into the DLL). The idName field should be set to the resource ID for a string that is displayed as the applet name. The idInfo field should be a resource ID for a string that is used to describe the applet. The final member, lData, can be set to any longint value (an application-defined value). Return a non-zero value to indicate this message was processed.

Value	Description
CPL_NEWINQUIRE	Sent to retrieve information about each applet within the control panel application DLL. This message is sent to the DLL as many times as was returned by the CPL_GETCOUNT message (essentially, once per applet). lParam1 contains the applet number for the query (this is zero-based). lParam2 contains a pointer to a TNewCPLInfo structure. The dwSize member must be set to SizeOf(TNewCPLInfo). The dwFlags and dwHelpContext members are ignored and are not used. The lpData member can be set to any longint value (an application-defined value). hIcon should be set to the handle of an icon to display. The szName member should be set to a null-terminated string that is displayed as the applet name. The szInfo member should be set to a null-terminated string that is used to describe the applet. The final member, szHelpFile, is ignored and not used. The CPL_NEWINQUIRE message is sent before the CPL_INQUIRE message, and if the message handler returns a non-zero value when processing CPL_NEWINQUIRE, the CPL_INQUIRE message will not be sent (for this specific applet). **Note**: Under Windows 95 and Windows NT, control panel applications should respond only to the CPL_INQUIRE message and should ignore the CPL_NEWINQUIRE message for better performance.
CPL_SELECT	Obsolete, and no longer used.
CPL_STARTWPARAMS	Similar in function to CPL_DBLCLICK, this message is sent to indicate that the control panel should display its dialog box. However, this message may carry additional information regarding the behavior of its execution. lParam1 contains the applet number. lParam2 contains a pointer to a string containing the extra execution information. Return a non-zero value to indicate this message was processed.
CPL_STOP	Sent when the control panel application is closing. This message is sent to the DLL as many times as was returned by the CPL_GETCOUNT message (essentially, once per applet). Any applet-specific cleanup (i.e., deallocating memory) should be performed at this time. lParam1 contains the applet number. lParam2 contains the user-defined lData value (see the CPL_INQUIRE message). Return a non-zero value to indicate this message was processed.

IMalloc.Alloc ActiveX.pas

Syntax

```
Alloc(
cb: Longint               {the size of the memory block}
): Pointer;               {returns a pointer to allocated memory}
```

Description

This method allocates a block of memory, returning a pointer to the allocated block.

Parameters

cb: Indicates the size of the memory block to allocate, in bytes.

Note: The actual memory size of the allocated block may be larger than the indicated value, as memory is allocated in 4-byte chunks so that it will be aligned along 32-bit boundaries.

Return Value

If the function succeeds, it returns a pointer to the newly allocated memory block; otherwise, it returns NIL.

See Also

IMalloc.Free, IMalloc.Realloc

Example

Listing 14-4: Using the shell's IMalloc interface to manipulate memory

```
procedure TForm1.Button1Click(Sender: TObject);
var
  SMalloc: IMalloc;       // holds a pointer to the shell's IMalloc interface
  MemBlock: Pointer;      // the example memory block
begin
  {retrieve a pointer to the shell's IMalloc interface}
  SHGetMalloc(SMalloc);

  {allocate a block of 100 bytes}
  MemBlock := SMalloc.Alloc(100);

  {if the memory block was successfully allocated (and it was allocated by
   this process)...}
  if (MemBlock  nil) and (SMalloc.DidAlloc(MemBlock) = 1) then
  begin
    {display the size of the memory block}
    ListBox1.Items.Add('Mem Size: ' + IntToStr(SMalloc.GetSize(MemBlock)));

    {now reallocate it to a smaller amount. note that while we've instructed
     the shell to allocate 50 bytes, it actually allocates 52 bytes because
     the memory allocated will align to 32-bit intervals}
    MemBlock := SMalloc.Realloc(MemBlock, 50);
    ListBox1.Items.Add('Mem Size: ' + IntToStr(SMalloc.GetSize(MemBlock)));

    {defragment the shell's memory heap}
    SMalloc.HeapMinimize;

    {free our memory block. Delphi will automatically release the IMalloc
     interface as soon as it goes out of scope (i.e., when the procedure exits)}
    SMalloc.Free(MemBlock);
  end;
end;
```

IMalloc.DidAlloc ActiveX.pas

Syntax

```
DidAlloc(
pv: Pointer             {a pointer to a memory block}
): Integer;             {returns an allocation flag}
```

Description

This method is used to determine if the calling instance of IMalloc allocated the specified memory block.

Parameters

pv: A pointer to the memory block.

Return Value

This function returns 1 if the specified memory block was allocated by the calling instance of IMalloc. It returns 0 if the block was not allocated by the calling instance. A –1 is returned if the function could not determine if the memory block was allocated by the calling instance or not.

See Also

IMalloc.Alloc, IMalloc.Realloc

Example

Please see Listing 14-4 under IMalloc.Alloc.

IMalloc.Free ActiveX.pas

Syntax

```
Free(
pv: Pointer             {a pointer to a memory block}
);                      {this procedure does not return a value}
```

Description

This method frees the specified memory block.

Parameters

pv: A pointer to a memory block to be freed.

See Also

IMalloc.Alloc, IMalloc.Realloc

Example

Please see Listing 14-4 under IMalloc.Alloc.

IMalloc.GetSize ActiveX.pas

Syntax

```
GetSize(
pv: Pointer              {a pointer to a memory block}
): Longint;              {returns the size, in bytes}
```

Description

This method returns the size of the specified memory block, in bytes.

Parameters

pv: A pointer to the memory block whose size is to be retrieved.

Return Value

If the function is successful, it returns the size of the specified memory block, in bytes; otherwise, it returns –1.

See Also

IMalloc.Alloc, IMalloc.Realloc

Example

Please see Listing 14-4 under IMalloc.Alloc.

IMalloc.HeapMinimize ActiveX.pas

Syntax

```
HeapMinimize;            {no parameters}
```

Description

This function defragments the system memory heap. Applications that have been running for a long time and allocating, reallocating, and freeing memory should call this function periodically.

See Also

IMalloc.Alloc, IMalloc.Free, IMalloc.Realloc

Example

Please see Listing 14-4 under IMalloc.Alloc.

IMalloc.Realloc ActiveX.pas

Syntax

```
Realloc(
pv: Pointer;            {a pointer to a block of memory}
cb: Longint             {the size of the new block of memory}
): Pointer;             {returns a pointer to allocated memory}
```

Description

This method reallocates a previously allocated block of memory, returning a pointer to the reallocated block.

Parameters

pv: A pointer to a block of memory previously allocated from a call to IMalloc.Alloc. If this parameter is NIL, the function allocates a new block of memory and otherwise acts like a call to IMalloc.Alloc.

cb: Indicates the size of the new block of memory, in bytes. The previous contents of the memory block remain unchanged. However, if the new size is larger than the previous size, the new areas within the block of memory will be undefined.

Note: The actual memory size of the allocated block may be larger than the indicated value, as memory is allocated in 4-byte chunks so that it will be aligned along 32-bit boundaries.

Return Value

If the function succeeds, it returns a pointer to the newly allocated block of memory. Otherwise, it returns NIL.

Note: Memory blocks can be moved by this function, so you should always use the returned pointer to refer to the new memory block.

See Also

IMalloc.Alloc, IMalloc.Free

Example

Please see Listing 14-4 under IMalloc.Alloc.

SHAppBarMessage ShellAPI.pas

Syntax

```
SHAppBarMessage(
dwMessage: DWORD;               {the appbar message}
var pData: TAppBarData          {a pointer to a TAppBarData data structure}
): UINT;                        {returns a message-dependent value}
```

Description

This function creates an application bar. An appbar is a window that is associated with a particular edge of the screen and reserves that screen space for its own use. Windows prevents other windows from using this space, moving them if necessary. Note that the application bar window must use the WS_EX_TOOLWINDOW and WS_POPUP styles to work properly.

Parameters

dwMessage: The application bar message identifier. This parameter can be one value from Table 14-3.

pData: A pointer to a TAppBarData data structure. This structure provides information to the SHAppBarMessage function and receives information as a result of the function call. The TAppBarData structure is defined as:

```
TAppBarData = record
      cbSize: DWORD;              {the size of the TAppBarData structure}
      hWnd: HWND;                 {a handle to a window}
      uCallbackMessage: UINT;     {an application-defined message identifier}
      uEdge: UINT;                {a screen edge flag}
      rc: TRect;                  {a rectangle in screen coordinates}
      lParam: LPARAM;             {a message-dependent value}
end;
```

cbSize: The size of the TAppBarData data structure, in bytes. This member should be set to SizeOf(TAppBarData).

hWnd: A handle to the appbar window.

uCallbackMessage: An application-defined message identifier. This message is sent to the window identified by the hWnd parameter to notify it of events. The wParam parameter of this message will contain one of the notification messages from Table 14-4.

uEdge: A flag indicating which edge of the screen is associated with the appbar. This member can be one value from Table 14-5.

rc: A TRect structure that holds the coordinates of the appbar window. These coordinates are in terms of the screen.

lParam: A message-dependent value. See Table 14-4 for an explanation of when this member is used.

Return Value

If the function succeeds, it returns a message-specific value; otherwise, it returns zero. See Table 14-3 for a description of possible return values.

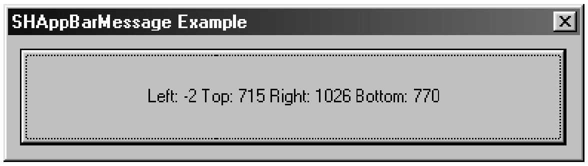
See Also

CreateWindow, CreateWindowEx, MoveWindow

Example

Listing 14-5: Retrieving the Windows taskbar coordinates

```
procedure TForm1.Button1Click(Sender: TObject);
var
  AppBarInfo: TAppBarData;   // holds the appbar information
begin
  {initialize the appbar data structure with the information needed}
  AppBarInfo.cbSize := SizeOf(TAppBarData);
  AppBarInfo.hWnd   := Form1.Handle;

  {retrieve the coordinates of the Windows taskbar...}
  SHAppBarMessage(ABM_GETTASKBARPOS, AppBarInfo);

  {...and display them}
  Button1.Caption := 'Left: '+IntToStr(AppBarInfo.rc.Left)+
                     ' Top: '+IntToStr(AppBarInfo.rc.Top)+
                     ' Right: '+IntToStr(AppBarInfo.rc.Right)+
                     ' Bottom: '+IntToStr(AppBarInfo.rc.Bottom);
end;
```



```
Left: -2 Top: 715 Right: 1026 Bottom: 770
```

Figure 14-4: The taskbar coordinates

Table 14-3: SHAppBarMessage dwMessage values

Value	Description
ABM_ACTIVATE	Notifies Windows that an appbar has been activated. The appbar should call this message when it receives a WM_ACTIVATE message. The cbSize and hWnd members of the TAppBarData structure must be initialized. All other members are ignored. This message is ignored if the hWnd parameter identifies an autohide appbar, as the system automatically sets their z-order. The function will always return a value greater than zero when using this message.
ABM_GETAUTOHIDEBAR	Retrieves the window handle of the autohide appbar associated with the specified screen edge. The cbSize, hWnd, and uEdge members of the TAppBarData structure must be initialized. All other members are ignored. If the function succeeds, it returns a handle to the appbar window associated with the specified screen edge. If the function fails, or there is no appbar associated with the specified screen edge, the function returns zero.

Value	Description
ABM_GETSTATE	Retrieves the autohide and always-on-top states of the Windows taskbar. The cbSize and hWnd members of the TAppBarData structure must be initialized. All other members are ignored. If the function succeeds, it returns either ABS_ALWAYSONTOP, a constant indicating the taskbar is in the always-on-top state, or ABS_AUTOHIDE, a constant indicating the taskbar is in the autohide state. The function can return both values if necessary. If the function fails, or the Windows taskbar is in neither state, it returns zero.
ABM_GETTASKBARPOS	Retrieves the bounding rectangular coordinates of the Windows taskbar. The cbSize and hWnd members of the TAppBarData structure must be initialized. All other members are ignored. If the function succeeds, it returns a value greater than zero, and the rc member will contain the bounding rectangle, in screen coordinates, of the Windows taskbar. If the function fails, it returns zero.
ABM_NEW	Registers a new appbar with the system. The function specifies the application-defined message identifier that is used to send the appbar notification messages. This message should be called before any other appbar messages. To register an autohide appbar, use the ABM_SETAUTOHIDEBAR message. The cbSize, hWnd, and uCallbackMessage members of the TAppBarData structure must be initialized. All other members are ignored. If the function succeeds, it returns a non-zero value. If the function fails, or the specified appbar is already registered, it returns zero.
ABM_QUERYPOS	Requests a bounding rectangle and screen edge position for the appbar. The system adjusts the specified rectangle so the appbar will not interfere with the Windows taskbar or any other appbar. The appbar should send this message before sending the ABM_SETPOS message. The cbSize, hWnd, uEdge, and rc members of the TAppBarData structure must be initialized. All other members are ignored. When the function returns, the rc member contains the adjusted coordinates for the new appbar position. This message causes the function to always return a non-zero value.
ABM_REMOVE	Unregisters an appbar from the system. The cbSize and hWnd members of the TAppBarData structure must be initialized. All other members are ignored. The function will always return a non-zero value when using this message. The ABN_POSCHANGED notification message is sent to all other appbars after this message is processed.
ABM_SETAUTOHIDEBAR	Registers or unregisters an autohide appbar. The system only allows one autohide appbar per screen edge. The lParam member of the TAppBarData structure is set to a non-zero value to register an autohide appbar or to zero to unregister the appbar. The cbSize, hWnd, uEdge, and lParam members of the TAppBarData structure must be initialized. All other members are ignored. If the function succeeds, it returns a non-zero value. If the function fails, or an appbar is already registered for the specified edge, the function returns zero.

Value	Description
ABM_SETPOS	Sets a bounding rectangle and screen edge position for the appbar. The system adjusts the specified rectangle so the appbar will not interfere with the Windows taskbar or any other appbar. The cbSize, hWnd, uEdge, and rc members of the TAppBarData structure must be initialized. All other members are ignored. When the function returns, the rc member contains the adjusted coordinates for the new appbar position. This message causes the function to always return a non-zero value. The system sends all appbars the ABN_POSCHANGED notification message after this message is processed.
ABM_WINDOWPOSCHANGED	Notifies the system that the appbar's position has changed. The appbar should call this message when responding to the WM_WINDOWPOSCHANGED message. The cbSize and hWnd members of the TAppBarData structure must be initialized. All other members are ignored. This message causes the function to always return a non-zero value. This message is ignored if the hWnd member identifies an autohide appbar.

Table 14-4: SHAppBarMessage pData.uCallbackMessage wParam notification messages

Value	Description
ABN_FULLSCREENAPP	Notifies an appbar when a full screen application is opening or closing. When a full screen application starts, the appbar must go to the bottom of the z-order. When the full screen application shuts down, the appbar can restore its original position in the z-order. If the lParam parameter is a non-zero value, it indicates that the full screen app is opening. If the lParam parameter is zero, the full screen app is shutting down.
ABN_POSCHANGED	Notifies the appbar of an event that may affect its size and position, such as adding, removing, or resizing another appbar, or changing the Windows taskbar position or state. Upon receiving this message, the appbar should send the ABM_QUERYPOS message followed by the ABM_SETPOS message to determine if its position has changed. The MoveWindow function is then called to physically move the appbar window into its new position.
ABN_STATECHANGE	Notifies the appbar that the taskbar's autohide or always-on-top state has changed.
ABN_WINDOWARRANGE	Notifies the appbar that the user has selected the Cascade, Tile Horizontally, or Tile Vertically command from the Windows taskbar context menu. If the lParam parameter is a non-zero value, it indicates that the arrangement command has started and no windows have been moved. A value of zero indicates that the arrangement command has finished and all windows are in their final positions. The appbar receives this message twice, once before the operation starts and again after the operation has finished.

Table 14-5: SHAppBarMessage pData.uEdge values

Value	Description
ABE_BOTTOM	The bottom edge of the screen.
ABE_LEFT	The left edge of the screen.
ABE_RIGHT	The right edge of the screen.
ABE_TOP	The top edge of the screen.

SHChangeNotify ShlObj.pas

Syntax

```
SHChangeNotify(
wEventId: Longint;          {event identifier}
uFlags: UINT;               {event-dependent flags}
dwItem1: Pointer;           {event-dependent value}
dwItem2: Pointer            {event-dependent value}
);                          {this procedure does not return a value}
```

Description

This function notifies the system of an action performed by the application that may affect the shell.

Parameters

wEventId: Flags indicating the type of event that has occurred and how it may affect the shell. This parameter may contain one or more values from Table 14-6.

uFlags: Flags indicating the meaning of the dwItem1 and dwItem2 parameters. This value is dependent on the value of the wEventId parameter and can contain one value from Table 14-7.

dwItem1: A value or a pointer to a value, dependent on the values of the wEventId and uFlags parameters. See Tables 14-6 and 14-7 for more details.

dwItem2: A value or a pointer to a value, dependent on the values of the wEventId and uFlags parameters. See Tables 14-6 and 14-7 for more details.

See Also

CreateDirectory, CreateDirectoryEx, CreateFile, DeleteFile, RemoveDirectory, SystemParametersInfo

Example

Listing 14-6: Creating a directory and informing the shell

```
{Note: this example creates a directory under the directory in which the
 example runs. thus, you cannot run this example straight off of the CD, you
 must copy the example to your hard drive first.}
procedure TForm1.Button1Click(Sender: TObject);
var
  CurDirectory: array[0..MAX_PATH] of char;
begin
```

```
  {get the current directory}
  GetCurrentDirectory(MAX_PATH, CurDirectory);

  {create a directory}
  CreateDirectory(PChar(string(CurDirectory) + '\DummyDirectory'), nil);

  {inform the system we've created a new directory}
  SHChangeNotify(SHCNE_MKDIR, SHCNF_PATH, PChar(string(CurDirectory) +
                 '\DummyDirectory'), nil);
end;
```

Table 14-6: SHChangeNotify wEventId values

Value	Description
SHCNE_ALLEVENTS	Indicates that all events have occurred.
SHCNE_ASSOCCHANGED	Indicates that a file association has changed. uFlags: Must contain SHCNF_IDLIST. dwItem1, dwItem2: Unused, set to NIL.
SHCNE_ATTRIBUTES	Indicates that the attributes of an item or folder have changed. uFlags: Must contain SHCNF_IDLIST or SHCNF_PATH. dwItem1: Indicates the item or folder that has changed. dwItem2: Unused; set to NIL.
SHCNE_CREATE	Indicates that a non-folder item has been created. uFlags: Must contain SHCNF_IDLIST or SHCNF_PATH. dwItem1: Indicates the created item. dwItem2: Unused; set to NIL.
SHCNE_DELETE	Indicates that a non-folder item has been deleted. uFlags: Must contain SHCNF_IDLIST or SHCNF_PATH. dwItem1: Indicates the deleted item. dwItem2: Unused; set to NIL.
SHCNE_DISKEVENTS	Indicates all of the disk events have occurred.
SHCNE_DRIVEADD	Indicates that a new drive has been added. uFlags: Must contain SHCNF_IDLIST or SHCNF_PATH. dwItem1: Indicates the root of the drive that was added. dwItem2: Unused; set to NIL.
SHCNE_DRIVEADDGUI	Indicates that a new drive has been added, and the shell should create a window for the drive. uFlags: Must contain SHCNF_IDLIST or SHCNF_PATH. dwItem1: Indicates the root of the drive that was added. dwItem2: Unused; set to NIL.

Value	Description
SHCNE_DRIVEREMOVED	Indicates that a drive has been removed.
	uFlags: Must contain SHCNF_IDLIST or SHCNF_PATH.
	dwItem1: Indicates the root of the drive that was removed.
	dwItem2: Unused; set to NIL.
SHCNE_FREESPACE	Indicates that the amount of free space on a drive has changed.
	uFlags: Must contain SHCNF_IDLIST or SHCNF_PATH.
	dwItem1: Indicates the root of the drive whose free space has changed.
	dwItem2: Unused; set to NIL.
SHCNE_GLOBALEVENTS	Indicates all of the global events have occurred.
SHCNE_INTERRUPT	Indicates that the specified event occurred as a result of a system interrupt. This value must be combined with other values from this table and cannot be used alone.
SHCNE_MEDIAINSERTED	Indicates storage media (i.e., a floppy disk or CD-ROM) has been inserted into a drive.
	uFlags: Must contain SHCNF_IDLIST or SHCNF_PATH.
	dwItem1: Indicates the root of the drive that has had new media inserted.
	dwItem2: Unused; set to NIL.
SHCNE_MEDIAREMOVED	Indicates storage media (i.e., a floppy disk or CD-ROM) has been removed from a drive.
	uFlags: Must contain SHCNF_IDLIST or SHCNF_PATH.
	dwItem1: Indicates the root of the drive that has had media removed.
	dwItem2: Unused; set to NIL.
SHCNE_MKDIR	Indicates a new directory has been created.
	uFlags: Must contain SHCNF_IDLIST or SHCNF_PATH.
	dwItem1: Indicates the new directory or folder.
	dwItem2: Unused; set to NIL.
SHCNE_NETSHARE	Indicates that a folder is now being shared over the network.
	uFlags: Must contain SHCNF_IDLIST or SHCNF_PATH.
	dwItem1: Indicates the directory or folder being shared.
	dwItem2: Unused; set to NIL.
SHCNE_NETUNSHARE	Indicates that a folder is no longer being shared over the network.
	uFlags: Must contain SHCNF_IDLIST or SHCNF_PATH.
	dwItem1: Indicates the directory or folder no longer being shared.
	dwItem2: Unused; set to NIL.

Value	Description
SHCNE_RENAMEFOLDER	Indicates that a folder has been renamed.
	uFlags: Must contain SHCNF_IDLIST or SHCNF_PATH.
	dwItem1: Contains the previous PIDL or name of the folder.
	dwItem2: Contains the new PIDL or name of the folder.
SHCNE_RENAMEITEM	Indicates that a non-folder item has been renamed.
	uFlags: Must contain SHCNF_IDLIST or SHCNF_PATH.
	dwItem1: Contains the previous PIDL or name of the item.
	dwItem2: Contains the new PIDL or name of the item.
SHCNE_RMDIR	Indicates that a folder has been deleted.
	uFlags: Must contain SHCNF_IDLIST or SHCNF_PATH.
	dwItem1: Indicates the directory or folder that was deleted.
	dwItem2: Unused; set to NIL.
SHCNE_SERVERDISCONNECT	Indicates that the computer has disconnected from a server.
	uFlags: Must contain SHCNF_IDLIST or SHCNF_PATH.
	dwItem1: Indicates the server that was disconnected.
	dwItem2: Unused; set to NIL.
SHCNE_UPDATEDIR	Indicates that the contents of a folder or directory have changed.
	uFlags: Must contain SHCNF_IDLIST or SHCNF_PATH.
	dwItem1: Indicates the folder or directory that has changed.
	dwItem2: Unused; set to NIL.
SHCNE_UPDATEIMAGE	Indicates that an image in the system image list has been changed.
	uFlags: Must contain SHCNF_DWORD.
	dwItem1: Indicates the index of the image in the system image list that was changed.
	dwItem2: Unused; set to NIL.
SHCNE_UPDATEITEM	Indicates that a non-folder item has changed.
	uFlags: Must contain SHCNF_IDLIST or SHCNF_PATH.
	dwItem1: Indicates the item that has changed.
	dwItem2: Unused; set to NIL.

Table 14-7: SHChangeNotify uFlags values

Value	Description
SHCNF_DWORD	Indicates that the dwItem1 and dwItem2 parameters are DWORD values.
SHCNF_FLUSH	Indicates that the function should not return until the notification has been delivered and processed by all affected objects.
SHCNF_FLUSHNOWAIT	Indicates that the function should return immediately (notification is delivered to affected objects asynchronously).
SHCNF_IDLIST	Indicates that the dwItem1 and dwItem2 parameters are pointers to item identifier lists. The PIDLs must be relative to the desktop folder.
SHCNF_PATH	Indicates that the dwItem1 and dwItem2 parameters are pointers to null-terminated strings containing qualified path and file names.
SHCNF_PRINTER	Indicates that the dwItem1 and dwItem2 parameters are pointers to null-terminated strings containing friendly printer names.

ShellAbout ShellAPI.pas

Syntax

```
ShellAbout(
Wnd: HWND;                  {a handle to a parent window}
szApp: PChar;               {a pointer to the title bar text}
szOtherStuff: PChar;        {a pointer to descriptive text}
Icon: HICON                 {a handle to an icon}
): Integer;                 {returns an integer value}
```

Description

This function displays the shell About dialog box. This is the About box that is displayed when About Windows is selected in the Windows Explorer. This dialog box displays an icon and text that is specific to the Windows operating system.

Parameters

Wnd: A handle to a parent window. If this parameter is zero, the About box acts like a modeless dialog box. If a handle is specified, the About box will be modal.

szApp: A pointer to text that is displayed in the title bar and on the first line of the About box.

szOtherStuff: A pointer to text that is displayed after the copyright information.

Icon: A handle to an icon that is displayed in the dialog box. If this parameter is zero, the dialog box will display the Windows or Windows NT icon.

Return Value

If the function succeeds, it returns a non-zero value; otherwise, it returns zero.

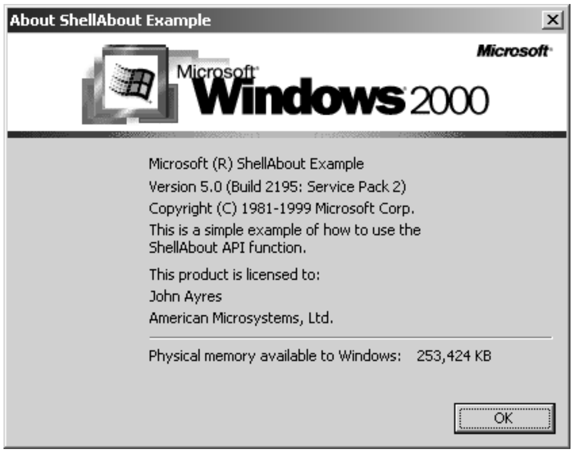
See Also

GetSystemInfo

Example

Listing 14-7: Displaying the ShellAbout dialog box

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  {call the Microsoft shell about box}
  ShellAbout(Form1.Handle, 'ShellAbout Example',
     'This is a simple example of how to use the ShellAbout API function.',0);
end;
```



Microsoft (R) ShellAbout Example
Version 5.0 (Build 2195: Service Pack 2)
Copyright (C) 1981-1999 Microsoft Corp.
This is a simple example of how to use the ShellAbout API function.
This product is licensed to:
John Ayres
American Microsystems, Ltd.
Physical memory available to Windows: 253,424 KB

Figure 14-5: The ShellAbout dialog box

ShellExecute *ShellAPI.pas*

Syntax

ShellExecute(	
hWnd: HWND;	{a handle to a parent window}
Operation: PChar;	{a pointer to a string describing the action}
FileName: PChar;	{a pointer to a filename or folder name}
Parameters: PChar;	{a pointer to executable file parameters}
Directory: PChar;	{a pointer to the default directory name}
ShowCmd: Integer	{file display flags}
): HINST;	{returns an instance handle}

Description

This function performs the specified action on the specified file and can be used to print a document, edit a file, search for a file, launch an executable file, or open a directory folder.

Parameters

hWnd: A handle to a parent window that receives message boxes if an error occurs.

Operation: A pointer to a null-terminated string specifying the action to perform on the file or folder indicated by the FileName parameter. Table 14-8 lists the standard actions that can be performed on a file or folder. However, these actions are not limited to those listed in the table. This parameter is dependent on the actions registered for the document or application in the registry, and new actions can be created through the Options menu in the Windows Explorer.

Note: On Windows prior to Windows 2000, if this parameter is NIL, the default verb is used if it is valid and available in the registry for the indicated file; otherwise, the "open" operation is used by default. On Windows 2000 and later systems, if this parameter is NIL, it attempts the steps listed above, and if neither verb is available, it uses the first verb listed for this filename in the registry.

FileName: A pointer to a null-terminated string containing the name of a document, executable file, or folder.

Parameters: If the FileName parameter indicates an executable file, this parameter contains a pointer to a null-terminated string specifying the command line parameters to pass to the executable file. The parameters must be separated by spaces. If the FileName parameter specifies a document or folder, this parameter should be NIL.

Directory: A pointer to a null-terminated string containing the path to the default directory. If this parameter is NIL, the current directory is used as the working directory.

ShowCmd: A flag that determines how the executable file indicated by the FileName parameter is to be displayed when it is launched. This parameter can be one value from Table 14-9.

Return Value

If the function succeeds, it returns the instance handle of the application that was launched or the handle to a dynamic data exchange server application. If the function fails, it returns a value from Table 14-10. This value will be less than 32.

See Also

FindExecutable, IShellExecuteHook.Execute, ShellExecuteEx

Example

Listing 14-8: Opening files

```
procedure TForm1.FileListBox1DblClick(Sender: TObject);
begin
   {open the file that was double-clicked in the file list box}
   ShellExecute(Form1.Handle, 'open', PChar(FileListBox1.FileName), nil, nil,
                SW_SHOWNORMAL);
end;
```

Table 14-8: ShellExecute operation values

Value	Description
“find”	Initiates a search starting from the specified directory.
“edit”	Opens the application associated with the file specified by the FileName parameter and passes the file to it for editing.
“open”	Opens the file or folder or launches the executable file identified by the FileName parameter.
“print”	Prints the document identified by the FileName parameter. If the FileName parameter identifies an executable file, it is launched as if a value of “open” had been specified.
“explore”	Opens a Windows Explorer window onto the folder identified by the FileName parameter.

Table 14-9: ShellExecute ShowCmd values

Value	Description
SW_HIDE	The window is hidden and another window is activated.
SW_MAXIMIZE	Maximizes the window.
SW_MINIMIZE	The window is minimized and the next top-level window in the relative z-order is activated.
SW_RESTORE	The window is activated and displayed in its original size and position.
SW_SHOW	The window is activated and displayed in its current size and position.
SW_SHOWDEFAULT	The window is shown based on the wShowWindow member of the TStartupInfo structure passed to the CreateProcess function by the program that started the application. This is used to set the initial show state of an application’s main window. This flag should be used when showing the window for the first time if the application could be run from a shortcut. This flag will cause the window to be shown using the Run settings under the shortcut properties.
SW_SHOWMAXIMIZED	The window is activated and displayed in a maximized state.
SW_SHOWMINIMIZED	The window is activated and displayed as an icon.
SW_SHOWMINNOACTIVE	The window is displayed as an icon. The active window remains active.
SW_SHOWNA	The window is displayed in its current state. The active window remains active.
SW_SHOWNOACTIVE	The window is displayed in its most recent state. The active window remains active.
SW_SHOWNORMAL	This is the same as SW_RESTORE.

Table 14-10: ShellExecute return value error codes

Value	Description
0	Not enough memory or resources.
ERROR_FILE_NOT_FOUND	The file specified by the FileName parameter could not be found.
ERROR_PATH_NOT_FOUND	The directory specified by the Directory parameter could not be found.
ERROR_BAD_FORMAT	The executable file is invalid or corrupt.
SE_ERR_ACCESSDENIED	Access to the specified file was denied.
SE_ERR_ASSOCINCOMPLETE	There is an incomplete or invalid executable file association for the specified file.
SE_ERR_DDEBUSY	The requested DDE transaction could not be completed due to other DDE transactions in progress.
SE_ERR_DDEFAIL	The requested DDE transaction failed.
SE_ERR_DDETIMEOUT	The requested DDE transaction failed because the DDE request timed out.
SE_ERR_DLLNOTFOUND	A required dynamic-link library could not be found.
SE_ERR_FNF	The file specified by the FileName parameter could not be found.
SE_ERR_NOASSOC	There is no executable file associated with the given filename extension. Also returned if the "print" verb is used on files that are not printable.
SE_ERR_OOM	The operation could not be completed due to insufficient memory.
SE_ERR_PNF	The directory specified by the Directory parameter could not be found.
SE_ERR_SHARE	A sharing violation has occurred.

ShellExecuteEx ShellAPI.ps

Syntax

```
ShellExecuteEx(
lpExecInfo: PShellExecuteInfo     {a pointer to a file execution information structure}
):BOOL;                           {returns TRUE or FALSE}
```

Description

Similar to ShellExecute, this function performs an action on a file and can be used to print a document, launch an executable file, or open a directory folder.

Parameters

lpExecInfo: A pointer to a TShellExecuteInfo structure. This structure contains information about the action to perform on a particular file and will receive information once the action is completed. The TShellExecuteInfo structure is defined as:

```
TShellExecuteInfo = record
    cbSize: DWORD;              {size of the structure in bytes}
    fMask: ULONG;               {flags indicating how to use other members}
    Wnd: HWND;                  {a handle to a parent window}
    lpVerb: PAnsiChar;          {a pointer to a string describing the action}
    lpFile: PAnsiChar;          {a pointer to a filename or folder name}
    lpParameters: PAnsiChar;    {a pointer to executable file parameters}
    lpDirectory: PAnsiChar;     {a pointer to the default directory name}
    nShow: Integer;             {file display flags}
    hInstApp: HINST;            {a handle to an application instance}
The following fields are optional:
    lpIDList: Pointer;          {a pointer to an item identifier list}
    lpClass: PAnsiChar;         {a pointer to the name of a file class or GUID}
    hkeyClass: HKEY;            {a handle to the file class registry key}
    dwHotKey: DWORD;            {the hot key associated with the application}
    hIcon: THandle;             {a handle to an icon for the file class}
    hProcess: THandle;          {a process handle for the newly launched
                                 application}
end;
```

cbSize: The size of the TShellExecuteInfo structure, in bytes. This member must be set to SizeOf(TShellExecuteInfo).

fMask: A series of flags that indicate if the optional members of the structure should be used. This member can be one or more values from Table 14-11.

Wnd: A handle to a parent window that receives message boxes if an error occurs.

lpVerb: A pointer to a null-terminated string specifying the action to perform on the file or folder indicated by the lpFile member. Table 14-12 lists the standard actions that can be performed on a file or folder. However, these actions are not limited to those listed in the table. This member is dependent on the actions registered for the document or application in the registry, and new actions can be created through the Options menu in the Windows Explorer.

Note: On Windows prior to Windows 2000, if this member is NIL, the default verb is used if it is valid and available in the registry for the indicated file; otherwise, the "open" verb is used by default. On Windows 2000 and later systems, if this member is NIL, it attempts the steps listed above, and if neither verb is available, it uses the first verb listed for this filename in the registry.

lpFile: A pointer to a null-terminated string containing the name of a document, executable file, or folder.

lpParameters: If the lpFile member indicates an executable file, this member contains a pointer to a null-terminated string specifying the command line parameters to pass to the executable file. The parameters must be separated by spaces. If the lpFile member specifies a document or folder, this parameter should be NIL.

lpDirectory: A pointer to a null-terminated string containing the path to the default directory. If this parameter is NIL, the current directory is used as the working directory.

nShow: A flag that determines how the executable file indicated by the lpFile member is to be displayed when it is launched. This parameter can be one value from Table 14-13.

hInstApp: If the function succeeds, this member contains a value greater than 32 upon return. If the function fails, this member will contain one of the values from Table 14-14.

The following fields are optional. These members do not have to be set in order for the ShellExecuteEx function to work properly.

lpIDList: A pointer to an item identifier list that uniquely identifies the executable file to launch. This member is ignored if the fMask member does not contain SEE_MASK_IDLIST.

lpClass: A pointer to a null-terminated string containing the name of a file class or globally unique identifier (GUID). This member is ignored if the fMask member does not contain SEE_MASK_CLASSNAME.

hkeyClass: A handle to the registry key for the file class. This member is ignored if the fMask member does not contain SEE_MASK_CLASSKEY.

dwHotKey: The hot key to associate with the launched executable file. The low-order word contains the virtual key code, and the high-order word contains a modifier flag. The modifier flag can be one or more values from Table 14-15. This member is ignored if the fMask member does not contain SEE_MASK_HOTKEY.

hIcon: A handle to an icon to use for the file class. This member is ignored if the fMask member does not contain SEE_MASK_ICON. Alternatively, if the fMask member contains SEE_MASK_HMONITOR, this member should contain a handle to the monitor upon which the application or document is displayed.

hProcess: If the function succeeds, upon return this member contains a process handle of the application that was started. This member is set to zero if the fMask member does not contain SEE_MASK_NOCLOSEPROCESS or if no new process was launched.

Return Value

If the function succeeds, it returns TRUE, and the hInstApp member of the TShellExecuteInfo structure contains an instance handle to the application that was started. If the function fails, it returns FALSE, and the hInstApp member will be set to one of the values from Table 14-14. To get extended error information, call the GetLastError function.

See Also

IShellExecuteHook.Execute, ShellExecute

Example

Listing 14-9: Another way to open files

```
procedure TForm1.FileListBox1DblClick(Sender: TObject);
var
   ExecInfo: TShellExecuteInfo;
begin
   {fill in the TShellExecuteInfo structure information}
   ExecInfo.cbSize       := SizeOf(TShellExecuteInfo);
   ExecInfo.fMask        := SEE_MASK_NOCLOSEPROCESS;
   ExecInfo.Wnd          := Form1.Handle;
   ExecInfo.lpVerb       := 'Open';
   ExecInfo.lpFile       := PChar(FileListBox1.FileName);
   ExecInfo.lpParameters := '';
   ExecInfo.lpDirectory  := '';
   ExecInfo.nShow        := SW_SHOWNORMAL;

   {open the specified file}
   ShellExecuteEx(@ExecInfo);
end;
```

Table 14-11: ShellExecuteEx lpExecInfo.fMask values

Value	Description
SEE_MASK_CLASSKEY	Use the class key specified by the hkeyClass member. This flag overrides the SEE_MASK_CLASSNAME flag.
SEE_MASK_CLASSNAME	Use the class name specified by the lpClass member.
SEE_MASK_CONNECTNETDRV	The lpFile member specifies a Universal Naming Convention path.
SEE_MASK_DOENVSUBST	Expand any environment variables included in the lpFile or lpDirectory members.
SEE_MASK_FLAG_DDEWAIT	If a DDE conversation is started, wait for it to end before returning.
SEE_MASK_FLAG_NO_UI	Do not display error message boxes if errors occur.
SEE_MASK_HMONITOR	Indicates a specific monitor on a multimonitor system. Use the hIcon member to indicate the monitor. This flag cannot be combined with the SEE_MASK_ICON flag.
SEE_MASK_HOTKEY	Use the hotkey specified by the dwHotKey member.
SEE_MASK_ICON	Use the icon specified by the hIcon member. This flag cannot be combined with the SEE_MASK_HMONITOR flag.
SEE_MASK_IDLIST	Use the item identifier list specified by the lpIDList member.
SEE_MASK_INVOKEIDLIST	Use the item identifier list specified by the lpIDList member. If the lpIDList member is NIL, the function creates an item identifier list and launches the application. This flag overrides the SEE_MASK_IDLIST flag.

Value	Description
SEE_MASK_NOCLOSEPROCESS	Causes the hProcess member to receive a handle to the process started. The process continues to run after the ShellExecuteEx function ends.
SEE_MASK_NO_CONSOLE	For console applications, this creates a new console for the new process instead of inheriting the parent's console.
SEE_MASK_UNICODE	Indicates a Unicode application.

Table 14-12: ShellExecuteEx lpExecInfo.lpVerb values

Value	Description
"find"	Initiates a search starting from the directory specified in the lpDirectory member.
"edit"	Opens the application associated with the file specified by the lpFile member and passes the file to it for editing.
"open"	Opens the file or folder or launches the executable file identified by the lpFile member.
"print"	Prints the document identified by the lpFile member. If the lpFile member identifies an executable file, it is launched as if a value of "open" had been specified.
"explore"	Opens a Windows Explorer window onto the folder identified by the lpFile parameter.

Table 14-13: ShellExecuteEx lpExecInfo.nShow values

Value	Description
SW_HIDE	The window is hidden and another window is activated.
SW_MAXIMIZE	Maximizes the window.
SW_MINIMIZE	The window is minimized and the next top-level window in the relative z-order is activated.
SW_RESTORE	The window is activated and displayed in its original size and position.
SW_SHOW	The window is activated and displayed in its current size and position.
SW_SHOWDEFAULT	The window is shown based on the wShowWindow member of the TStartupInfo structure passed to the CreateProcess function by the program that started the application. This is used to set the initial show state of an application's main window. This flag should be used when showing the window for the first time if the application could be run from a shortcut. This flag will cause the window to be shown using the Run settings under the shortcut properties.

Value	Description
SW_SHOWMAXIMIZED	The window is activated and displayed in a maximized state.
SW_SHOWMINIMIZED	The window is activated and displayed as an icon.
SW_SHOWMINNOACTIVE	The window is displayed as an icon. The active window remains active.
SW_SHOWNA	The window is displayed in its current state. The active window remains active.
SW_SHOWNOACTIVE	The window is displayed in its most recent state. The active window remains active.
SW_SHOWNORMAL	This is the same as SW_RESTORE.

Table 14-14: ShellExecuteEx lpExecInfo.hInstApp error codes

Value	Description
SE_ERR_ACCESSDENIED	Access to the specified file was denied.
SE_ERR_ASSOCINCOMPLETE	There is an incomplete or invalid executable file association for the specified file.
SE_ERR_DDEBUSY	The requested DDE transaction could not be completed due to other DDE transactions in progress.
SE_ERR_DDEFAIL	The requested DDE transaction failed.
SE_ERR_DDETIMEOUT	The requested DDE transaction failed because the DDE request timed out.
SE_ERR_DLLNOTFOUND	A required dynamic-link library could not be found.
SE_ERR_FNF	The file specified by the FileName parameter could not be found.
SE_ERR_NOASSOC	There is no executable file associated with the given filename extension.
SE_ERR_OOM	The operation could not be completed due to insufficient memory.
SE_ERR_PNF	The directory specified by the Directory parameter could not be found.
SE_ERR_SHARE	A sharing violation has occurred.

Table 14-15: ShellExecuteEx lpExecInfo.dwHotKey modifier flag values

Value	Description
HOTKEYF_ALT	The Alt key must be held down.
HOTKEYF_CONTROL	The Ctrl key must be held down.
HOTKEYF_SHIFT	The Shift key must be held down.

Shell_NotifyIcon ShellAPI.pas

Syntax

```
Shell_NotifyIcon(
dwMessage: DWORD;              {a notify icon message}
lpData: PNotifyIconData        {a pointer to a notify icon data structure}
): BOOL;                       {returns TRUE or FALSE}
```

Description

This function adds, modifies, or removes a notification icon from the taskbar system tray.

Parameters

dwMessage: A notification icon message identifier indicating the action to perform. This can be one value from Table 14-16.

lpData: A pointer to a TNotifyIconData data structure. This structure is defined as:

```
TNotifyIconData = record
      cbSize: DWORD;                         {the size of the TNotifyIconData structure}
      Wnd: HWND;                             {a handle to a window}
      uID: UINT;                             {an application-defined identifier}
      uFlags: UINT;                          {modification flags}
      uCallbackMessage: UINT;                {an application-defined message identifier}
      hIcon: HICON;                          {a handle to an icon}
      szTip: array [0..127] of AnsiChar;     {a tooltip string}
      dwState: DWORD;                        {icon state flags}
      dwStateMask: DWORD;                    {icon state bit mask}
      szInfo: array[0..255] of AnsiChar;     {balloon tooltip text}
      uTimeout: UINT;                     {balloon tooltip timeout (or behavior version)}
      szInfoTitle: array[0..63] of AnsiChar;      {balloon tooltip title}
      dwInfoFlags: DWORD                     {balloon tooltip icons}
end;
```

cbSize: The size of the TNotifyIconData structure, in bytes. This member should be set to SizeOf(TNotifyIconData).

Note: When running the application under versions of Windows prior to Windows 2000, this member should be set to SizeOf(ShellAPI.T-NotifyIcon).

Wnd: A handle to the window that receives notification messages when an event happens to the icon in the system tray. The value in the uID member indicates which icon is associated with the notification messages.

uID: An application-defined identifier for the notification icon.

uFlags: A combination of flags that indicate which other members of the TNotifyIconData structure are valid and should be used. This member can be any combination of the values in Table 14-17.

uCallbackMessage: An application-defined message. This message is sent to the window associated with the window handle set in the Wnd member whenever a mouse event happens in the icon in the system tray or when the icon is selected and activated by the keyboard. The wParam parameter of the message will contain the identifier of the icon affected by the message (as defined by the uID member above when the icon is first added). The lParam member contains the mouse or keyboard message identifier of the event that occurred. When running under Windows 2000 or later, the system will send various messages as outlined in Table 14-18.

hIcon: A handle to an icon to display in the system tray.

szTip: A pointer to a null-terminated string used as the tooltip text for the notification icon.

Note: On machines running a version of Windows prior to Windows 2000, this member can have a maximum of 64 characters.

dwState: **Windows 2000 or later**: A flag indicating the state of the icon. This member can be zero or one value from Table 14-19.

dwStateMask: **Windows 2000 or later**: A flag indicating which bits of the dwState member are to be retrieved or modified. This member can be set to one value from Table 14-19.

szInfo: **Windows 2000 or later**: A null-terminated string used as the text of a balloon tooltip. Set this member to NIL to remove the tooltip.

uTimeout: **Windows 2000 or later**: Indicates the time-out value for the balloon tooltip, in milliseconds. As of publication, Windows defines a minimum and maximum time- out of 10 seconds to 30 seconds, respectively. Any time-out value greater than or less than these values gets rounded to the nearest minimum or maximum value.

This member has an additional use. When the dwMessage parameter is set to NIM_SETVERSION, the value of this member instructs Windows which behavior version to use. Setting this member to zero indicates the system should use the Windows 95 behavior. Alternatively, setting this member to NOTIFYICON_VERSION indicates the system should use the Windows 2000 behavior.

Note: Setting the version on Windows prior to Windows 2000 has no effect.

szInfoTitle: **Windows 2000 or later**: A null-terminated string used as the title of a balloon tooltip.

dwInfoFlags: **Windows 2000 or later**: A flag indicating an icon to use in the balloon tooltip. This member may be set to one value from Table 14-20.

Return Value

If the function succeeds, it returns TRUE; otherwise, it returns FALSE.

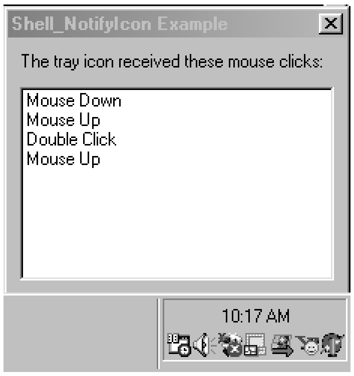
See Also

SHAppBarMessage

Example

Listing 14-10: Adding an icon to the system tray

```
const
  {Delphi does not define some of the new constants used in Shell_NotifyIcon}
  NIM_SETFOCUS    = $00000003;  // return focus to taskbar notification area
  NIM_SETVERSION  = $00000004;  // sets the behavior

  NIF_STATE       = $00000008;  // indicates dwState and dwStateMask are valid
  NIF_INFO        = $00000010;  // use a balloon tooltip

  NIS_HIDDEN      = $00000001;  // icon is hidden
  NIS_SHAREDICON  = $00000002;  // icon is shared

  NIIF_NONE       = $00000000;  // no icon
  NIIF_INFO       = $00000001;  // information icon
  NIIF_WARNING    = $00000002;  // warning icon
  NIIF_ERROR      = $00000003;  // error icon

  NOTIFYICON_VERSION = 3;       // using Windows 2000 behavior

  NIN_SELECT      = (WM_USER + 0);              // icon selected by the mouse
  NINF_KEY        = $1;
  NIN_KEYSELECT   = (NIN_SELECT or NINF_KEY); // icon selected by the keyboard

  NIN_BALLOONSHOW       = (WM_USER + 2);  // balloon is shown
  NIN_BALLOONHIDE       = (WM_USER + 3);  // balloon is hidden
  NIN_BALLOONTIMEOUT    = (WM_USER + 4);  // balloon hidden due to timeout
  NIN_BALLOONUSERCLICK  = (WM_USER + 5);  // balloon hidden due to mouse click

type
  {the new TNotifyIcon structure, required to take advantage of new
   functionality introduced by Windows 2000}
  PNotifyIconData = ^TNotifyIconData;
  TNotifyIconData = record
    cbSize: DWORD;                        {the size of the TNotifyIconData structure}
    Wnd: HWND;                            {a handle to a window}
    uID: UINT;                            {an application-defined identifier}
    uFlags: UINT;                         {modification flags}
    uCallbackMessage: UINT;               {an application-defined message identifier}
    hIcon: HICON;                         {a handle to an icon}
    szTip: array [0..127] of AnsiChar;    {a tooltip string}
    dwState: DWORD;                       {icon state flags}
    dwStateMask: DWORD;                   {icon state bit mask}
    szInfo: array[0..255] of AnsiChar;    {balloon tooltip text}
    uTimeout: UINT;                       {balloon tooltip timeout (or behavior version)}
    szInfoTitle: array[0..63] of AnsiChar; {balloon tooltip title}
    dwInfoFlags: DWORD                    {balloon tooltip icons}
  end;

const
  {the application-defined notification message}
  WM_TRAYICONCLICKED = WM_USER+1;

type
  TForm1 = class(TForm)
    ListBox1: TListBox;
    Label1: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
    {the message handler for the tray icon notification message}
    procedure WMTrayIconClicked(var Msg: TMessage); message WM_TRAYICONCLICKED;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  IconData: TNotifyIconData;  // the tray notification icon data structure

const
  DELTRAYICON = 1;            // the tray icon ID

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  {initialize the tray notification icon structure}
  with IconData do
  begin
    cbSize           := SizeOf(ShellAPI.TNotifyIconData);
    Wnd              := Form1.Handle;
    uID              := DELTRAYICON;
    uFlags           := NIF_ICON or NIF_MESSAGE or NIF_TIP;
    uCallbackMessage := WM_TRAYICONCLICKED;
    hIcon            := Application.Icon.Handle;
    szTip            := 'Delphi TrayIcon';
  end;

  {notify the system that we are adding a tray notification icon}
  Shell_NotifyIcon(NIM_ADD, @IconData);
end;

procedure TForm1.WMTrayIconClicked(var Msg: TMessage);
begin
  {the tray icon has received a message, so display it}
  case Msg.lParam of
     WM_LBUTTONDBLCLK: Listbox1.Items.Add('Double Click');
     WM_LBUTTONDOWN: Listbox1.Items.Add('Mouse Down');
     WM_LBUTTONUP: Listbox1.Items.Add('Mouse Up');
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  {remove the icon from the system}
  Shell_NotifyIcon(NIM_DELETE, @IconData);
end;
```



Mouse Down
Mouse Up
Double Click
Mouse Up

Figure 14-6: The new notification icon

Table 14-16: Shell_NotifyIcon dwMessage values

Value	Description
NIM_ADD	Adds a notification icon to the taskbar system tray.
NIM_DELETE	Deletes a notification icon from the taskbar system tray.
NIM_MODIFY	Modifies a notification icon in the taskbar system tray.
NIM_SETFOCUS	**Windows 2000 and later**: Returns the focus to the taskbar notification tray. This message should be used when notification icons complete any user interface operation.
NIM_SETVERSION	**Windows 2000 and later**: Indicates which version of tray icon behavior to use. The uTimeout member of the TNotifyIconData structure contains the value indicating the behavior (see above).

Table 14-17: Shell_NotifyIcon lpData.uFlags values

Value	Description
NIF_ICON	The icon handle in the hIcon member is valid.
NIF_MESSAGE	The message identifier in the uCallbackMessage member is valid.
NIF_TIP	The tooltip string pointed to by the szTip member is valid.
NIF_STATE	The state values in the dwState and dwStateMask members are valid.
NIF_INFO	Indicates the use of a balloon tooltip instead of a regular tooltip. The values in the szInfo, uTimeout, szInfoTitle, and dwInfoFlags members are valid.

Table 14-18: Shell_NotifyIcon lpData.uCallbackMessage values

Value	Description
NIN_KEYSELECT	**Windows 2000 or later**: The user selected the notification icon via the keyboard and activated it by pressing the Spacebar or Enter key.
NIN_SELECT	**Windows 2000 or later**: The user selected the notification icon with the mouse and activated it by pressing the Spacebar or Enter key.
NIN_BALLOONSHOW	**Windows 2000 or later**: The balloon tooltip has been shown.
NIN_BALLOONHIDE	**Windows 2000 or later**: The balloon tooltip has been hidden (not sent when the balloon tooltip was hidden due to timeout or the user clicking the mouse).
NIN_BALLOONTIMEOUT	**Windows 2000 or later**: The balloon tooltip was hidden due to timeout.
NIN_BALLOONUSERCLICK	**Windows 2000 or later**: The balloon tooltip was hidden because the user clicked the mouse.

Table 14-19: Shell_NotifyIcon lpData.dwState and lpData.dwStateMask values

Value	Description
NIS_HIDDEN	**Windows 2000 or later**: The icon is hidden.
NIS_SHAREDICON	**Windows 2000 or later**: The icon is shared.

Table 14-20: Shell_NotifyIcon lpData.dwInfoFlags values

Value	Description
NIIF_ERROR	Displays an error icon.
NIIF_INFO	Displays an info icon.
NIIF_NONE	No icon is displayed.
NIIF_WARNING	Displays a warning icon.

SHGetMalloc ShlObj.pas

Syntax

```
SHGetMalloc(
var ppMalloc: IMalloc        {IMalloc pointer}
): HResult;                  {returns an OLE result}
```

Description

This function retrieves a pointer to the shell's IMalloc interface.

Parameters

ppMalloc: A variable that receives a pointer to the shell's IMalloc interface.

Return Value

If the function succeeds, it returns S_OK; otherwise, it returns E_FAIL.

See Also

IMalloc.Alloc, IMalloc.DidAlloc, IMalloc.Free, IMalloc.GetSize, IMalloc.HeapMinimize, IMalloc.Realloc,

Example

Please see Listing 14-4 under IMalloc.Alloc.

Appendix A

Bibliography

There exists quite a large knowledge base on Windows programming in general and Delphi programming in particular. The information for this book is based in part on research and knowledge gleaned from the following books:

Beveridge and Wiener, *Multithreading Applications in Win32*, [Addison-Wesley Developers Press, 1997]

Calvert, Charles, *Delphi 2 Unleashed* [Sams Publishing, 1996]

Cluts, Nancy, *Programming The Windows 95 User Interface* [Microsoft Press, 1995]

Cooke and Telles, *Windows 95 How-To* [Waite Group Press, 1996]

Esposito, Dino, *Visual C++ Windows Shell Programming,* [Wrox Press, 1998]

Frerking, Wallace, and Niddery, *Borland Delphi How-To* [Waite Group Press, 1995]

Harmon, Eric, *Delphi COM Programming,* [Macmillan Technical Publishing, 2000]

Jarol, Haygood, and Coppola, *Delphi 2 Multimedia Adventure Set* [Coriolis Group Books, 1996]

Konopka, Ray, *Developing Custom Delphi 3 Components* [Coriolis Group Books, 1997]

Lischner, Ray, *Secrets of Delphi 2* [Waite Group Press, 1996]

Miller, Powell, et. al., *Special Edition Using Delphi 3* [QUE, 1997]

Pacheco and Teixeira, *Delphi 2 Developers Guide* [Sams Publishing, 1996]

Petzold and Yao, *Programming Windows 95* [Microsoft Press, 1996]

Pietrek, Matt, *Windows 95 System Programming Secrets* [IDG Books, 1995]

Rector and Newcomer, *Win32 Programming* [Addison-Wesley Developers Press, 1997]

Richter, Jeffrey, *Advanced Windows,* [Microsoft Press, 1997]

Roberts, Scott, *Programming Microsoft Internet Explorer 5,* [Microsoft Press, 1999]

Simon, Gouker, and Barnes, *Windows 95 Win32 Programming API Bible* [Waite Group Press, 1996]

Swan and Cogswell, *Delphi 32-Bit programming Secrets* [IDG Books, 1996]

Thorpe, Danny, *Delphi Component Design* [Addison-Wesley Developers Press, 1997]

Wallace and Tendon, *Delphi 2 Developer’s Solutions* [Waite Group Press, 1996]

Appendix B

Virtual Key Code Chart

Virtual Key Code	Decimal Value	Hex Value	Description
VK_LBUTTON	1	$1	Left mouse button
VK_RBUTTON	2	$2	Right mouse button
VK_CANCEL	3	$3	Ctrl+Break key combination
VK_MBUTTON	4	$4	Middle mouse button
VK_BACK	8	$8	Backspace
VK_TAB	9	$9	Tab
VK_CLEAR	12	$C	Numeric Keypad 5, NumLock off
VK_RETURN	13	$D	Enter
VK_SHIFT	16	$10	Shift
VK_CONTROL	17	$11	Ctrl
VK_MENU	18	$12	Alt
VK_PAUSE	19	$13	Pause
VK_CAPITAL	20	$14	Caps Lock
VK_ESCAPE	27	$1B	Esc
VK_SPACE	32	$20	Space bar
VK_PRIOR	33	$21	Page Up
VK_NEXT	34	$22	Page Down
VK_END	35	$23	End
VK_HOME	36	$24	Home
VK_LEFT	37	$25	Left cursor key
VK_UP	38	$26	Up cursor key
VK_RIGHT	39	$27	Right cursor key
VK_DOWN	40	$28	Down cursor key
VK_SNAPSHOT	44	$2C	Print Screen
VK_INSERT	45	$2D	Insert
VK_DELETE	46	$2E	Delete
VK_LWIN	91	$5B	Left Windows key on a Windows 95 compatible keyboard

Virtual Key Code	Decimal Value	Hex Value	Description
VK_RWIN	92	$5C	Right Windows key on a Windows 95 compatible keyboard
VK_APPS	93	$5D	Menu key on a Windows 95 compatible keyboard
VK_NUMPAD0	96	$60	Numeric keypad 0
VK_NUMPAD1	97	$61	Numeric keypad 1
VK_NUMPAD2	98	$62	Numeric keypad 2
VK_NUMPAD3	99	$63	Numeric keypad 3
VK_NUMPAD4	100	$64	Numeric keypad 4
VK_NUMPAD5	101	$65	Numeric keypad 5
VK_NUMPAD6	102	$66	Numeric keypad 6
VK_NUMPAD7	103	$67	Numeric keypad 7
VK_NUMPAD8	104	$68	Numeric keypad 8
VK_NUMPAD9	105	$69	Numeric keypad 9
VK_MULTIPLY	106	$6A	Numeric keypad multiply (*)
VK_ADD	107	$6B	Numeric keypad add (+)
VK_SUBTRACT	109	$6D	Numeric keypad subtract (–)
VK_DECIMAL	110	$6E	Numeric keypad decimal (.)
VK_DIVIDE	111	$6F	Numeric keypad divide (/)
VK_F1	112	$70	F1
VK_F2	113	$71	F2
VK_F3	114	$72	F3
VK_F4	115	$73	F4
VK_F5	116	$74	F5
VK_F6	117	$75	F6
VK_F7	118	$76	F7
VK_F8	119	$77	F8
VK_F9	120	$78	F9
VK_F10	121	$79	F10
VK_F11	122	$7A	F11
VK_F12	123	$7B	F12
VK_F13	124	$7C	F13
VK_F14	125	$7D	F14
VK_F15	126	$7E	F15
VK_F16	127	$7F	F16
VK_F17	128	$80	F17
VK_F18	129	$81	F18
VK_F19	130	$82	F19
VK_F20	131	$83	F20

Virtual Key Code	Decimal Value	Hex Value	Description
VK_F21	132	$84	F21
VK_F22	133	$85	F22
VK_F23	134	$86	F23
VK_F24	135	$87	F24
VK_NUMLOCK	144	$90	Num Lock
VK_SCROLL	145	$91	Scroll Lock
VK_LSHIFT	160	$A0	Left shift key
VK_RSHIFT	161	$A1	Right shift key
VK_LCONTROL	162	$A2	Left Ctrl key
VK_RCONTROL	163	$A3	Right Ctrl key
VK_LMENU	164	$A4	Left Alt key
VK_RMENU	165	$A5	Right Alt key

Appendix C

Tertiary Raster Operation Codes

ROP Code	Boolean Operation
$00000042	Result is all black
$00010289	NOT (brush OR source OR destination)
$00020C89	NOT (brush OR source) AND destination
$000300AA	NOT (brush OR source)
$00040C88	NOT (brush OR destination) AND source
$000500A9	NOT (brush OR destination)
$00060865	NOT (brush OR NOT(source XOR destination))
$000702C5	NOT (brush OR (source AND destination))
$00080F08	NOT brush AND source AND destination
$00090245	NOT (brush OR (source XOR destination))
$000A0329	NOT brush AND destination
$000B0B2A	NOT (brush OR (source AND NOT destination))
$000C0324	NOT brush AND source
$000D0B25	NOT (brush OR (NOT source AND destination))
$000E08A5	NOT (brush OR NOT (source OR destination))
$000F0001	NOT brush
$00100C85	brush AND NOT (source OR destination)
$001100A6	NOT (source OR destination)
$00120868	NOT (source OR NOT (brush XOR destination))
$001302C8	NOT (source OR (brush AND destination))
$00140869	NOT (destination OR NOT (brush XOR source))
$001502C9	NOT (destination OR (brush AND source))
$00165CCA	brush XOR (source XOR (destination AND NOT (brush AND source)))
$00171D54	NOT (source XOR ((source XOR brush) AND (source XOR destination)))
$00180D59	(brush XOR source) AND (brush XOR destination)
$00191CC8	NOT (source XOR (destination AND NOT (brush AND source)))
$001A06C5	brush XOR (destination OR (source AND brush))
$001B0768	NOT (source XOR (destination AND (brush XOR source)))

ROP Code	Boolean Operation
$001C06CA	brush XOR (source OR (brush AND destination))
$001D0766	NOT (destination XOR (source AND (brush XOR destination)))
$001E01A5	brush XOR (source OR destination)
$001F0385	NOT (brush AND (source OR destination))
$00200F09	brush AND NOT source AND destination
$00210248	NOT (source OR (brush XOR destination))
$00220326	NOT source AND destination
$00230B24	NOT (source OR (brush AND NOT destination))
$00240D55	(source XOR brush) AND (source XOR destination)
$00251CC5	NOT (brush XOR (destination AND NOT (source AND brush)))
$002606C8	source XOR (destination OR (brush AND source))
$00271868	source XOR (destination OR NOT (brush XOR source))
$00280369	destination AND (brush XOR source)
$002916CA	NOT (brush XOR (source XOR (destination OR (brush AND source))))
$002A0CC9	destination AND NOT (brush AND source)
$002B1D58	NOT (source XOR ((source XOR brush) AND (brush AND destination)))
$002C0784	source XOR (brush AND (source OR destination))
$002D060A	brush XOR (source OR NOT destination)
$002E064A	brush XOR (source OR (brush XOR destination))
$002F0E2A	NOT (brush AND (source OR NOT destination))
$0030032A	brush AND NOT source
$00310B28	NOT (source OR (NOT brush AND destination))
$00320688	source XOR (brush OR source OR destination)
$00330008	NOT source
$003406C4	source XOR (brush OR (source AND destination))
$00351864	source XOR (brush OR NOT (source XOR destination))
$003601A8	source XOR (brush OR destination)
$00370388	NOT (source AND (brush OR destination))
$0038078A	brush XOR (source AND (brush OR destination))
$00390604	source XOR (brush OR NOT destination)
$003A0644	source XOR (brush XOR (source XOR destination))
$003B0E24	NOT (source AND (brush OR NOT destination))
$003C004A	brush XOR source
$003D18A4	source XOR (brush OR NOT (source OR destination))
$003E1B24	source XOR (brush OR (NOT source AND destination))
$003F00EA	NOT (brush AND source)
$00400F0A	brush AND source AND NOT destination
$00410249	NOT (destination OR (brush XOR source))

ROP Code	Boolean Operation
$00420D5D	(source XOR destination) AND (brush XOR destination)
$00431CC4	NOT (source XOR (brush AND NOT (source AND destination)))
$00440328	source AND NOT destination
$00450B29	NOT (destination OR (brush AND NOT source))
$004606C6	destination XOR (source OR (brush AND destination))
$0047076A	NOT (brush XOR (source AND (brush XOR destination)))
$00480368	source AND (brush XOR destination)
$004916C5	NOT (brush XOR (destination XOR (source OR (brush AND destination))))
$004A0789	destination XOR (brush AND (source OR destination))
$004B0605	brush XOR (NOT source OR destination)
$004C0CC8	source AND NOT (brush AND destination)
$004D1954	NOT (source XOR ((brush XOR source) OR (source XOR destination)))
$004E0645	brush XOR (destination OR (brush XOR source))
$004F0E25	NOT (brush AND (NOT source OR destination))
$00500325	brush AND NOT destination
$00510B26	NOT (destination OR (NOT brush AND source))
$005206C9	destination XOR (brush OR (source AND destination))
$00530764	NOT (source XOR (brush AND (source XOR destination)))
$005408A9	NOT (destination OR NOT (brush OR source))
$00550009	NOT destination
$005601A9	destination XOR (brush OR source)
$00570389	NOT (destination AND (brush OR source))
$00580785	brush XOR (destination AND (brush OR source))
$00590609	destination XOR (brush OR NOT source)
$005A0049	brush XOR destination
$005B18A9	destination XOR (brush OR NOT (source OR destination))
$005C0649	destination XOR (brush OR (source XOR destination))
$005D0E29	NOT (destination AND (brush OR NOT source))
$005E1B29	destination XOR (brush OR (source AND NOT destination))
$005F00E9	NOT (brush AND destination)
$00600365	brush AND (source XOR destination)
$006116C6	NOT (destination XOR (source XOR (brush OR (source AND destination))))
$00620786	destination XOR (source AND (brush OR destination))
$00630608	source XOR (NOT brush OR destination)
$00640788	source XOR (destination AND (brush OR source))
$00650606	destination XOR (NOT brush OR source)
$00660046	source XOR destination
$006718A8	source XOR (destination OR NOT (brush OR source))

ROP Code	Boolean Operation
$006858A6	NOT (destination XOR (source XOR (brush OR NOT (source OR destination))))
$00690145	NOT (brush XOR (source XOR destination))
$006A01E9	destination XOR (brush AND source)
$006B178A	NOT (brush XOR (source XOR (destination AND (source OR brush))))
$006C01E8	source XOR (brush AND destination)
$006D1785	NOT (brush XOR (destination XOR (source AND (brush OR destination))))
$006E1E28	source XOR (destination AND (brush OR NOT source))
$006F0C65	NOT (brush AND NOT (source XOR destination))
$00700CC5	brush AND NOT (source AND destination)
$00711D5C	NOT (source XOR ((source XOR destination) AND (brush XOR destination)))
$00720648	source XOR (destination OR (brush XOR source))
$00730E28	NOT (source AND (NOT brush OR destination))
$00740646	destination XOR (source OR (brush XOR destination))
$00750E26	NOT (destination AND (NOT brush OR source))
$00761B28	source XOR (destination OR (brush AND NOT source))
$007700E6	NOT (source AND destination)
$007801E5	brush XOR (source AND destination)
$00791786	NOT (destination XOR (source XOR (brush AND (source OR destination))))
$007A1E29	destination XOR (brush AND (source OR NOT destination))
$007B0C68	NOT (source AND NOT (brush XOR destination))
$007C1E24	source XOR (brush AND (NOT source OR destination))
$007D0C69	NOT(destination AND NOT (source XOR brush))
$007E0955	(brush XOR source) OR (source XOR destination)
$007F03C9	NOT (brush AND source AND destination)
$008003E9	brush AND source AND destination
$00810975	NOT ((brush XOR source) OR (source XOR destination))
$00820C49	NOT (brush XOR source) AND destination
$00831E04	NOT (source XOR (brush AND (NOT source OR destination)))
$00840C48	source AND NOT (brush XOR destination)
$00851E05	NOT (brush XOR (destination AND (NOT brush OR source)))
$008617A6	destination XOR (source XOR (brush AND (source OR destination)))
$008701C5	NOT (brush XOR (source and destination))
$00800C6	source AND destination
$00891B08	NOT (source XOR (destination OR (brush AND NOT source)))
$008A0E06	(NOT brush OR source) AND destination
$008B0666	NOT(destination XOR (source OR (brush OR destination)))
$008C0E08	source AND (NOT brush OR destination)
$008D0668	NOT (source XOR (destination OR (brush XOR source)))

ROP Code	Boolean Operation
$008E1D7C	source XOR ((source XOR destination AND (brush XOR destination))
$008F0CE5	NOT (brush AND NOT (source AND destination))
$00900C45	brush AND NOT (source XOR destination)
$00911E08	NOT (source XOR (destination AND (brush OR NOT source)))
$009217A9	destination XOR (brush XOR (source AND (brush OR destination)))
$009301C4	NOT (source XOR (brush AND destination))
$009417AA	brush XOR (source XOR (destination AND (brush OR source)))
$009501C9	NOT (destination XOR (brush AND source))
$00960169	brush XOR source XOR destination
$0097588A	brush XOR (source XOR (destination OR NOT (brush OR source)))
$00981888	NOT (source XOR (destination OR NOT (brush OR source)))
$00990066	NOT (source XOR destination)
$009A0709	(brush AND NOT source)XOR destination
$009B07A8	NOT (source XOR (destination AND (brush OR source)))
$009C0704	source XOR (brush AND NOT destination)
$009D07A6	NOT (destination XOR (source AND (brush OR destination)))
$009E16E6	(source XOR (brush OR (source AND destination)))XOR destination
$009F0345	NOT(brush AND (source XOR destination))
$00A000C9	brush AND destination
$00A11B05	NOT (brush XOR (destination OR (NOT brush AND source)
$00A20E09	(brush OR NOT source) AND destination
$00A30699	NOT (destination XOR (brush OR (source XOR destination)))
$00A41885	NOT (brush XOR (destination OR NOT (brush OR source)))
$00A50065	NOT (brush XOR destination)
$00A60706	(NOT brush AND source) XOR destination
$00A707A5	NOT (brush XOR (destination AND (brush OR source)))
$00A803A9	(brush OR source) AND destination
$00A90189	NOT ((brush OR source) XOR destination)
$00AA0029	destination
$00AB0889	NOT(brush OR source) OR destination
$00AC0744	source XOR (brush AND (source XOR destination))
$00AD06E9	NOT (destination XOR (brush OR (source AND destination)))
$00AE0B06	(NOT brush AND source) OR destination
$00AF0229	NOT brush OR destination
$00B00E05	brush AND (NOT source OR destination)
$00B10665	NOT (brush OR (destination OR (brush XOR source)))
$00B12974	source XOR ((brush XOR source) OR (source XOR destination))
$00B03CE8	NOT (source AND NOT (brush AND destination))

ROP Code	Boolean Operation
$00B4070A	brush XOR (source AND NOT destination)
$00B507A9	NOT (destination XOR (brush AND (source OR destination)))
$00B616E9	destination XOR (brush XOR (source OR (brush AND destination)))
$00B70348	NOT (source And (brush XOR destination))
$00B8074A	brush XOR (source AND (brush XOR destination))
$00B906E6	NOT (destination XOR (source OR (brush AND destination)))
$00BA0B09	(brush AND NOT source) OR destination
$00BB0226	NOT source OR destination
$00BC1CE4	source XOR (brush AND NOT (source AND destination))
$00BD0D7D	NOT ((brush XOR destination) AND (source XOR destination))
$00BE0269	(brush XOR source) OR destination
$00BF08C9	NOT (brush AND source) OR destination
$00C000CA	brush AND source
$00C11B04	NOT (source XOR (brush OR (NOT source AND destination)))
$00C21884	NOT (source XOR (brush OR NOT(source OR destination)))
$00C3006A	NOT (brush XOR source)
$00C40E04	source AND (brush OR NOT destination)
$00C50664	NOT (source XOR (brush OR (source XOR destination)))
$00C60708	source XOR (NOT brush AND destination)
$00C707AA	NOT (brush XOR (source AND (brush OR destination))
$00C803A8	source AND (brush OR destination)
$00C90184	NOT (source XOR (brush OR destination))
$00CA0749	destination XOR (brush AND (source XOR destination))
$00CB06E4	NOT (source XOR (brush OR (source AND destination))
$00CC0020	source
$00CD0888	source OR NOT (brush OR destination)
$00CE0B08	source OR (NOT brush AND destination)
$00CF0224	source OR NOT brush
$00D00E0A	brush AND (source OR NOT destination)
$00D1066A	NOT (brush XOR (source OR (brush XOR destination)))
$00D20705	brush XOR (NOT source AND destination)
$00D307A4	NOT (source XOR (brush AND (source OR destination)))
$00D41D78	source XOR ((brush XOR source AND (brush XOR destination))
$00D50CE9	NOT (destination AND NOT (brush AND source))
$00D616EA	brush XOR (source XOR (destination OR (brush AND source)))
$00D70349	NOT (destination AND (brush XOR source))
$00D80745	brush XOR (destination AND (brush XOR source))
$00D906E8	NOT (source XOR (destination OR (brush AND source)))

ROP Code	Boolean Operation
$00DA1CE9	destination XOR (brush AND NOT (source XOR destination))
$00DB0D75	NOT ((brush XOR source) AND (source XOR destination)
$00DC0B04	source OR (brush AND NOT destination)
$00DD0228	source OR NOT destination
$00DE0268	source OR (brush XOR destination)
$00DF08C8	source OR NOT (brush AND destination)
$00E003A5	brush AND (destination OR source)
$00E10185	NOT (brush XOR (source OR destination))
$00E20746	destination XOR (source AND (brush XOR destination))
$00E306EA	NOT (brush XOR (source OR (brush AND destination)))
$00E40748	source XOR (destination AND (brush XOR source))
$00E506E5	NOT (brush XOR (destination OR (brush AND source)
$00E61CE8	source XOR (destination AND NOT (brush AND source))
$00E70D79	NOT ((brush XOR source) AND (brush XOR destination))
$00E81D74	source XOR ((brush XOR source) AND (source XOR destination))
$00E95CE6	NOT (destination XOR (source XOR (brush AND NOT (source AND destination))))
$00EA02E9	(brush AND source) OR destination
$00EB0849	NOT (brush XOR source) OR destination
$00EC02E8	source OR (brush AND destination)
$00ED0848	source OR NOT (brush XOR destination)
$00EE0086	source OR destination
$00EF0A08	NOT brush OR source OR destination
$00F00021	brush
$00F10885	brush OR NOT (source OR destination)
$00F20B05	brush OR (NOT source AND destination)
$00F3022A	brush OR NOT source
$00F40B0A	brush OR (source AND NOT destination)
$00F50225	brush OR NOT destination
$00F60265	brush OR (source XOR destination)
$00F708C5	brush OR NOT (source AND destination)
$00F802E5	brush OR (source AND destination)
$00F90845	brush OR NOT (source XOR destination)
$00FA0089	brush OR destination
$00FB0A09	brush OR NOT source OR destination
$00FC008A	brush OR source
$00FD0A0A	brush OR source OR NOT destination
$00FE02A9	brush OR source OR destination
$00FF0062	Result is all white

Appendix D

ASCII Character Set

Dec	Hex	Char	Description
0	00	NULL	Null
1	01	☺	Start of heading
2	02	☻	Start of text
3	03	♥	End of text
4	04	♦	End of transmission
5	05	♣	Inquiry
6	06	♠	Acknowledge
7	07	•	Bell
8	08	◘	Backspace
9	09	○	Horizontal tab
10	0A	◙	Line feed
11	0B	♂	Vertical tab
12	0C	♀	Form feed
13	0D	♪	Carriage return
14	0E	♬	Shift out
15	0F	☼	Shift in
16	10	►	Data link escape
17	11	◄	Device control 1
18	12	↕	Device control 2
19	13	‼	Device control 3
20	14	¶	Device control 4
21	15	§	Negative acknowledge
22	16	▬	Synchronous idle
23	17	↨	End transmission block
24	18	↑	Cancel
25	19	↓	End of medium
26	1A	→	Substitute
27	1B	←	Escape
28	1C	∟	File separator

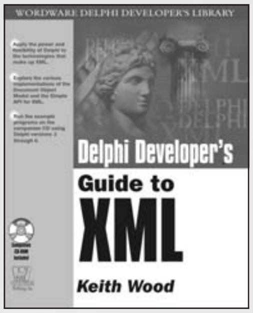
Dec	Hex	Char	Description
29	1D	↔	Group separator
30	1E	▲	Record separator
31	1F	▼	Unit separator

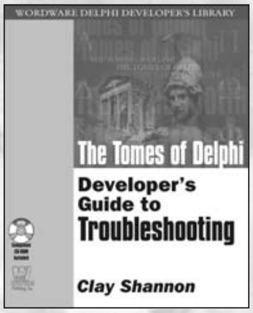
Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
32	20	SPACE	65	41	A	98	62	b
33	21	!	66	42	B	99	63	c
34	22	"	67	43	C	100	64	d
35	23	#	68	44	D	101	65	e
36	24	$	69	45	E	102	66	f
37	25	%	70	46	F	103	67	g
38	26	&	71	47	G	104	68	h
39	27	'	72	48	H	105	69	i
40	28	(	73	49	I	106	6A	j
41	29	)	74	4A	J	107	6B	k
42	2A	*	75	4B	K	108	6C	l
43	2B	+	76	4C	L	109	6D	m
44	2C	,	77	4D	M	110	6E	n
45	2D	-	78	4E	N	111	6F	o
46	2E	.	79	4F	O	112	70	p
47	2F	/	80	50	P	113	71	q
48	30	0	81	51	Q	114	72	r
49	31	1	82	52	R	115	73	s
50	32	2	83	53	S	116	74	t
51	33	3	84	54	T	117	75	u
52	34	4	85	55	U	118	76	v
53	35	5	86	56	V	119	77	w
54	36	6	87	57	W	120	78	x
55	37	7	88	58	X	121	79	y
56	38	8	89	59	Y	122	7A	z
57	39	9	90	5A	Z	123	7B	{
58	3A	:	91	5B	[	124	7C	|
59	3B	;	92	5C	\	125	7D	}
60	3C	<	93	5D	]	126	7E	~
61	3D	=	94	5E	^	127	7F	^
62	3E	>	95	5F	_	128	80	Ç
63	3F	?	96	60	`	129	81	ü
64	40	@	97	61	a	130	82	é

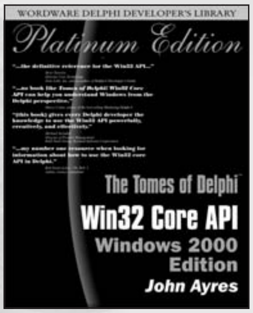
Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
131	83	â	170	AA	¬	209	D1	╤
132	84	ä	171	AB	½	210	D2	╥
133	85	à	172	AC	¼	211	D3	╙
134	86	å	173	AD	¡	212	D4	╘
135	87	ç	174	AE	«	213	D5	╒
136	88	ê	175	AF	»	214	D6	╓
137	89	ë	176	B0	░	215	D7	╫
138	8A	è	177	B1	▒	216	D8	╪
139	8B	ï	178	B2	▓	217	D9	┘
140	8C	î	179	B3	│	218	DA	┌
141	8D	ì	180	B4	┤	219	DB	█
142	8E	Ä	181	B5	╡	220	DC	▄
143	8F	Å	182	B6	╢	221	DD	▌
144	90	É	183	B7	╖	222	DE	▐
145	91	æ	184	B8	╕	223	DF	▀
146	92	Æ	185	B9	╣	224	E0	α
147	93	ô	186	BA	║	225	E1	ß
148	94	ö	187	BB	╗	226	E2	Γ
149	95	ò	188	BC	╝	227	E3	π
150	96	û	189	BD	╜	228	E4	Σ
151	97	ù	190	BE	╛	229	E5	σ
152	98	ÿ	191	BF	┐	230	E6	µ
153	99	Ö	192	C0	└	231	E7	τ
154	9A	Ü	193	C1	┴	232	E8	Φ
155	9B	¢	194	C2	┬	233	E9	Θ
156	9C	£	195	C3	├	234	EA	Ω
157	9D	¥	196	C4	─	235	EB	δ
158	9E	₧	197	C5	┼	236	EC	∞
159	9F	ƒ	198	C6	╞	237	ED	φ
160	A0	á	199	C7	╟	238	EE	ε
161	A1	í	200	C8	╚	239	EF	∩
162	A2	ó	201	C9	╔	240	F0	≡
163	A3	ú	202	CA	╩	241	F1	±
164	A4	ñ	203	CB	╦	242	F2	≥
165	A5	Ñ	204	CC	╠	243	F3	≤
166	A6	ª	205	CD	═	244	F4	⌠
167	A7	º	206	CE	╬	245	F5	⌡
168	A8	¿	207	CF	╧	246	F6	÷
169	A9	⌐	208	D0	╨	247	F7	≈

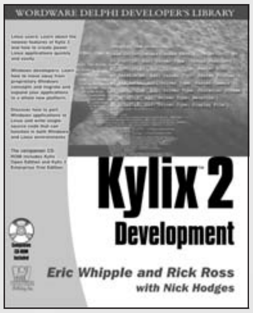
Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
248	F8	°	251	FB	√	254	FE	■
249	F9	·	252	FC	ⁿ	255	FF	
250	FA	·	253	FD	²			

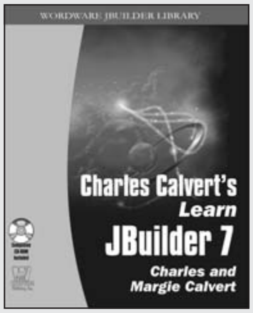
Index

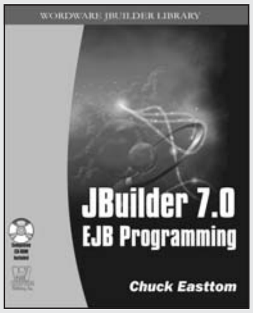


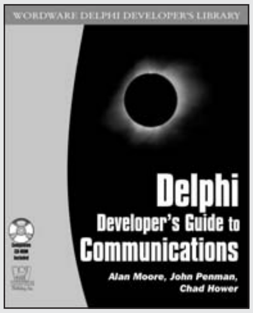












About the CD

The companion CD-ROM contains the code and compiled executables for every example in the book. The files are organized by chapter and listing, and are accessible using Windows Explorer.

There is also a comprehensive Windows help file covering every function explained in the book.

Warning By opening the CD package, you accept the terms and conditions of the CD/Source Code Usage License Agreement on the following page.

Opening the CD pac age ma es this boo nonreturnable