Menus and Other Resources

Article • 08/19/2020

A *resource* is binary data that you can add to the executable file of a Windows-based application. A resource can be either standard or defined. The data in a *standard resource* describes an icon, cursor, menu, dialog box, bitmap, enhanced metafile, font, accelerator table, message-table entry, string-table entry, or version information. An *application-defined resource*, also called a *custom resource*, contains any data required by a specific application.

In This Section

Name	Description
Introduction to Resources	Provides an overview of resources.
Carets	Discusses carets, which are blinking lines, blocks, or bitmaps in the client area of a window.
Cursors	Discusses cursors, which are small pictures whose location on the screen is controlled by a pointing device, such as a mouse, pen, or trackball.
Icons	Discusses icons, which are bitmap images combined with a mask to create transparent areas in the picture.
Keyboard Accelerators	Discusses keyboard accelerators, which are keystrokes that provide access to the commands for an application.
Menus	Discusses menus.
Strings	Discusses the string functions.
Version Information	Discusses the version-information resource.
Resource Compiler	Discusses the resource compiler, Rc.exe, and resource-definition files.
Package resource indexing (PRI) reference	A set of APIs for working with a resource indexer. A resource indexer is used to generate package resource index (PRI) files for a UWP app.

For more information about creating message resources, see **Message Compiler**.

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Introduction to Resources

Article • 04/27/2021

In This Section

- Resource Overviews
- Resource Reference

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Resource Overviews

Article • 04/27/2021

In This Section

- Adding, Deleting, and Replacing Resources
- Enumerating Resources
- Finding and Loading Resources
- Resource File Formats
- Using Resources

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Adding, Deleting, and Replacing Resources

Article • 08/23/2019

Applications must frequently add, delete, or replace resources in executable files. Two methods can be used to accomplish these tasks. The first is to edit the resource-definition file, recompile the resources, and add the recompiled resources to the application's executable file. The second method is to copy the resource data directly into the application's executable file.

For example, to localize an English-language application for use in Norway, it may be necessary to replace the English dialog box with one using Norwegian. A developer creates an appropriate dialog box by using a dialog box editor or by writing a template in the resource-definition file. The developer then recompiles the resources and adds the new resources to the application's executable file.

If an appropriate dialog box exists in binary form, however, the developer can copy the data directly to the executable file being localized by using the following functions. The **BeginUpdateResource** function creates an update handle to the executable file whose resources are to be changed. The **UpdateResource** function uses this handle to add, delete, or replace a resource in the executable file. The **EndUpdateResource** function closes the handle.

After an update handle to an executable file is created by **BeginUpdateResource**, an application can use **UpdateResource** repeatedly to make changes to the resource data. Each call to **UpdateResource** contributes to an internal list of additions, deletions, and replacements but does not actually write the data to the executable file. Immediately before closing the update handle, **EndUpdateResource** writes the accumulated changes to the executable file.

Sometimes, an application must copy resources from another file. Updating Resources shows an example of obtaining the resource data and its size from a file.

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Enumerating resources

Article • 06/02/2022

In certain situations, you might want to discover the resource contents of an unknown portable executable (PE) module. The Windows SDK provides resource enumeration functions that enable your application to obtain lists of resource types, names, and languages in a specified module.

- The **EnumResourceTypeW** and **EnumResourceTypesExW** functions provide a list of resource types found in the module.
- The **EnumResourceNamesW** and **EnumResourceNamesExW** functions provide the name of each resource within a given type.
- The **EnumResourceLanguagesW** and **EnumResourceLanguagesExW** functions provide the language of each resource of a given name and type.

These functions and their associated callback functions enable your application to create a list of all resources in a module. This process is described in Creating a resource list.

See also

Msistuff.exe sample app

Feedback

Finding and Loading Resources

Article • 08/19/2020

Before using a resource, an application must load it into memory. The FindResource and FindResourceEx functions find a resource in a module and return a handle to the binary resource data. **FindResource** locates a resource by type and name. **FindResourceEx** locates the resource by type, name, and language. Information about **FindResource** in this topic also applies to **FindResourceEx**.

The LoadResource function uses the resource handle returned by FindResource to load the resource into memory. After an application loads a resource by using **LoadResource**, the system will unload the associated memory only when all references to its module are freed through FreeLibrary. Applications which need to repeatedly access the same or many resources within a particular module may incur performance penalties due to the memory mapping taking place in repeated LoadLibrary and **FreeLibrary** calls. Applications should store a single module handle until resources are no longer needed, and then call **FreeLibrary**. After a module is unloaded from memory, resource handles become invalid.

An application can use FindResource and LoadResource to find and load any type of resource, but these functions should be used only in one of these situations:

- When the application cannot access the resource by using an existing resource-specific function.
- When the application must access the resource as binary data for subsequent function calls.

Whenever possible, an application should instead use one of the following resource-specific functions to find and load resources in one call:

Function	Action	To remove resource
FormatMessage	Loads and formats a message-table entry.	No action needed.
LoadAccelerators	Loads an accelerator table.	DestroyAcceleratorTable
LoadBitmap	Loads a bitmap resource.	DeleteObject
LoadCursor	Loads a cursor resource.	DestroyCursor
LoadIcon	Loads an icon resource.	DestroyIcon

Function	Action	To remove resource
LoadImage	Loads an icon, cursor, or bitmap.	DestroyIcon, DestroyCursor, DeleteObject
LoadMenu	Loads a menu resource.	DestroyMenu
LoadString	Loads a string-table entry.	No action needed.

Note the release functions in the table above. Before terminating, an application should release the memory occupied by accelerator tables, bitmaps, cursors, icons, and menus by using the appropriate functions.

Memory associated with resources loaded through **FindResource** and **LoadResource** will be released once the module has been unloaded by a call to **FreeLibrary**. Any resources which remain unloaded at application termination will be automatically released by the system.

Resource File Formats

Article • 06/17/2023

This section describes the format of the binary resource file that the resource compiler creates based on the contents of the resource-definition file. This file usually has an .res extension. The linker reformats the .res file into a resource object file and then links it to the executable file of an application.

A binary resource file consists of a number of concatenated resource entries. Each entry consists of a resource header and the data for that resource. A resource header is **DWORD**-aligned in the file and consists of the following:

- A **DWORD** that contains the size of the resource header
- A **DWORD** that contains the size of the resource data
- The resource type
- The resource name
- Additional resource information

The **RESOURCEHEADER** structure describes the format of this header. The data for the resource follows the resource header and is specific to each type of resource. Some resources also employ a resource-specific group header structure to provide information about a group of resources.

Accelerator Table Resources

An accelerator table is one resource entry in a resource file. It does not have a group header. An **ACCELTABLEENTRY** structure describes each entry in the accelerator table. Multiple accelerator tables are permitted.

Cursor and Icon Resources

The system handles each icon and cursor as a single file. However, these are stored in .res files and in executable files as a group of RT_GROUP_ICON icon resources or a RT_GROUP_CURSOR group of cursor resources. The file formats of icon and cursor resources are similar. In the .res file a resource group header follows all of the individual icon or cursor group components.

The group header for both icon and cursor resources consists of a **NEWHEADER** structure plus one or more **RESDIR** structures. There is one **RESDIR** structure for each icon or cursor. The group header contains the information an application needs to select

the correct icon or cursor to display. Both the group header and the data that repeats for each icon or cursor in the group have a fixed length. This allows the application to randomly access the information.

The format of each RT_ICON icon or RT_CURSOR cursor resource component closely resembles the format of the .ico/.cur file. Each image is stored in a **BITMAPINFO** structure followed by the color device-independent bitmap (DIB) bits of the icon's **XOR** mask. The monochrome DIB bits of the **AND** mask follow the color DIB bits. Significant difference between cursors and icons is that cursors have a **LOCALHEADER** structure with a hotspot inserted before the bitmap data, while icons do not.

Since Windows Vista RT_ICON icon or RT_CURSOR cursor resource may contain PNG-compressed image data.

Dialog Box Resources

A dialog box is also one resource entry in the resource file. It consists of one **DLGTEMPLATE** dialog box header structure plus one **DLGITEMTEMPLATE** structure for each control in the dialog box. The **DLGTEMPLATEEX** and the **DLGITEMTEMPLATEEX** structures describe the format of extended dialog box resources.

Font Resources

Fonts are stored in the resource file as a group of resources. Individual fonts make up a font group. A FONT Statement resource definition statement in the .RC file defines each font. Each individual font in the resource consists of the complete contents of the related .fnt file. A **FONTGROUPHDR** structure follows all the individual font components in the .res file.

Font resources are not added to the resources of a specific application. Instead, they are normally added to executable files that have a .fon extension. These files are usually resource-only DLLs rather than applications.

Menu Resources

A *menu resource* consists of a **MENUHEADER** structure followed by one or more **NORMALMENUITEM** or **POPUPMENUITEM** structures, one for each menu item in the menu template. The **MENUEX_TEMPLATE_HEADER** and the **MENUEX_TEMPLATE_ITEM** structures describe the format of extended menu resources.

Message Table Resources

A *message table* is a resource that contains formatted text for display as an error message or in a message box. The main structure in a message table resource is the **MESSAGE_RESOURCE_DATA** structure.

Version Resources

The main structure in a version resource is the **VS_FIXEDFILEINFO** structure. Additional structures include the **VarFileInfo** structure to store language information data, and **StringFileInfo** for user-defined string information. All strings in a version resource are in Unicode format. Each block of information is aligned on a **DWORD** boundary.

Using Resources

Article • 06/04/2021

This section contains code snippets for the following tasks:

- Updating Resources
- Creating a Resource List

Updating Resources

The following example copies a dialog box resource from one executable file, Hand.exe, to another, Foot.exe, by following these steps:

1. Use the **LoadLibrary** function to load the executable file Hand.exe.
2. Use the **FindResource** and **LoadResource** functions to locate and load the dialog box resource.
3. Use the **LockResource** function to retrieve a pointer to the dialog box resource data.
4. Use the **BeginUpdateResource** function to open an update handle to Foot.exe.
5. Use the **UpdateResource** function to copy the dialog box resource from Hand.exe to Foot.exe.
6. Use the **EndUpdateResource** function to complete the update.

The following code implements these steps.

Security Warning: Using **LoadLibrary** incorrectly can compromise the security of your application by loading the wrong DLL. Refer to the **LoadLibrary** documentation for information on how to correctly load DLLs with different versions of Windows.

C++

```
HGLOBAL hResLoad;   // handle to loaded resource
HMODULE hExe;       // handle to existing .EXE file
HRSRC hRes;         // handle/ptr. to res. info. in hExe
HANDLE hUpdateRes;  // update resource handle
LPVOID lpResLock;   // pointer to resource data
BOOL result;
#define IDD_HAND_ABOUTBOX   103
#define IDD_FOOT_ABOUTBOX   110

// Load the .EXE file that contains the dialog box you want to copy.
hExe = LoadLibrary(TEXT("hand.exe"));
if (hExe == NULL)
{
    ErrorHandler(TEXT("Could not load exe."));
```

```
    return;
}

// Locate the dialog box resource in the .EXE file.
hRes = FindResource(hExe, MAKEINTRESOURCE(IDD_HAND_ABOUTBOX), RT_DIALOG);
if (hRes == NULL)
{
    ErrorHandler(TEXT("Could not locate dialog box."));
    return;
}

// Load the dialog box into global memory.
hResLoad = LoadResource(hExe, hRes);
if (hResLoad == NULL)
{
    ErrorHandler(TEXT("Could not load dialog box."));
    return;
}

// Lock the dialog box into global memory.
lpResLock = LockResource(hResLoad);
if (lpResLock == NULL)
{
    ErrorHandler(TEXT("Could not lock dialog box."));
    return;
}

// Open the file to which you want to add the dialog box resource.
hUpdateRes = BeginUpdateResource(TEXT("foot.exe"), FALSE);
if (hUpdateRes == NULL)
{
    ErrorHandler(TEXT("Could not open file for writing."));
    return;
}

// Add the dialog box resource to the update list.
result = UpdateResource(hUpdateRes,    // update resource handle
    RT_DIALOG,                         // change dialog box resource
    MAKEINTRESOURCE(IDD_FOOT_ABOUTBOX),         // dialog box id
    MAKELANGID(LANG_NEUTRAL, SUBLANG_NEUTRAL),  // neutral language
    lpResLock,                         // ptr to resource info
    SizeofResource(hExe, hRes));       // size of resource info

if (result == FALSE)
{
    ErrorHandler(TEXT("Could not add resource."));
    return;
}

// Write changes to FOOT.EXE and then close it.
if (!EndUpdateResource(hUpdateRes, FALSE))
{
    ErrorHandler(TEXT("Could not write changes to file."));
    return;
}
```

```
// Clean up.
if (!FreeLibrary(hExe))
{
    ErrorHandler(TEXT("Could not free executable."));
    return;
}
```

Creating a Resource List

The following example creates a list of every resource in the Hand.exe file. The list is written to the Resinfo.txt file.

The code demonstrates how to load the executable file, create a file in which to write resource information, and call the **EnumResourceTypes** function to send each resource type found in the module to the application-defined callback function `EnumTypesFunc`. See *EnumResTypeProc* for information on callback functions of this type. This callback function uses the **EnumResourceNames** function to pass the name of every resource within the specified type to another application-defined callback function, `EnumNamesFunc`. See *EnumResNameProc* for information on callback functions of this type. `EnumNamesFunc` uses the **EnumResourceLanguages** function to pass the language of every resource of the specified type and name to a third callback function, `EnumLangsFunc`. See *EnumResLangProc* for information on callback functions of this type. `EnumLangsFunc` writes information about the resource of the specified type, name, and language to the Resinfo.txt file.

Note that the *lpszType* in *EnumResTypeProc* is either a resource ID or a pointer to a string (containing a resource ID or type name); *lpszType* and *lpszName* in *EnumResNameProc* and *EnumResLangProc* are similar. To load an enumerated resource, just call the appropriate function. For example, if a menu resource (**RT_MENU**) was enumerated, then pass *lpszName* to **LoadMenu**. For custom resources, pass *lpszType* and *lpszName* to **FindResource**.

The Updating Resources code follows a similar pattern for a dialog box resource.

Security Warning: Using **LoadLibrary** incorrectly can compromise the security of your application by loading the wrong DLL. Refer to the **LoadLibrary** documentation for information on how to correctly load DLLs with different versions of Windows.

```
C++
```

```
HANDLE g_hFile;   // global handle to resource info file
// Declare callback functions.
BOOL EnumTypesFunc(
```

```
        HMODULE hModule,
        LPTSTR lpType,
        LONG lParam);

BOOL EnumNamesFunc(
        HMODULE hModule,
        LPCTSTR lpType,
        LPTSTR lpName,
        LONG lParam);

BOOL EnumLangsFunc(
        HMODULE hModule,
        LPCTSTR lpType,
        LPCTSTR lpName,
        WORD wLang,
        LONG lParam);
```

C++

```
HMODULE hExe;        // handle to .EXE file
TCHAR szBuffer[80];  // print buffer for info file
DWORD cbWritten;     // number of bytes written to resource info file
size_t cbString;     // length of string in szBuffer
HRESULT hResult;

// Load the .EXE whose resources you want to list.
hExe = LoadLibrary(TEXT("hand.exe"));
if (hExe == NULL)
{
    // Add code to fail as securely as possible.
    return;
}

// Create a file to contain the resource info.
g_hFile = CreateFile(TEXT("resinfo.txt"),   // name of file
    GENERIC_READ | GENERIC_WRITE,      // access mode
    0,                                 // share mode
    (LPSECURITY_ATTRIBUTES) NULL,      // default security
    CREATE_ALWAYS,                     // create flags
    FILE_ATTRIBUTE_NORMAL,             // file attributes
    (HANDLE) NULL);                    // no template
if (g_hFile == INVALID_HANDLE_VALUE)
{
    ErrorHandler(TEXT("Could not open file."));
    return;
}

// Find all of the loaded file's resources.
hResult = StringCchPrintf(szBuffer, sizeof(szBuffer)/sizeof(TCHAR),
    TEXT("The file contains the following resources:\r\n\r\n"));
if (FAILED(hResult))
{
    // Add code to fail as securely as possible.
```

```
    return;
}

hResult = StringCchLength(szBuffer, sizeof(szBuffer)/sizeof(TCHAR),
&cbString);
if (FAILED(hResult))
{
    // Add code to fail as securely as possible.
    return;
}

WriteFile(g_hFile,       // file to hold resource info
    szBuffer,            // what to write to the file
    (DWORD) cbString,    // number of bytes in szBuffer
    &cbWritten,          // number of bytes written
    NULL);               // no overlapped I/O

EnumResourceTypes(hExe,              // module handle
    (ENUMRESTYPEPROC)EnumTypesFunc,  // callback function
    0);                              // extra parameter

// Unload the executable file whose resources were
// enumerated and close the file created to contain
// the resource information.
FreeLibrary(hExe);
CloseHandle(g_hFile);
```

C++

```
//    FUNCTION: EnumTypesFunc(HANDLE, LPSTR, LONG)
//
//    PURPOSE:  Resource type callback
BOOL EnumTypesFunc(
        HMODULE hModule,  // module handle
        LPTSTR lpType,    // address of resource type
        LONG lParam)      // extra parameter, could be
                          // used for error checking
{
    TCHAR szBuffer[80];  // print buffer for info file
    DWORD cbWritten;     // number of bytes written to resource info file
    size_t cbString;
    HRESULT hResult;

    // Write the resource type to a resource information file.
    // The type may be a string or an unsigned decimal
    // integer, so test before printing.
    if (!IS_INTRESOURCE(lpType))
    {
        hResult = StringCchPrintf(szBuffer, sizeof(szBuffer)/sizeof(TCHAR),
TEXT("Type: %s\r\n"), lpType);
        if (FAILED(hResult))
        {
            // Add code to fail as securely as possible.
```

```
            return FALSE;
        }
    }
    else
    {
        hResult = StringCchPrintf(szBuffer, sizeof(szBuffer)/sizeof(TCHAR),
TEXT("Type: %u\r\n"), (USHORT)lpType);
        if (FAILED(hResult))
        {
            // Add code to fail as securely as possible.
            return FALSE;
        }
    }

    hResult = StringCchLength(szBuffer, sizeof(szBuffer)/sizeof(TCHAR),
&cbString);
    if (FAILED(hResult))
    {
        // Add code to fail as securely as possible.
        return FALSE;
    }

    WriteFile(g_hFile, szBuffer, (DWORD) cbString, &cbWritten, NULL);
    // Find the names of all resources of type lpType.
    EnumResourceNames(hModule,
        lpType,
        (ENUMRESNAMEPROC)EnumNamesFunc,
        0);

    return TRUE;
}

//    FUNCTION: EnumNamesFunc(HANDLE, LPSTR, LPSTR, LONG)
//
//    PURPOSE:  Resource name callback
BOOL EnumNamesFunc(
        HMODULE hModule,  // module handle
        LPCTSTR lpType,   // address of resource type
        LPTSTR lpName,    // address of resource name
        LONG lParam)      // extra parameter, could be
                          // used for error checking
{
    TCHAR szBuffer[80];  // print buffer for info file
    DWORD cbWritten;     // number of bytes written to resource info file
    size_t cbString;
    HRESULT hResult;

    // Write the resource name to a resource information file.
    // The name may be a string or an unsigned decimal
    // integer, so test before printing.
    if (!IS_INTRESOURCE(lpName))
    {
        hResult = StringCchPrintf(szBuffer, sizeof(szBuffer)/sizeof(TCHAR),
TEXT("\tName: %s\r\n"), lpName);
        if (FAILED(hResult))
```

```
        {
            // Add code to fail as securely as possible.
            return FALSE;
        }
    }
    else
    {
        hResult = StringCchPrintf(szBuffer, sizeof(szBuffer)/sizeof(TCHAR),
TEXT("\tName: %u\r\n"), (USHORT)lpName);
        if (FAILED(hResult))
        {
            // Add code to fail as securely as possible.
            return FALSE;
        }
    }

    hResult = StringCchLength(szBuffer, sizeof(szBuffer)/sizeof(TCHAR),
&cbString);
    if (FAILED(hResult))
    {
        // Add code to fail as securely as possible.
        return FALSE;
    }

    WriteFile(g_hFile, szBuffer, (DWORD) cbString, &cbWritten, NULL);
    // Find the languages of all resources of type
    // lpType and name lpName.
    EnumResourceLanguages(hModule,
        lpType,
        lpName,
        (ENUMRESLANGPROC)EnumLangsFunc,
        0);

    return TRUE;
}

//    FUNCTION: EnumLangsFunc(HANDLE, LPSTR, LPSTR, WORD, LONG)
//
//    PURPOSE:  Resource language callback
BOOL EnumLangsFunc(
        HMODULE hModule, // module handle
        LPCTSTR lpType,  // address of resource type
        LPCTSTR lpName,  // address of resource name
        WORD wLang,      // resource language
        LONG lParam)     // extra parameter, could be
                         // used for error checking
{
    HRSRC hResInfo;
    TCHAR szBuffer[80];  // print buffer for info file
    DWORD cbWritten;     // number of bytes written to resource info file
    size_t cbString;
    HRESULT hResult;

    hResInfo = FindResourceEx(hModule, lpType, lpName, wLang);
    // Write the resource language to the resource information file.
```

```
    hResult = StringCchPrintf(szBuffer, sizeof(szBuffer)/sizeof(TCHAR),
TEXT("\t\tLanguage: %u\r\n"), (USHORT) wLang);
    if (FAILED(hResult))
    {
        // Add code to fail as securely as possible.
        return FALSE;
    }

    hResult = StringCchLength(szBuffer, sizeof(szBuffer)/sizeof(TCHAR),
&cbString);
    if (FAILED(hResult))
    {
        // Add code to fail as securely as possible.
        return FALSE;
    }

    WriteFile(g_hFile, szBuffer, (DWORD) cbString, &cbWritten, NULL);
    // Write the resource handle and size to buffer.
    hResult = StringCchPrintf(szBuffer,
        sizeof(szBuffer)/sizeof(TCHAR),
        TEXT("\t\thResInfo == %lx,  Size == %lu\r\n\r\n"),
        hResInfo,
        SizeofResource(hModule, hResInfo));
    if (FAILED(hResult))
    {
        // Add code to fail as securely as possible.
        return FALSE;
    }

    hResult = StringCchLength(szBuffer, sizeof(szBuffer)/sizeof(TCHAR),
&cbString);
    if (FAILED(hResult))
    {
        // Add code to fail as securely as possible.
        return FALSE;
    }

    WriteFile(g_hFile, szBuffer, (DWORD)cbString, &cbWritten, NULL);
    return TRUE;
}
```

Feedback

Resource Reference (Menus and Other Resources)

Article • 04/27/2021

In This Section

- Resource Functions
- Resource Macros
- Resource Structures
- **Resource Types**

Related topics

Resource-Definition Statements

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Resource Functions (Menus and Other Resources)

Article • 06/04/2021

In This Section

- BeginUpdateResource
- CopyImage
- EndUpdateResource
- *EnumResLangProc*
- *EnumResNameProc*
- EnumResourceLanguages
- EnumResourceLanguagesEx
- EnumResourceNames
- EnumResourceNamesEx
- EnumResourceTypes
- EnumResourceTypesEx
- *EnumResTypeProc*
- FindResource
- FindResourceEx
- FreeResource
- LoadImage
- LoadResource
- LockResource
- SizeofResource
- UpdateResource

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

BeginUpdateResourceA function (winbase.h)

Article02/09/2023

Retrieves a handle that can be used by the UpdateResource function to add, delete, or replace resources in a binary module.

Syntax

```
C++

HANDLE BeginUpdateResourceA(
  [in] LPCSTR pFileName,
  [in] BOOL   bDeleteExistingResources
);
```

Parameters

`[in] pFileName`

Type: **LPCTSTR**

The binary file in which to update resources. An application must be able to obtain write-access to this file; the file referenced by *pFileName* cannot be currently executing. If *pFileName* does not specify a full path, the system searches for the file in the current directory.

`[in] bDeleteExistingResources`

Type: **BOOL**

Indicates whether to delete the *pFileName* parameter's existing resources. If this parameter is **TRUE**, existing resources are deleted and the updated file includes only resources added with the UpdateResource function. If this parameter is **FALSE**, the updated file includes existing resources unless they are explicitly deleted or replaced by using **UpdateResource**.

Return value

Type: **HANDLE**

If the function succeeds, the return value is a handle that can be used by the UpdateResource and EndUpdateResource functions. The return value is **NULL** if the specified file is not a PE, the file does not exist, or the file cannot be opened for writing. To get extended error information, call GetLastError.

Remarks

It is recommended that the resource file is not loaded before this function is called. However, if that file is already loaded, it will not cause an error to be returned.

There are some restrictions on resource updates in files that contain Resource Configuration(RC Config) data: LN files and the associated .mui files. Details on which types of resources are allowed to be updated in these files are in the Remarks section for the UpdateResource function.

This function can update resources within modules that contain both code and resources. As noted above, there are restrictions on resource updates in LN files and .mui files, both of which contain RC Config data; details of the restrictions are in the reference for the UpdateResource function.

Examples

For an example see, Updating Resources.

> **Note**
>
> The winbase.h header defines BeginUpdateResource as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Conceptual

EndUpdateResource

Reference

Resources

UpdateResource

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CreateResourceIndexer function (resourceindexer.h)

Article10/13/2021

Creates a new resource indexer for the specified paths of the root of the project files and the extension DLL.

Syntax

```
C++

HRESULT CreateResourceIndexer(
  [in]           PCWSTR projectRoot,
  [in, optional] PCWSTR extensionDllPath,
  [out]          PVOID  *ppResourceIndexer
);
```

Parameters

`[in] projectRoot`

The path of the root folder to use for the project for the files to be produced, in string form. This path is used to determine file paths relative to the package that contains them. This path must be an absolute path with the drive letter specified. Long file paths are not supported.

`[in, optional] extensionDllPath`

The full path to an extension dynamic-link library (DLL) that is Microsoft-signed and implements the ext-ms-win-mrmcorer-environment-l1 API set. This path determines the file path from where the extension DLL for the modern resource technology (MRT) environment is loaded. This path must be an absolute path with the drive letter specified. Long file paths are not supported.

`[out] ppResourceIndexer`

The newly created resource indexer.

Return value

If this function succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	resourceindexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

DestroyResourceIndexer

Feedback

CopyImage function (winuser.h)

Article03/11/2023

Creates a new image (icon, cursor, or bitmap) and copies the attributes of the specified image to the new one. If necessary, the function stretches the bits to fit the desired size of the new image.

Syntax

C++

```
HANDLE CopyImage(
  [in] HANDLE h,
  [in] UINT   type,
  [in] int    cx,
  [in] int    cy,
  [in] UINT   flags
);
```

Parameters

`[in] h`

Type: **HANDLE**

A handle to the image to be copied.

`[in] type`

Type: **UINT**

The type of image to be copied. This parameter can be one of the following values.

Expand table

Value	Meaning
IMAGE_BITMAP 0	Copies a bitmap.
IMAGE_CURSOR 2	Copies a cursor.
IMAGE_ICON	Copies an icon.

1

`[in] cx`

Type: **int**

The desired width, in pixels, of the image. If this is zero, then the returned image will have the same width as the original *hImage*.

`[in] cy`

Type: **int**

The desired height, in pixels, of the image. If this is zero, then the returned image will have the same height as the original *hImage*.

`[in] flags`

Type: **UINT**

This parameter can be one or more of the following values.

Expand table

Value	Meaning
LR_COPYDELETEORG 0x00000008	Deletes the original image after creating the copy.
LR_COPYFROMRESOURCE 0x00004000	Tries to reload an icon or cursor resource from the original resource file rather than simply copying the current image. This is useful for creating a different-sized copy when the resource file contains multiple sizes of the resource. Without this flag, **CopyImage** stretches the original image to the new size. If this flag is set, **CopyImage** uses the size in the resource file closest to the desired size. This will succeed only if *hImage* was loaded by LoadIcon or LoadCursor, or by LoadImage with the LR_SHARED flag.
LR_COPYRETURNORG 0x00000004	Returns the original *hImage* if it satisfies the criteria for the copy—that is, correct dimensions and color depth—in which case the **LR_COPYDELETEORG** flag is ignored. If this flag is not specified, a new object is always created.
LR_CREATEDIBSECTION 0x00002000	If this is set and a new bitmap is created, the bitmap is created as a DIB section. Otherwise, the bitmap image is

	created as a device-dependent bitmap. This flag is only valid if *uType* is **IMAGE_BITMAP**.
LR_DEFAULTCOLOR 0x00000000	Uses the default color format.
LR_DEFAULTSIZE 0x00000040	Uses the width or height specified by the system metric values for cursors or icons, if the *cxDesired* or *cyDesired* values are set to zero. If this flag is not specified and *cxDesired* and *cyDesired* are set to zero, the function uses the actual resource size. If the resource contains multiple images, the function uses the size of the first image.
LR_MONOCHROME 0x00000001	Creates a new monochrome image.

Return value

Type: **HANDLE**

If the function succeeds, the return value is the handle to the newly created image.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

When you are finished using the resource, you can release its associated memory by calling one of the functions in the following table.

Expand table

Resource	Release function
Bitmap	DeleteObject
Cursor	DestroyCursor
Icon	DestroyIcon

The system automatically deletes the resource when its process terminates, however, calling the appropriate function saves memory and decreases the size of the process's working set.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-gui-l1-3-0 (introduced in Windows 10, version 10.0.10240)

See also

Conceptual

LoadImage

Reference

Resources

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

DestroyIndexedResults function (resourceindexer.h)

Article02/22/2024

Frees the parameters that the IndexFilePath method returned.

Syntax

C++

```
void DestroyIndexedResults(
  [in, optional] PWSTR                    resourceUri,
  [in]           ULONG                    qualifierCount,
  [in, optional] IndexedResourceQualifier *qualifiers
);
```

Parameters

`[in, optional] resourceUri`

A uniform resource indicator (URI) that uses the ms-resource URI scheme and represents the named resource for the candidate, where the authority of the URI or the resource map is empty. For example, ms-resource:///Resources/String1 or ms-resource:///Files/images/logo.png.

`[in] qualifierCount`

The number of indexed resource qualifiers that the list in the *ppQualifiers* parameter contains.

`[in, optional] qualifiers`

A list of indexed resource qualifiers that declare the context under which the resources are appropriate.

Return value

None

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	resourceindexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

IndexFilePath

IndexedResourceQualifier

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

DestroyResourceIndexer function (resourceindexer.h)

Article02/22/2024

Frees the computational resources associated with the specified resource indexer.

Syntax

```
C++
```

```
void DestroyResourceIndexer(
  [in, optional] PVOID resourceIndexer
);
```

Parameters

`[in, optional] resourceIndexer`

The resource indexer for which you want to free the computational resources.

Return value

None

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	resourceindexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

CreateResourceIndexer

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

EndUpdateResourceA function (winbase.h)

Article02/22/2024

Commits or discards changes made prior to a call to UpdateResource.

Syntax

```
C++

BOOL EndUpdateResourceA(
  [in] HANDLE hUpdate,
  [in] BOOL   fDiscard
);
```

Parameters

`[in] hUpdate`

Type: **HANDLE**

A module handle returned by the BeginUpdateResource function, and used by UpdateResource, referencing the file to be updated.

`[in] fDiscard`

Type: **BOOL**

Indicates whether to write the resource updates to the file. If this parameter is **TRUE**, no changes are made. If it is **FALSE**, the changes are made: the resource updates will take effect.

Return value

Type: **BOOL**

Returns **TRUE** if the function succeeds; **FALSE** otherwise. If the function succeeds and *fDiscard* is **TRUE**, then no resource updates are made to the file; otherwise all successful resource updates are made to the file. To get extended error information, call GetLastError.

Remarks

Before you call this function, make sure all file handles other than the one returned by BeginUpdateResource are closed.

This function can update resources within modules that contain both code and resources. There are restrictions on resource updates in LN files and .mui files, both of which contain Resource Configuration data; details of the restrictions are in the reference for the UpdateResource function.

Examples

For an example, see Updating Resources.

> **Note**
>
> The winbase.h header defines EndUpdateResource as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

BeginUpdateResource

Conceptual

Reference

Resources

UpdateResource

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

ENUMRESNAMEPROCW callback function (libloaderapi.h)

Article08/23/2022

An application-defined callback function used with the EnumResourceNames and EnumResourceNamesEx functions. It receives the type and name of a resource. The **ENUMRESNAMEPROC** type defines a pointer to this callback function. *EnumResNameProc* is a placeholder for the application-defined function name.

Syntax

C++

```
ENUMRESNAMEPROCW Enumresnameprocw;

BOOL Enumresnameprocw(
  [in, optional] HMODULE hModule,
                 LPCWSTR lpType,
                 LPWSTR lpName,
  [in]           LONG_PTR lParam
)
{...}
```

Parameters

`[in, optional] hModule`

Type: **HMODULE**

A handle to the module whose executable file contains the resources that are being enumerated. If this parameter is **NULL**, the function enumerates the resource names in the module used to create the current process.

`lpType`

Type: **LPCTSTR**

The type of resource for which the name is being enumerated. Alternately, rather than a pointer, this parameter can be `MAKEINTRESOURCE(ID)`, where ID is an integer value representing a predefined resource type. For standard resource types, see Resource Types. For more information, see the Remarks section below.

`lpName`

Type: **LPTSTR**

The name of a resource of the type being enumerated. Alternately, rather than a pointer, this parameter can be `MAKEINTRESOURCE(ID)`, where ID is the integer identifier of the resource. For more information, see the Remarks section below.

`[in] lParam`

Type: **LONG_PTR**

An application-defined parameter passed to the EnumResourceNames or EnumResourceNamesEx function. This parameter can be used in error checking.

Return value

Type: **BOOL**

Returns **TRUE** to continue enumeration or **FALSE** to stop enumeration.

Remarks

If IS_INTRESOURCE(*lpszType*) is **TRUE**, then *lpszType* specifies the integer identifier of the given resource type. Otherwise, it is a pointer to a null-terminated string. If the first character of the string is a pound sign (#), then the remaining characters represent a decimal number that specifies the integer identifier of the resource type. For example, the string "#258" represents the identifier 258.

Similarly, if IS_INTRESOURCE(*lpszName*) is **TRUE**, then *lpszName* specifies the integer identifier of the given resource. Otherwise, it is a pointer to a null-terminated string. If the first character of the string is a pound sign (#), then the remaining characters represent a decimal number that specifies the integer identifier of the resource.

An application must register this function by passing its address to the EnumResourceNames or EnumResourceNamesEx function.

If the callback function returns **FALSE**, then EnumResourceNames or EnumResourceNamesEx will stop enumeration and return **FALSE**. On Windows XP and earlier the value obtained from GetLastError will be **ERROR_SUCCESS**; starting with Windows Vista, the last error value will be **ERROR_RESOURCE_ENUM_USER_STOP**.

> ⓘ **Note**

The libloaderapi.h header defines ENUMRESNAMEPROC as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	libloaderapi.h (include Windows.h)

See also

Conceptual

EnumResourceNames

EnumResourceNamesEx

IS_INTRESOURCE

Reference

Resources

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

EnumResourceLanguagesA function (winbase.h)

Article02/09/2023

Enumerates language-specific resources, of the specified type and name, associated with a binary module.

Syntax

```
C++

BOOL EnumResourceLanguagesA(
  [in] HMODULE          hModule,
  [in] LPCSTR           lpType,
  [in] LPCSTR           lpName,
  [in] ENUMRESLANGPROCA lpEnumFunc,
  [in] LONG_PTR         lParam
);
```

Parameters

`[in] hModule`

Type: **HMODULE**

The handle to a module to be searched. Starting with Windows Vista, if this is a language-neutral Portable Executable (LN file), then appropriate .mui files (if any exist) are included in the search. If this is a specific .mui file, only that file is searched for resources.

If this parameter is **NULL**, that is equivalent to passing in a handle to the module used to create the current process.

`[in] lpType`

Type: **LPCTSTR**

The type of resource for which the language is being enumerated. Alternately, rather than a pointer, this parameter can be MAKEINTRESOURCE(ID), where ID is an integer value representing a predefined resource type. For a list of predefined resource types, see Resource Types. For more information, see the Remarks section below.

`[in] lpName`

Type: **LPCTSTR**

The name of the resource for which the language is being enumerated. Alternately, rather than a pointer, this parameter can be MAKEINTRESOURCE(ID), where ID is the integer identifier of the resource. For more information, see the Remarks section below.

`[in] lpEnumFunc`

Type: **ENUMRESLANGPROC**

A pointer to the callback function to be called for each enumerated resource language. For more information, see EnumResLangProcA.

`[in] lParam`

Type: **LONG_PTR**

An application-defined value passed to the callback function. This parameter can be used in error checking.

Return value

Type: **BOOL**

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call GetLastError.

Remarks

If IS_INTRESOURCE(*lpType*) is **TRUE**, then *lpType* specifies the integer identifier of the given resource type. Otherwise, it is a pointer to a null-terminated string. If the first character of the string is a pound sign (#), then the remaining characters represent a decimal number that specifies the integer identifier of the resource type. For example, the string "#258" represents the identifier 258.

Similarly, if IS_INTRESOURCE(*lpName*) is **TRUE**, then *lpName* specifies the integer identifier of the given resource. Otherwise, it is a pointer to a null-terminated string. If the first character of the string is a pound sign (#), then the remaining characters represent a decimal number that specifies the integer identifier of the resource.

Starting with Windows Vista, the binary module is typically a language-neutral Portable Executable (LN file), and the enumeration will also include resources from the

corresponding language-specific resource files (.mui files) that contain localizable language resources.

For each resource found, **EnumResourceLanguages** calls an application-defined callback function *lpEnumFunc*, passing the language identifier (see Language Identifiers) of the language for which a resource was found, as well as the various other parameters that were passed to **EnumResourceLanguages**.

Alternately, applications can call EnumResourceLanguagesEx, which provides more precise control of what resources are enumerated.

The **EnumResourceLanguages** function continues to enumerate resource languages until the callback function returns **FALSE** or all resource languages have been enumerated.

In Windows Vista and later, if *hModule* specifies an LN file, then the resources enumerated can reside either in the LN file or in an .mui file associated with it. If no .mui files are found, only resources from the LN file are returned. Unlike EnumResourceNames and EnumResourceTypes, this search will look at multiple .mui files. The enumeration begins with .mui files in the folders associated with EnumUILanguages. These are followed by any other .mui files whose paths conform to the scheme described at MUI Resource Management. Finally, the file designated by *hModule* is also searched.

The enumeration never includes duplicates: if a resource with the same name, type, and language is contained in both the LN file and in an .mui file, the resource will only be enumerated once.

Examples

For an example, see Creating a Resource List.

> **Note**
>
> The winbase.h header defines EnumResourceLanguages as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Conceptual

EnumResLangProc

EnumResourceLanguagesEx

EnumResourceNames

EnumResourceTypes

Reference

Resources

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

EnumResourceLanguagesExW function (libloaderapi.h)

Article05/17/2023

Enumerates language-specific resources, of the specified type and name, associated with a specified binary module. Extends EnumResourceLanguages by allowing more control over the enumeration.

Syntax

C++

```
BOOL EnumResourceLanguagesExW(
  [in] HMODULE          hModule,
  [in] LPCWSTR          lpType,
  [in] LPCWSTR          lpName,
  [in] ENUMRESLANGPROCW lpEnumFunc,
  [in] LONG_PTR         lParam,
  [in] DWORD            dwFlags,
  [in] LANGID           LangId
);
```

Parameters

`[in] hModule`

Type: **HMODULE**

The handle to a module to search. Typically this is a language-neutral Portable Executable (LN file), and if flag **RESOURCE_ENUM_MUI** is set, then appropriate .mui files are included in the search. Alternately, this can be a handle to an .mui file or other LN file. If this is a specific .mui file, only that file is searched for resources.

If this parameter is **NULL**, it is equivalent to passing in a handle to the module used to create the current process.

`[in] lpType`

Type: **LPCTSTR**

The type of the resource for which the language is being enumerated. Alternately, rather than a pointer, this parameter can be MAKEINTRESOURCE(ID), where ID is an integer

value representing a predefined resource type. For a list of predefined resource types, see Resource Types. For more

information, see the Remarks section below.

`[in] lpName`

Type: **LPCTSTR**

The name of the resource for which the language is being enumerated. Alternately, rather than a pointer, this parameter can be MAKEINTRESOURCE(ID), where ID is the integer identifier of the resource. For more information, see the Remarks section below.

`[in] lpEnumFunc`

Type: **ENUMRESLANGPROC**

A pointer to the callback function to be called for each enumerated resource language. For more information, see EnumResLangProcW.

`[in] lParam`

Type: **LONG_PTR**

An application-defined value passed to the callback function. This parameter can be used in error checking.

`[in] dwFlags`

Type: **DWORD**

The type of file to be searched. The following values are supported. Note that if *dwFlags* is zero, then the **RESOURCE_ENUM_LN** and **RESOURCE_ENUM_MUI** flags are assumed to be specified.

Expand table

Value	Meaning
RESOURCE_ENUM_MUI 0x0002	Search for language-specific resources in .mui files associated with the LN file specified by *hModule*. Alternately, if *LangId* is nonzero, the only .mui file searched will be the one matching the specified *LangId*. Typically this flag should be used only if *hModule* references an LN file. If *hModule* references an .mui file, then that file is actually covered by the **RESOURCE_LN**

	flag, despite the name of the flag. See the Remarks section below for sequence of search.
RESOURCE_ENUM_LN 0x0001	Searches the file specified by *hModule*, regardless of whether the file is an LN file, another type of LN file, or an .mui file.
RESOURCE_ENUM_MUI_SYSTEM 0x0004	Restricts the .mui files search to system-installed MUI languages.
RESOURCE_ENUM_VALIDATE 0x0008	Performs extra validation on the resource section and its reference in the PE header while doing the enumeration to ensure that resources are properly formatted.

`[in] LangId`

Type: **LANGID**

The localization language used to filter the search in the .mui file. This parameter is used only when the **RESOURCE_ENUM_MUI** flag is set in *dwFlags*. If zero is specified, then all .mui files are included in the search. If a nonzero *LangId* is specified, then the only .mui file searched will be the one matching the specified *LangId*.

Return value

Type: **BOOL**

Returns **TRUE** if the function succeeds or **FALSE** if the function does not find a resource of the type specified, or if the function fails for another reason. To get extended error information, call GetLastError.

Remarks

If IS_INTRESOURCE(*lpType*) is **TRUE**, then *lpType* specifies the integer identifier of the given resource type. Otherwise, it is a pointer to a null-terminated string. If the first character of the string is a pound sign (#), then the remaining characters represent a decimal number that specifies the

integer identifier of the resource type. For example, the string "#258" represents the identifier 258.

Similarly, if IS_INTRESOURCE(*lpName*) is **TRUE**, then *lpName* specifies the integer identifier of the given resource. Otherwise, it is a pointer to a null-terminated string. If

the first character of the string is a pound sign (#), then the remaining characters represent a decimal number that specifies the

integer identifier of the resource.

Starting with Windows Vista, the binary module is typically an LN file, and the enumeration will also include resources from the corresponding language-specific resource files (.mui files) that contain localizable language resources.

For each such resource found, **EnumResourceLanguagesEx** calls an application-defined callback function *lpEnumFunc*, passing to the callback function the language identifier (see Language Identifiers) of the language for which a resource was found (as well as the various other parameters that were passed to **EnumResourceLanguagesEx**).

The search can include both an LN file and its associated .mui files, or it can be limited either to a single binary module of any type, or to the .mui files associated with a single LN file. Also, by specifying an LN file for the *hModule* parameter and a nonzero *LangId* parameter, the search can be limited to the unique .mui file associated with that LN file and language.

The **EnumResourceLanguagesEx** function continues to enumerate resource languages until the callback function returns **FALSE** or all resource languages have been enumerated.

If *hModule* specifies an LN file, and both flags are selected, the languages enumerated include all languages whose resources reside either in the LN file or in any .mui files associated with it. If no .mui files are found, only languages from the LN file are returned.

If *dwFlags* contains **RESOURCE_ENUM_MUI** or **NULL** and *LangId* is 0, then the enumeration first includes the languages associated with all system-installed .mui files, using languages retrieved from EnumUILanguages.. Finally, if the **RESOURCE_ENUM_LN** flag is also set, the file designated by *hModule* is also searched.

If the *LangId* is nonzero, then only the .mui file corresponding to that language identifier will be searched. Language fallbacks will not be used. If an .mui file for that language does not exist, the enumeration will be empty (unless resources for that language exist in the LN file, and the flag is set to search the LN file as well).

The enumeration never includes duplicates: if resources for a particular language are contained in both the LN file and in an .mui file, the type will only be enumerated once.

Examples

For an example, see Creating a Resource List.

> **Note**
>
> The libloaderapi.h header defines EnumResourceLanguagesEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	libloaderapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Conceptual

EnumResLangProcW

EnumResourceNamesEx

EnumResourceTypesEx

MAKEINTRESOURCE

Reference

Resources

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

EnumResourceNamesA function (libloaderapi.h)

Article07/27/2022

Enumerates resources of a specified type within a binary module. For Windows Vista and later, this is typically a language-neutral Portable Executable (LN file), and the enumeration will also include resources from the corresponding language-specific resource files (.mui files) that contain localizable language resources. It is also possible for *hModule* to specify an .mui file, in which case only that file is searched for resources.

Syntax

```
C++

BOOL EnumResourceNamesA(
  [in, optional] HMODULE          hModule,
  [in]           LPCSTR           lpType,
  [in]           ENUMRESNAMEPROCA lpEnumFunc,
  [in]           LONG_PTR         lParam
);
```

Parameters

`[in, optional] hModule`

Type: **HMODULE**

A handle to a module to be searched. Starting with Windows Vista, if this is an LN file, then appropriate .mui files (if any exist) are included in the search.

If this parameter is **NULL**, that is equivalent to passing in a handle to the module used to create the current process.

`[in] lpType`

Type: **LPCTSTR**

The type of the resource for which the name is being enumerated. Alternately, rather than a pointer, this parameter can be MAKEINTRESOURCE(ID), where ID is an integer value representing a predefined resource type. For a list of predefined resource types, see Resource Types. For more information, see the Remarks section below.

`[in] lpEnumFunc`

Type: **ENUMRESNAMEPROC**

A pointer to the callback function to be called for each enumerated resource name or ID. For more information, see ENUMRESNAMEPROC.

`[in] lParam`

Type: **LONG_PTR**

An application-defined value passed to the callback function. This parameter can be used in error checking.

Return value

Type: **BOOL**

The return value is **TRUE** if the function succeeds or **FALSE** if the function does not find a resource of the type specified, or if the function fails for another reason. To get extended error information, call GetLastError.

Remarks

If IS_INTRESOURCE(*lpszType*) is **TRUE**, then *lpszType* specifies the integer identifier of the given resource type. Otherwise, it is a pointer to a null-terminated string. If the first character of the string is a pound sign (#), then the remaining characters represent a decimal number that specifies the integer identifier of the resource type. For example, the string "#258" represents the identifier 258.

For each resource found, **EnumResourceNames** calls an application-defined callback function *lpEnumFunc*, passing the name or the ID of each resource it finds, as well as the various other parameters that were passed to **EnumResourceNames**.

Alternately, applications can call EnumResourceNamesEx, which provides more precise control of what resources are enumerated.

If a resource has an ID, the ID is passed to the callback function; otherwise the resource name is passed to the callback function. For more information, see ENUMRESNAMEPROC.

The **EnumResourceNames** function continues to enumerate resources until the callback function returns **FALSE** or all resources have been enumerated.

Starting with Windows Vista, if *hModule* specifies an LN file, then the resources enumerated can reside either in the LN file or in a .mui file associated with it. If no .mui files are found, only resources from the LN file are returned. The order in which .mui files are searched is the usual Resource Loader search order; see User Interface Language Management for details. Once one appropriate .mui file is found, the .mui file search stops. Because all .mui files that correspond to a single LN file have the same resource types, only the resources in the found .mui file need to be enumerated.

The enumeration never includes duplicates: if resources with the same name are contained in both the LN file and in an .mui file, the resource will only be enumerated once.

Examples

For an example, see Creating a Resource List.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	libloaderapi.h
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Conceptual

- ENUMRESNAMEPROC
- EnumResourceLanguages
- EnumResourceNamesEx
- EnumResourceTypes

Reference

- Menus and Other Resources

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

EnumResourceNamesExW function (libloaderapi.h)

Article02/09/2023

Enumerates resources of a specified type that are associated with a specified binary module. The search can include both an LN file and its associated .mui files, or it can be limited in several ways.

Syntax

```
C++

BOOL EnumResourceNamesExW(
  [in, optional] HMODULE          hModule,
                 LPCWSTR          lpType,
  [in]           ENUMRESNAMEPROCW lpEnumFunc,
  [in]           LONG_PTR         lParam,
  [in]           DWORD            dwFlags,
  [in]           LANGID           LangId
);
```

Parameters

`[in, optional] hModule`

Type: **HMODULE**

The handle to a module to search. Typically this is an LN file, and if flag **RESOURCE_ENUM_MUI** is set, then appropriate .mui files are included in the search. Alternately, this can be a handle to an .mui file or other LN file.

If this parameter is **NULL**, it is equivalent to passing in a handle to the module used to create the current process.

`lpType`

Type: **LPCTSTR**

The type of the resource for which the name is being enumerated. Alternately, rather than a pointer, this parameter can be MAKEINTRESOURCE(ID), where ID is an integer value representing a predefined resource type. For a list of predefined resource types, see Resource Types. For more information, see the Remarks section below.

`[in] lpEnumFunc`

Type: **ENUMRESNAMEPROC**

A pointer to the callback function to be called for each enumerated resource name. For more information, see EnumResNameProc.

`[in] lParam`

Type: **LONG_PTR**

An application-defined value passed to the callback function. This parameter can be used in error checking.

`[in] dwFlags`

Type: **DWORD**

The type of file to search. The following values are supported. Note that if *dwFlags* is zero, then the **RESOURCE_ENUM_LN** and **RESOURCE_ENUM_MUI** flags are assumed to be specified.

Expand table

Value	Meaning
RESOURCE_ENUM_MUI 0x0002	Search for resources in .mui files associated with the LN file specified by *hModule* and with the current language preferences, following the usual Resource Loader strategy (see User Interface Language Management). Alternately, if *LangId* is nonzero, then only the specified .mui file will be searched. Typically this flag should be used only if *hModule* references an LN file. If *hModule* references an .mui file, then that file is actually covered by the **RESOURCE_ENUM_LN** flag, despite the name of the flag.
RESOURCE_ENUM_LN 0x0001	Searches the file specified by *hModule*, regardless of whether the file is an LN file, another type of LN file, or an .mui file.
RESOURCE_ENUM_VALIDATE 0x0008	Performs extra validation on the resource section and its reference in the PE header while doing the enumeration to ensure that resources are properly formatted. The validation sets a maximum limit of 260 characters for each name that is enumerated.

`[in] LangId`

Type: **LANGID**

The localization language used to filter the search in the MUI module. This parameter is used only when the **RESOURCE_ENUM_MUI** flag is set in *dwFlags*. If zero is specified, then all .mui files that match current language preferences are included in the search, following the usual Resource Loader strategy (see User Interface Language Management). If a nonzero *LangId* is specified, then the only .mui file searched will be the one matching the specified *LangId*.

Return value

Type: **BOOL**

The function **TRUE** if successful, or **FALSE** if the function does not find a resource of the type specified, or if the function fails for another reason. To get extended error information, call GetLastError.

Remarks

If IS_INTRESOURCE(*lpszType*) is **TRUE**, then *lpszType* specifies the integer identifier of the given resource type. Otherwise, it is a pointer to a null-terminated string. If the first character of the string is a pound sign (#), then the remaining characters represent a decimal number that specifies the

integer identifier of the resource type. For example, the string "#258" represents the identifier 258.

The enumeration search can include both an LN file and its associated .mui files. It can be limited to a single binary module of any type. It can also be limited to the .mui files associated with a single LN file. By specifying an LN file for the *hModule* parameter and a nonzero *LangId* parameter, the search can be limited to the unique .mui file associated with that LN file and language.

For each resource found, **EnumResourceNamesEx** calls an application-defined callback function *lpEnumFunc*, passing the name or the ID of each resource it finds, as well as the various other parameters that were passed to **EnumResourceNamesEx**.

If a resource has an ID, the ID is returned to the callback function; otherwise the resource name is returned to the callback function. For more information, see EnumResNameProc.

The **EnumResourceNamesEx** function continues to enumerate resource names until the callback function returns **FALSE** or all resource names for this type have been enumerated.

If *hModule* specifies an LN file, and both flags are selected, the names enumerated correspond to resources residing either in that LN file or the .mui files associated with it. If no .mui files are found, only names from the LN file are returned. After one appropriate .mui file is found the search will not continue further, because all .mui files corresponding to a single LN file have the same resource names.

If *dwFlags* and *LangId* are both zero, then the function behaves like EnumResourceNames.

If *LangId* is nonzero, then only the .mui file corresponding to that Language identifier will be searched. Language fallbacks will not be used. If an .mui file for that language does not exist, the enumeration will be empty (unless resources for that language exist in the LN file, and the flag is set to search the LN file as well).

The enumeration never includes duplicates: if resources for a particular language are contained in both the LN file and in an .mui file, the name will only be enumerated once.

Examples

For an example, see Creating a Resource List.

> **Note**
>
> The libloaderapi.h header defines EnumResourceNamesEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	libloaderapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Conceptual

EnumResNameProc

EnumResourceLanguagesEx

EnumResourceNames

EnumResourceTypesEx

Reference

Resources

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

EnumResourceTypesA function (winbase.h)

Article02/09/2023

Enumerates resource types within a binary module. Starting with Windows Vista, this is typically a language-neutral Portable Executable (LN file), and the enumeration also includes resources from one of the corresponding language-specific resource files (.mui files)—if one exists—that contain localizable language resources. It is also possible to use *hModule* to specify a .mui file, in which case only that file is searched for resource types.

Alternately, applications can call EnumResourceTypesEx, which provides more precise control over which resource files to enumerate.

Syntax

```
C++

BOOL EnumResourceTypesA(
  [in, optional] HMODULE          hModule,
  [in]           ENUMRESTYPEPROCA lpEnumFunc,
  [in]           LONG_PTR         lParam
);
```

Parameters

`[in, optional] hModule`

Type: **HMODULE**

A handle to a module to be searched. This handle must be obtained through LoadLibrary or LoadLibraryEx.

See Remarks for more information.

If this parameter is **NULL**, that is equivalent to passing in a handle to the module used to create the current process.

`[in] lpEnumFunc`

Type: **ENUMRESTYPEPROC**

A pointer to the callback function to be called for each enumerated resource type. For more information, see the EnumResTypeProc function.

`[in] lParam`

Type: **LONG_PTR**

An application-defined value passed to the callback function.

Return value

Type: **BOOL**

Returns **TRUE** if successful; otherwise, **FALSE**. To get extended error information, call GetLastError.

Remarks

For each resource type found, **EnumResourceTypes** calls an application-defined callback function *lpEnumFunc*, passing each resource type it finds, as well as the various other parameters that were passed to **EnumResourceTypes**.

EnumResourceTypes continues to enumerate resource types until the callback function returns **FALSE** or all resource types have been enumerated.

Starting with Windows Vista, if *hModule* specifies an LN file, then the types enumerated correspond to resources that reside in the LN file and in the .mui file associated with it. If no .mui files are found, only types from the LN file are returned. The order in which .mui files are searched is the usual Resource Loader search order; see User Interface Language Management for details. After one appropriate .mui file is found, the search does not continue further to other .mui files associated with the LN file, because all .mui files that correspond to a single LN file have the same set of resource types.

The enumeration never includes duplicates: if a given resource type is contained in both the LN file and in an .mui file, the type is enumerated only once.

Examples

For an example, see Creating a Resource List.

> ⓘ Note

The winbase.h header defines EnumResourceTypes as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Conceptual

EnumResTypeProc

EnumResourceLanguages

EnumResourceNames

EnumResourceTypesEx

Reference

Resources

Feedback

EnumResourceTypesExW function (libloaderapi.h)

Article02/09/2023

Enumerates resource types associated with a specified binary module. The search can include both a language-neutral Portable Executable file (LN file) and its associated .mui files. Alternately, it can be limited to a single binary module of any type, or to the .mui files associated with a single LN file. The search can also be limited to a single associated .mui file that contains resources for a specific language.

For each resource type found, **EnumResourceTypesEx** calls an application-defined callback function *lpEnumFunc*, passing the resource type it finds, as well as the various other parameters that were passed to **EnumResourceTypesEx**.

Syntax

C++

```
BOOL EnumResourceTypesExW(
  [in, optional] HMODULE          hModule,
  [in]           ENUMRESTYPEPROCW lpEnumFunc,
  [in]           LONG_PTR         lParam,
  [in]           DWORD            dwFlags,
  [in]           LANGID           LangId
);
```

Parameters

`[in, optional] hModule`

Type: **HMODULE**

The handle to a module to be searched. Typically this is an LN file, and if flag **RESOURCE_ENUM_MUI** is set, then appropriate .mui files can be included in the search. Alternately, this can be a handle to an .mui file or other LN file.

If this parameter is **NULL**, it is equivalent to passing in a handle to the module used to create the current process.

`[in] lpEnumFunc`

Type: **ENUMRESTYPEPROC**

A pointer to the callback function to be called for each enumerated resource type. For more information, see EnumResTypeProc.

`[in] lParam`

Type: **LONG_PTR**

An application-defined value passed to the callback function.

`[in] dwFlags`

Type: **DWORD**

The type of file to be searched. The following values are supported. Note that if *dwFlags* is zero, then the **RESOURCE_ENUM_LN** and **RESOURCE_ENUM_MUI** flags are assumed to be specified.

Expand table

Value	Meaning
RESOURCE_ENUM_MUI 0x0002	Search for resource types in one of the .mui files associated with the file specified by *hModule* and with the current language preferences, following the usual Resource Loader strategy (see User Interface Language Management). Alternately, if *LangId* is nonzero, then only the .mui file of the language as specified by *LangId* will be searched. Typically this flag should be used only if *hModule* references an LN file. If *hModule* references an .mui file, then that file is actually covered by the **RESOURCE_ENUM_LN** flag, despite the name of the flag.
RESOURCE_ENUM_LN 0x0001	Searches only the file specified by *hModule*, regardless of whether the file is an LN file or an .mui file.
RESOURCE_ENUM_VALIDATE 0x0008	Performs extra validation on the resource section and its reference in the PE header while doing the enumeration to ensure that resources are properly formatted. The validation sets a maximum limit of 260 characters for each type that is enumerated.

`[in] LangId`

Type: **LANGID**

The language used to filter the search in the MUI module. This parameter is used only when the **RESOURCE_ENUM_MUI** flag is set in *dwFlags*. If zero is specified, then all .mui files that match current language preferences are included in the search, following the usual Resource Loader strategy (see User Interface Language Management). If a nonzero *LangId* is specified, then the only .mui file searched will be the one matching the specified *LangId*.

Return value

Type: **BOOL**

Returns **TRUE** if successful or **FALSE** if the function does not find a resource of the type specified, or if the function fails for another reason. To get extended error information, call GetLastError.

Remarks

The **EnumResourceTypesEx** function continues to enumerate resource types until the callback function returns **FALSE** or all resource types have been enumerated.

If *hModule* specifies an LN file, and both flags are selected, the types enumerated correspond to resources residing either in the LN file or in the .mui files associated with it. If no .mui files are found, only types from the LN file are returned. Once one appropriate .mui file is found the search will not continue further, because all .mui files corresponding to a single LN file have the same resource types.

If *dwFlags* and *LangId* are both zero, then the function behaves like EnumResourceTypes.

If *LangId* is nonzero, then only the .mui file corresponding to that language identifier will be searched. Language fallbacks will not be used. If an .mui file for that language does not exist, the enumeration will be empty (unless resources for that language exist in the LN file, and the flag is set to search the LN file as well).

The enumeration never includes duplicates: if resources for a particular language are contained in both the LN file and in an .mui file, the type will only be enumerated once.

Examples

For an example, see Creating a Resource List.

> ⓘ **Note**

The libloaderapi.h header defines EnumResourceTypesEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	libloaderapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Conceptual

EnumResTypeProc

EnumResourceLanguagesEx

EnumResourceNamesEx

EnumResourceTypes

Reference

Resources

Feedback

ENUMRESTYPEPROCW callback function (libloaderapi.h)

Article07/27/2022

An application-defined callback function used with the EnumResourceTypes and EnumResourceTypesEx functions. It receives resource types. The **ENUMRESTYPEPROC** type defines a pointer to this callback function. *EnumResTypeProc* is a placeholder for the application-defined function name.

Syntax

```
C++

ENUMRESTYPEPROCW Enumrestypeprocw;

BOOL Enumrestypeprocw(
  [in, optional] HMODULE hModule,
                 LPWSTR lpType,
  [in]           LONG_PTR lParam
)
{...}
```

Parameters

`[in, optional] hModule`

Type: **HMODULE**

A handle to the module whose executable file contains the resources for which the types are to be enumerated. If this parameter is **NULL**, the function enumerates the resource types in the module used to create the current process.

`lpType`

Type: **LPTSTR**

The type of resource for which the type is being enumerated.

Alternately, rather than a pointer, this parameter can be MAKEINTRESOURCE(ID), where ID is the integer identifier of the given resource type. For standard resource types, see Resource Types. For more information, see the Remarks section below.

`[in] lParam`

Type: **LONG_PTR**

An application-defined parameter passed to the EnumResourceTypes or EnumResourceTypesEx function. This parameter can be used in error checking.

Return value

Type: **BOOL**

Returns **TRUE** to continue enumeration or **FALSE** to stop enumeration.

Remarks

If IS_INTRESOURCE(*lpszType*) is **TRUE**, then *lpszType* specifies the integer identifier of the given resource type. Otherwise, it is a pointer to a null-terminated string. If the first character of the string is a pound sign (#), then the remaining characters represent a decimal number that specifies the integer identifier of the resource type. For example, the string "#258" represents the identifier 258.

An application must register this function by passing its address to the EnumResourceTypes or EnumResourceTypesEx function.

If the callback function returns **FALSE**, then EnumResourceTypes or EnumResourceTypesEx will stop enumeration and return **FALSE**. On Windows XP and earlier the value obtained from GetLastError will be **ERROR_SUCCESS**; starting with Windows Vista, the last error value will be **ERROR_RESOURCE_ENUM_USER_STOP**.

> ⓘ **Note**
>
> The libloaderapi.h header defines ENUMRESTYPEPROC as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	libloaderapi.h (include Windows.h)

See also

Conceptual

EnumResourceTypes

EnumResourceTypesEx

IS_INTRESOURCE

Reference

Resources

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

FindResourceA function (winbase.h)

Article07/27/2022

Determines the location of a resource with the specified type and name in the specified module.

To specify a language, use the FindResourceEx function.

Syntax

```
C++

HRSRC FindResourceA(
  [in, optional] HMODULE hModule,
  [in]           LPCSTR  lpName,
  [in]           LPCSTR  lpType
);
```

Parameters

`[in, optional] hModule`

Type: **HMODULE**

A handle to the module whose portable executable file or an accompanying MUI file contains the resource. If this parameter is **NULL**, the function searches the module used to create the current process.

`[in] lpName`

Type: **LPCTSTR**

The name of the resource. Alternately, rather than a pointer, this parameter can be MAKEINTRESOURCE(ID), where ID is the integer identifier of the resource. For more information, see the Remarks section below.

`[in] lpType`

Type: **LPCTSTR**

The resource type. Alternately, rather than a pointer, this parameter can be MAKEINTRESOURCE(ID), where ID is the integer identifier of the given resource type. For

standard resource types, see Resource Types. For more information, see the Remarks section below.

Return value

Type: **HRSRC**

If the function succeeds, the return value is a handle to the specified resource's information block. To obtain a handle to the resource, pass this handle to the LoadResource function.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

If IS_INTRESOURCE is **TRUE** for x = *lpName* or *lpType*, x specifies the integer identifier of the name or type of the given resource. Otherwise, those parameters are long pointers to null-terminated strings. If the first character of the string is a pound sign (#), the remaining characters represent a decimal number that specifies the integer identifier of the resource's name or type. For example, the string "#258" represents the integer identifier 258.

To reduce the amount of memory required for a resource, an application should refer to it by integer identifier instead of by name.

An application can use **FindResource** to find any type of resource, but this function should be used only if the application must access the binary resource data by making subsequent calls to LoadResource and then to LockResource.

To use a resource immediately, an application should use one of the following resource-specific functions to find the resource and convert the data into a more usable form.

Expand table

Function	Action
FormatMessage	Loads and formats a message-table entry.
LoadAccelerators	Loads an accelerator table.
LoadBitmap	Loads a bitmap resource.
LoadCursor	Loads a cursor resource.

LoadIcon	Loads an icon resource.
LoadMenu	Loads a menu resource.
LoadString	Loads a string-table entry.

For example, an application can use the LoadIcon function to load an icon for display on the screen. However, the application should use **FindResource** and LoadResource if it is loading the icon to copy its data to another application.

String resources are stored in sections of up to 16 strings per section. The strings in each section are stored as a sequence of counted (not necessarily null-terminated) Unicode strings. The LoadString function will extract the string resource from its corresponding section.

Examples

For an example, see Updating Resources.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Conceptual

FindResourceEx

FormatMessage

IS_INTRESOURCE

LoadAccelerators

LoadBitmap

LoadCursor

LoadIcon

LoadMenu

LoadResource

LoadString

LockResource

Other Resources

Reference

Resources

SizeofResource

Feedback

FindResourceExA function (winbase.h)

Article07/27/2022

Determines the location of the resource with the specified type, name, and language in the specified module.

Syntax

```
C++

HRSRC FindResourceExA(
  [in, optional] HMODULE hModule,
  [in]           LPCSTR  lpType,
  [in]           LPCSTR  lpName,
  [in]           WORD    wLanguage
);
```

Parameters

`[in, optional] hModule`

Type: **HMODULE**

A handle to the module whose portable executable file or an accompanying MUI file contains the resource. If this parameter is **NULL**, the function searches the module used to create the current process.

`[in] lpType`

Type: **LPCTSTR**

The resource type. Alternately, rather than a pointer, this parameter can be MAKEINTRESOURCE(ID), where ID is the integer identifier of the given

resource type. For standard resource types, see Resource Types. For more information, see the Remarks section below.

`[in] lpName`

Type: **LPCTSTR**

The name of the resource. Alternately, rather than a pointer, this parameter can be MAKEINTRESOURCE(ID), where ID is the integer identifier of the resource. For more

information, see the Remarks section below.

`[in] wLanguage`

Type: **WORD**

The language of the resource. If this parameter is `MAKELANGID(LANG_NEUTRAL, SUBLANG_NEUTRAL)`, the current language associated with the calling thread is used.

To specify a language other than the current language, use the MAKELANGID macro to create this parameter. For more information, see **MAKELANGID**.

Return value

Type: **HRSRC**

If the function succeeds, the return value is a handle to the specified resource's information block. To obtain a handle to the resource, pass this handle to the LoadResource function.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

If IS_INTRESOURCE is **TRUE** for x = *lpType* or *lpName*, x specifies the integer identifier of the type or name of the given resource. Otherwise, those parameters are long pointers to null-terminated strings. If the first character of the string is a pound sign (#), the remaining characters represent a decimal number that specifies the integer identifier of the resource's name or type. For example, the string "#258" represents the integer identifier 258.

To reduce the amount of memory required for a resource, an application should refer to it by integer identifier instead of by name.

An application can use **FindResourceEx** to find any type of resource, but this function should be used only if the application must access the binary resource data by making subsequent calls to LoadResource and then to LockResource.

To use a resource immediately, an application should use one of the following resource-specific functions to find the resource and convert the data into a more usable form.

Expand table

Function	Action
FormatMessage	Loads and formats a message-table entry.
LoadAccelerators	Loads an accelerator table.
LoadBitmap	Loads a bitmap resource.
LoadCursor	Loads a cursor resource.
LoadIcon	Loads an icon resource.
LoadMenu	Loads a menu resource.
LoadString	Loads a string-table entry.

For example, an application can use the LoadIcon function to load an icon for display on the screen. However, the application should use **FindResourceEx** and LoadResource if it is loading the icon to copy its data to another application.

String resources are stored in sections of up to 16 strings per section. The strings in each section are stored as a sequence of counted (not necessarily null-terminated) Unicode strings. The LoadString function will extract the string resource from its corresponding section.

Examples

For an example, see Creating a Resource List.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib

Requirement	Value
DLL	Kernel32.dll

See also

Conceptual

FindResource

FormatMessage

IS_INTRESOURCE

LoadAccelerators

LoadBitmap

LoadCursor

LoadIcon

LoadMenu

LoadResource

LoadString

MAKELANGID

Other Resources

Reference

Resources

Feedback

FreeResource function (libloaderapi.h)

Article02/22/2024

> **Note**
>
> This function is obsolete and is only supported for backward compatibility with 16-bit Windows. For 32-bit Windows applications, it is not necessary to free the resources loaded using **LoadResource**. For modern versions of Windows this function always returns **FALSE**.

Decrements (decreases by one) the reference count of a loaded resource. When the reference count reaches zero, the memory occupied by the resource is freed.

Syntax

```
C++

BOOL FreeResource(
  [in] HGLOBAL hResData
);
```

Parameters

`[in] hResData`

Type: **HGLOBAL**

A handle of the resource. It is assumed that *hglbResource* was created by LoadResource.

Return value

Type: **BOOL**

If the function succeeds, the return value is zero.

If the function fails, the return value is nonzero, which indicates that the resource has not been freed.

Remarks

For resources loaded with other functions, **FreeResource** has been replaced by the following functions:

Expand table

Resource type	FreeResource replacement
Accelerator	DestroyAcceleratorTable
Bitmap	DeleteObject
Cursor	DestroyCursor
Icon	DestroyIcon
Menu	DestroyMenu

The reference count for a resource is incremented (increased by one) each time an application calls the LoadResource function for the resource.

The system automatically deletes these resources when the process that created them terminates. However, calling the appropriate function saves memory. For more information, see LoadResource.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	libloaderapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

DeleteObject

DestroyAcceleratorTable

DestroyCursor

DestroyIcon

DestroyMenu

LoadResource

Other Resources

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

IndexFilePath function (resourceindexer.h)

Article10/13/2021

Indexes a file path for file and folder naming conventions.

Syntax

C++

```
HRESULT IndexFilePath(
  [in]  PVOID                     resourceIndexer,
  [in]  PCWSTR                    filePath,
  [out] PWSTR                     *ppResourceUri,
  [out] ULONG                     *pQualifierCount,
  [out] IndexedResourceQualifier **ppQualifiers
);
```

Parameters

`[in] resourceIndexer`

The resource indexer object that you created by calling the CreateResourceIndexer function.

`[in] filePath`

The path for the folder that you want to index. The path must be an absolute path with the drive letter specified. Long file paths are not supported.

`[out] ppResourceUri`

A uniform resource indicator (URI) that uses the ms-resource URI scheme and represents the named resource for the candidate, where the authority of the URI or the resource map is empty. For example, ms-resource:///Resources/String1 or ms-resource:///Files/images/logo.png.

`[out] pQualifierCount`

The number of indexed resource qualifiers that the list in the *ppQualifiers* parameter contains.

`[out] ppQualifiers`

A list of indexed resource qualifiers that declare the context under which the resources are appropriate.

Return value

If this function succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	resourceindexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

CreateResourceIndexer

IndexedResourceQualifier

Feedback

LoadImageA function (winuser.h)

Article07/20/2023

Loads an icon, cursor, animated cursor, or bitmap.

Syntax

C++

```
HANDLE LoadImageA(
  [in, optional] HINSTANCE hInst,
  [in]           LPCSTR    name,
  [in]           UINT      type,
  [in]           int       cx,
  [in]           int       cy,
  [in]           UINT      fuLoad
);
```

Parameters

`[in, optional] hInst`

Type: **HINSTANCE**

A handle to the module of either a DLL or executable (.exe) that contains the image to be loaded. For more information, see GetModuleHandle. Note that as of 32-bit Windows, an instance handle (**HINSTANCE**), such as the application instance handle exposed by system function call of WinMain, and a module handle (**HMODULE**) are the same thing.

To load a predefined image or a standalone resource (icon, cursor, or bitmap file), set this parameter to **NULL**.

`[in] name`

Type: **LPCTSTR**

The image to be loaded.

If the *hInst* parameter is non-**NULL** and the *fuLoad* parameter omits **LR_LOADFROMFILE**, *name* specifies the image resource in the *hInst* module.

If the image resource is to be loaded by name from the module, the *name* parameter is a pointer to a null-terminated string that contains the name of the image resource.

If the image resource is to be loaded by ordinal from the module, use the MAKEINTRESOURCE macro to convert the image ordinal into a form that can be passed to the **LoadImage** function.

If the *hInst* parameter is **NULL** and the *fuLoad* parameter omits the **LR_LOADFROMFILE** value and includes the **LR_SHARED**, the *name* specifies the predefined image to load.

The predefined image identifiers are defined in `Winuser.h` and have the following prefixes:

Expand table

Prefix	Meaning
OBM_	OEM bitmaps. Use the MAKEINTRESOURCE macro to pass these.
OIC_	OEM icons. Use the MAKEINTRESOURCE macro to pass these.
OCR_	OEM cursors. Use the MAKEINTRESOURCE macro to pass these.
IDI_	Standard icons
IDC_	Standard cursors

To pass OEM image identifiers constants to the **LoadImage** function, use the MAKEINTRESOURCE macro. For example, to load the **OCR_NORMAL** cursor, pass `MAKEINTRESOURCE(OCR_NORMAL)` as the *name* parameter, **NULL** as the *hInst* parameter, and **LR_SHARED** as one of the flags to the *fuLoad* parameter.

If the *hInst* parameter is **NULL** and the *fuLoad* parameter includes the **LR_LOADFROMFILE** value, *name* is the name of the file that contains the standalone resource (icon, cursor, or bitmap file), - for example, `c:\myicon.ico`.

For more information, see the Remarks section below.

`[in] type`

Type: **UINT**

The type of image to be loaded.

This parameter can be one of the following values:

Expand table

Value	Meaning
IMAGE_BITMAP	Loads a bitmap.
IMAGE_CURSOR	Loads a cursor.
IMAGE_ICON	Loads an icon.

`[in] cx`

Type: **int**

The width, in pixels, of the icon or cursor. If this parameter is zero and the *fuLoad* parameter is **LR_DEFAULTSIZE**, the function uses the **SM_CXICON** or **SM_CXCURSOR** system metric value to set the width. If this parameter is zero and **LR_DEFAULTSIZE** is not used, the function uses the actual resource width.

`[in] cy`

Type: **int**

The height, in pixels, of the icon or cursor. If this parameter is zero and the *fuLoad* parameter is **LR_DEFAULTSIZE**, the function uses the **SM_CYICON** or **SM_CYCURSOR** system metric value to set the height. If this parameter is zero and **LR_DEFAULTSIZE** is not used, the function uses the actual resource height.

`[in] fuLoad`

Type: **UINT**

This parameter can be one or more of the following values.

Expand table

Value	Meaning
LR_CREATEDIBSECTION 0x00002000	When the *uType* parameter specifies **IMAGE_BITMAP**, causes the function to return a DIB section bitmap rather than a compatible bitmap. This flag is useful for loading a bitmap without mapping it to the colors of the display device.
LR_DEFAULTCOLOR 0x00000000	The default flag; it does nothing. All it means is "not **LR_MONOCHROME**".

LR_DEFAULTSIZE 0x00000040	Uses the width or height specified by the system metric values for cursors or icons, if the *cxDesired* or *cyDesired* values are set to zero. If this flag is not specified and *cxDesired* and *cyDesired* are set to zero, the function uses the actual resource size. If the resource contains multiple images, the function uses the size of the first image.
LR_LOADFROMFILE 0x00000010	Loads the standalone image from the file specified by *name* (icon, cursor, or bitmap file).
LR_LOADMAP3DCOLORS 0x00001000	Searches the color table for the image and replaces the following shades of gray with the corresponding 3-D color. • Dk Gray, RGB(128,128,128) with **COLOR_3DSHADOW** • Gray, RGB(192,192,192) with **COLOR_3DFACE** • Lt Gray, RGB(223,223,223) with **COLOR_3DLIGHT** Do not use this option if you are loading a bitmap with a color depth greater than 8bpp.
LR_LOADTRANSPARENT 0x00000020	Retrieves the color value of the first pixel in the image and replaces the corresponding entry in the color table with the default window color (**COLOR_WINDOW**). All pixels in the image that use that entry become the default window color. This value applies only to images that have corresponding color tables. Do not use this option if you are loading a bitmap with a color depth greater than 8bpp. If *fuLoad* includes both the **LR_LOADTRANSPARENT** and **LR_LOADMAP3DCOLORS** values, **LR_LOADTRANSPARENT** takes precedence. However, the color table entry is replaced with **COLOR_3DFACE** rather than **COLOR_WINDOW**.
LR_MONOCHROME 0x00000001	Loads the image in black and white.
LR_SHARED 0x00008000	Shares the image handle if the image is loaded multiple times. If **LR_SHARED** is not set, a second call to **LoadImage** for the same resource will load the image again and return a different handle. When you use this flag, the system will destroy the resource when it is no longer needed. Do not use **LR_SHARED** for images that have non-standard sizes, that may change after loading, or that are loaded from a file.

	When loading a system icon or cursor, you must use **LR_SHARED** or the function will fail to load the resource. This function finds the first image in the cache with the requested resource name, regardless of the size requested.
LR_VGACOLOR 0x00000080	Uses true VGA colors.

Return value

Type: **HANDLE**

If the function succeeds, the return value is the handle of the newly loaded image.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

If IS_INTRESOURCE(*name*) is **TRUE**, then *name* specifies the integer identifier of the given resource. Otherwise, it is a pointer to a null-terminated string. If the first character of the string is a pound sign (#), then the remaining characters represent a decimal number that specifies the integer identifier of the resource. For example, the string "#258" represents the identifier 258.

When you are finished using a bitmap, cursor, or icon you loaded without specifying the **LR_SHARED** flag, you can release its associated memory by calling one of the functions in the following table.

Expand table

Resource	Release function
Bitmap	DeleteObject
Cursor	DestroyCursor
Icon	DestroyIcon

The system automatically deletes these resources when the process that created them terminates; however, calling the appropriate function saves memory and decreases the

size of the process's working set.

Examples

For an example, see Using Window Classes.

> **Note**
>
> The winuser.h header defines LoadImage as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-gui-l1-1-0 (introduced in Windows 8)

See also

Conceptual

CopyImage

GetSystemMetrics

LoadBitmap

LoadCursor

LoadIcon

Other Resources

Reference

Resources

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

LoadResource function (libloaderapi.h)

Article02/22/2024

Retrieves a handle that can be used to obtain a pointer to the first byte of the specified resource in memory.

Syntax

```
C++

HGLOBAL LoadResource(
  [in, optional] HMODULE hModule,
  [in]           HRSRC   hResInfo
);
```

Parameters

`[in, optional] hModule`

Type: **HMODULE**

A handle to the module whose executable file contains the resource. If *hModule* is **NULL**, the system loads the resource from the module that was used to create the current process.

`[in] hResInfo`

Type: **HRSRC**

A handle to the resource to be loaded. This handle is returned by the FindResource or FindResourceEx function.

Return value

Type: **HGLOBAL**

If the function succeeds, the return value is a handle to the data associated with the resource.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

The return type of **LoadResource** is **HGLOBAL** for backward compatibility, not because the function returns a handle to a global memory block. Do not pass this handle to the GlobalLock or GlobalFree function. To obtain a pointer to the first byte of the resource data, call the LockResource function; to obtain the size of the resource, call SizeofResource.

GlobalSize returns 0 for a resource HGLOBAL. As a result, any APIs that depend on GlobalSize to determine the size of the HGLOBAL will not function correctly. For example, use SHCreateMemStream instead of CreateStreamOnHGlobal.

To use a resource immediately, an application should use the following resource-specific functions to find and load the resource in one call.

Expand table

Function	Action	To remove resource
FormatMessage	Loads and formats a message-table entry	No action needed
LoadAccelerators	Loads an accelerator table	DestroyAcceleratorTable
LoadBitmap	Loads a bitmap resource	DeleteObject
LoadCursor	Loads a cursor resource	DestroyCursor
LoadIcon	Loads an icon resource	DestroyIcon
LoadMenu	Loads a menu resource	DestroyMenu
LoadString	Loads a string resource	No action needed

For example, an application can use the LoadIcon function to load an icon for display on the screen, followed by DestroyIcon when done.

Examples

For an example see Updating Resources.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	libloaderapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Conceptual

FindResource

FindResourceEx

LoadLibrary

LoadModule

LockResource

Other Resources

Reference

Resources

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

LockResource function (libloaderapi.h)

Article02/22/2024

Retrieves a pointer to the specified resource in memory.

Syntax

```
C++

LPVOID LockResource(
  [in] HGLOBAL hResData
);
```

Parameters

`[in] hResData`

Type: **HGLOBAL**

A handle to the resource to be accessed. The LoadResource function returns this handle. Note that this parameter is listed as an **HGLOBAL** variable only for backward compatibility. Do not pass any value as a parameter other than a successful return value from the **LoadResource** function.

Return value

Type: **LPVOID**

If the loaded resource is available, the return value is a pointer to the first byte of the resource; otherwise, it is **NULL**.

Remarks

The pointer returned by **LockResource** is valid until the module containing the resource is unloaded. It is not necessary to unlock resources because the system automatically deletes them when the process that created them terminates.

Do not try to lock a resource by using the handle returned by the FindResourceA function or FindResourceExA function function. Such a handle points to random data.

> **Note**
>
> **LockResource** does not actually lock memory; it is just used to obtain a pointer to the memory containing the resource data. The name of the function comes from versions prior to Windows XP, when it was used to lock a global memory block allocated by **LoadResource**.

Examples

For an example, see Updating Resources.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	libloaderapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Conceptual

- Resources

Reference

- Menus and Other Resources
- FindResourceA function = FindResourceExA function
- LoadResource function

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

SizeofResource function (libloaderapi.h)

Article02/22/2024

Retrieves the size, in bytes, of the specified resource.

Syntax

```
C++

DWORD SizeofResource(
  [in, optional] HMODULE hModule,
  [in]           HRSRC   hResInfo
);
```

Parameters

`[in, optional] hModule`

Type: **HMODULE**

A handle to the module whose executable file contains the resource. Default is the module used to create the current process.

`[in] hResInfo`

Type: **HRSRC**

A handle to the resource. This handle must be created by using the FindResource or FindResourceEx function.

Return value

Type: **DWORD**

If the function succeeds, the return value is the number of bytes in the resource.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	libloaderapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Conceptual

FindResource

FindResourceEx

Reference

Resources

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

UpdateResourceA function (winbase.h)

Article02/09/2023

Adds, deletes, or replaces a resource in a portable executable (PE) file. There are some restrictions on resource updates in files that contain Resource Configuration (RC Config) data: language-neutral (LN) files and language-specific resource (.mui) files.

Syntax

C++

```
BOOL UpdateResourceA(
  [in]           HANDLE hUpdate,
  [in]           LPCSTR lpType,
  [in]           LPCSTR lpName,
  [in]           WORD   wLanguage,
  [in, optional] LPVOID lpData,
  [in]           DWORD  cb
);
```

Parameters

`[in] hUpdate`

Type: **HANDLE**

A module handle returned by the BeginUpdateResource function, referencing the file to be updated.

`[in] lpType`

Type: **LPCTSTR**

The resource type to be updated. Alternatively, rather than a pointer, this parameter can be MAKEINTRESOURCE(ID), where ID is an integer value representing a predefined resource type. If the first character of the string is a pound sign (#), then the remaining characters represent a decimal number that specifies the integer identifier of the resource type. For example, the string "#258" represents the identifier 258.

For a list of predefined resource types, see Resource Types.

`[in] lpName`

Type: **LPCTSTR**

The name of the resource to be updated. Alternatively, rather than a pointer, this parameter can be MAKEINTRESOURCE(ID), where ID is a resource ID. When creating a new resource do not use a string that begins with a '#' character for this parameter.

`[in] wLanguage`

Type: **WORD**

The language identifier of the resource to be updated. For a list of the primary language identifiers and sublanguage identifiers that make up a language identifier, see the MAKELANGID macro.

`[in, optional] lpData`

Type: **LPVOID**

The resource data to be inserted into the file indicated by *hUpdate*. If the resource is one of the predefined types, the data must be valid and properly aligned. Note that this is the raw binary data to be stored in the file indicated by *hUpdate*, not the data provided by LoadIcon, LoadString, or other resource-specific load functions. All data containing strings or text must be in Unicode format. *lpData* must not point to ANSI data.

If *lpData* is **NULL** and *cbData* is 0, the specified resource is deleted from the file indicated by *hUpdate*.

`[in] cb`

Type: **DWORD**

The size, in bytes, of the resource data at *lpData*.

Return value

Type: **BOOL**

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call GetLastError.

Remarks

It is recommended that the resource file is not loaded before this function is called. However, if that file is already loaded, it will not cause an error to be returned.

An application can use **UpdateResource** repeatedly to make changes to the resource data. Each call to **UpdateResource** contributes to an internal list of additions, deletions, and replacements but does not actually write the data to the file indicated by *hUpdate*. The application must use the EndUpdateResource function to write the accumulated changes to the file.

This function can update resources within modules that contain both code and resources.

Prior to Windows 7: If *lpData* is **NULL** and *cbData* is nonzero, the specified resource is NOT deleted and an exception is thrown.

Starting with Windows Vista: As noted above, there are restrictions on resource updates in files that contain RC Config data: LN files and .mui files. The restrictions are as follows:

Expand table

Action	LN file	.mui file
1. Add a new type that doesn't exist in the LN or .mui files.	Add type in the LN file and treat as language-neutral (non-localizable) and add new type or item in the RC Config data	The only additions allowed are the following types: file Version, RC Config data, Side-by-side Assembly XML Manifest.
2. Add a new resource item to an existing type.	Uses the RC Config data to check whether the type exists in the .mui files associated with this LN file. If the type doesn't exist in the .mui files, add the item and treat new item as un-localizable. If the type exists in the .mui files, then adding is not allowed.	Only items of the following types may be added: File Version, RC Config data, Side-by-side Assembly XML Manifest.
3. Update a resource item.	Uses the RC Config data to check whether the type exists in the .mui files associated with the LN file. If the type doesn't exist in the .mui files, then this resource item update is allowed in the LN file. Otherwise, if the type exists in the .mui files associated with this LN file, then this update is not allowed.	The only updates allowed are items of the following types: file Version, RC Config data, Side-by-side Assembly XML Manifest.
4. Add a type/item for a new language.	Not allowed.	Not allowed.

5. Remove an existing type/item.	Works similarly to case 3. Uses the RC Config data to check whether the type exists in the .mui files associated with the LN file. If not, then the removal of the type/item from the LN file is allowed. Otherwise, if the type/item exists in the .mui files associated with this LN file, then the removal is not allowed.	The only types allowed to be removed are: file Version, RC Config data, Side-by-side Assembly XML Manifest; also, only items of these types may be removed.
6. Add/delete/update a type not included in the RC Config data (such as Version, Side-by-side Assembly XML Manifest, or RC Config data itself).	Allowed.	Allowed.
7. Other update of non-localizable data, such as TYPELIB, reginst, and so on.	Update type or item in the LN file, treat as non-localizable, and add new type or item in the RC Config data.	Not applicable.
8. Add RC Config data.	Can be done but the integrity of the RC Config data is not checked.	Can be done but the integrity of the RC Config data is not checked.

Examples

For an example, see Updating Resources.

> **Note**
>
> The winbase.h header defines UpdateResource as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

BeginUpdateResource

Conceptual

EndUpdateResource

LoadIcon

LoadString

LockResource

MAKEINTRESOURCE

MAKELANGID

Other Resources

Reference

Resources

SizeofResource

Feedback

Resource Macros

Article • 04/27/2021

In This Section

- IS_INTRESOURCE
- MAKEINTRESOURCE

Feedback

Was this page helpful? Yes No

IS_INTRESOURCE macro (winuser.h)

Article02/22/2024

Determines whether a value is an integer identifier for a resource.

Syntax

```
C++

void IS_INTRESOURCE(
   _r
);
```

Parameters

`_r`

The pointer to be tested whether it contains an integer resource identifier.

Return value

None

Remarks

This macro checks whether all bits except the least 16 bits are zero. When true, *p* is an integer identifier for a resource. Otherwise it is typically a pointer to a string.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	winuser.h (include Windows.h)

See also

Resources Overview

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MAKEINTRESOURCEA macro (winuser.h)

Article02/22/2024

Converts an integer value to a resource type compatible with the resource-management functions. This macro is used in place of a string containing the name of the resource.

Syntax

```
C++

void MAKEINTRESOURCEA(
   i
);
```

Parameters

`i`

The integer value to be converted.

Return value

None

Remarks

The return value should be passed only to functions which explicitly indicate that they accept **MAKEINTRESOURCE** as a parameter. For example, the resource management functions allow the return value of **MAKEINTRESOURCE** to be passed as the *lpType* or *lpName* parameters.

> **Note**
>
> The winuser.h header defines MAKEINTRESOURCE as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

See also

Resources Overview

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Resource Structures (Menus and Other Resources)

Article • 04/27/2021

In This Section

- ACCELTABLEENTRY
- CURSORDIR
- CURSORSHAPE
- DIRENTRY
- FONTDIRENTRY
- FONTGROUPHDR
- ICONRESDIR
- LOCALHEADER
- MENUHEADER
- MENUHELPID
- MESSAGE_RESOURCE_BLOCK
- MESSAGE_RESOURCE_DATA
- MESSAGE_RESOURCE_ENTRY
- NEWHEADER
- NORMALMENUITEM
- POPUPMENUITEM
- RESDIR
- RESOURCEHEADER

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

ACCELTABLEENTRY structure

Article • 12/11/2020

Describes the data in an individual accelerator table resource. The structure definition provided here is for explanation only; it is not present in any standard header file.

Syntax

C++

```
typedef struct {
  WORD fFlags;
  WORD wAnsi;
  WORD wId;
  WORD padding;
} ACCELTABLEENTRY;
```

Members

fFlags

Type: **WORD**

Describes keyboard accelerator characteristics. This member can have one or more of the following values from Winuser.h.

Value	Meaning
FVIRTKEY TRUE	The accelerator key is a virtual-key code. If this flag is not specified, the accelerator key is assumed to specify an ASCII character code.
FNOINVERT 0x02	A menu item on the menu bar is not highlighted when an accelerator is used. This attribute is obsolete and retained only for backward compatibility with resource files designed for 16-bit Windows.
FSHIFT 0x04	The accelerator is activated only if the user presses the SHIFT key. This flag applies only to virtual keys.
FCONTROL 0x08	The accelerator is activated only if the user presses the CTRL key. This flag applies only to virtual keys.
FALT 0x10	The accelerator is activated only if the user presses the ALT key. This flag applies only to virtual keys.
0x80	The entry is last in an accelerator table.

wAnsi

Type: **WORD**

An ANSI character value or a virtual-key code that identifies the accelerator key.

wId

Type: **WORD**

An identifier for the keyboard accelerator. This is the value passed to the window procedure when the user presses the specified key.

padding

Type: **WORD**

The number of bytes inserted to ensure that the structure is aligned on a **DWORD** boundary.

Remarks

The **ACCELTABLEENTRY** structure is repeated for all accelerator table entries in the resource. The last entry in the table is flagged with the value 0x0080.

You can compute the number of elements in the table if you divide the length of the resource by eight. Then your application can randomly access the individual fixed-length entries.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

CreateAcceleratorTable

Conceptual

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

CURSORDIR structure

Article • 06/17/2023

Contains the dimensions of an individual cursor image in a resource group.

The structure definition provided here is for explanation only; it is not present in any standard header file.

Syntax

```
C++
```

```
typedef struct {
  WORD Width;
  WORD Height;
} CURSORDIR;
```

Members

Width

Type: **WORD**

The width of the cursor, in pixels.

The value 0 is accepted as representing a width of 256.

Height

Type: **WORD**

The height of the cursor, in pixels.

The value 0 is accepted as representing a height of 256.

Remarks

The **CURSORDIR** structure is passed in the **RESDIR** structure if the **RESDIR** structure describes a cursor.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

RESDIR

Conceptual

Resources

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

CURSORSHAPE structure (winuser.h)

Article02/22/2024

Contains information about a cursor.

Syntax

C++

```
typedef struct tagCURSORSHAPE {
  int  xHotSpot;
  int  yHotSpot;
  int  cx;
  int  cy;
  int  cbWidth;
  BYTE Planes;
  BYTE BitsPixel;
} CURSORSHAPE, *LPCURSORSHAPE;
```

Members

`xHotSpot`

Type: **int**

The horizontal position of the hot spot, relative to the upper-left corner of the cursor bitmap.

`yHotSpot`

Type: **int**

The vertical position of the hot spot, relative to the upper-left corner of the cursor bitmap.

`cx`

Type: **int**

The width, in pixels, of the cursor.

`cy`

Type: **int**

The height, in pixels, of the cursor.

`cbWidth`

Type: **int**

The width, in bytes, of the cursor bitmap.

`Planes`

Type: **BYTE**

The number of color planes.

`BitsPixel`

Type: **BYTE**

The number of bits used to indicate the color of a single pixel in the cursor.

Remarks

When an application passes a cursor handle to the LockResource function, the function returns a pointer to a buffer containing information about the cursor. An application can use the **CURSORSHAPE** structure to access the information.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

LockResource

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

DIRENTRY structure

Article • 12/11/2020

Contains a unique ordinal that identifies an individual font in the font resource group. The structure definition provided here is for explanation only; it is not present in any standard header file.

Syntax

```
C++
```

```
typedef struct {
  WORD fontOrdinal;
} DIRENTRY;
```

Members

fontOrdinal

Type: **WORD**

A unique ordinal identifier for an individual font in a font resource group.

Remarks

The **FONTDIRENTRY** structure for the specified font directly follows the **DIRENTRY** structure for that font.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

FONTDIRENTRY

FONTGROUPHDR

Conceptual

Resources

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

FONTDIRENTRY structure

Article • 12/11/2020

Contains information about an individual font in a font resource group. The structure definition provided here is for explanation only; it is not present in any standard header file.

Syntax

C++

```
typedef struct {
  WORD  dfVersion;
  DWORD dfSize;
  CHAR  dfCopyright[60];
  WORD  dfType;
  WORD  dfPoints;
  WORD  dfVertRes;
  WORD  dfHorizRes;
  WORD  dfAscent;
  WORD  dfInternalLeading;
  WORD  dfExternalLeading;
  BYTE  dfItalic;
  BYTE  dfUnderline;
  BYTE  dfStrikeOut;
  WORD  dfWeight;
  BYTE  dfCharSet;
  WORD  dfPixWidth;
  WORD  dfPixHeight;
  BYTE  dfPitchAndFamily;
  WORD  dfAvgWidth;
  WORD  dfMaxWidth;
  BYTE  dfFirstChar;
  BYTE  dfLastChar;
  BYTE  dfDefaultChar;
  BYTE  dfBreakChar;
  WORD  dfWidthBytes;
  DWORD dfDevice;
  DWORD dfFace;
  DWORD dfReserved;
  CHAR  szDeviceName;
  CHAR  szFaceName;
} FONTDIRENTRY;
```

Members

dfVersion

Type: **WORD**

A user-defined version number for the resource data that tools can use to read and write resource files.

dfSize

Type: **DWORD**

The size of the file, in bytes.

dfCopyright[60]

Type: **CHAR**

The font supplier's copyright information.

dfType

Type: **WORD**

The type of font file.

dfPoints

Type: **WORD**

The point size at which this character set looks best.

dfVertRes

Type: **WORD**

The vertical resolution, in dots per inch, at which this character set was digitized.

dfHorizRes

Type: **WORD**

The horizontal resolution, in dots per inch, at which this character set was digitized.

dfAscent

Type: **WORD**

The distance from the top of a character definition cell to the baseline of the typographical font.

dfInternalLeading

Type: **WORD**

The amount of leading inside the bounds set by the **dfPixHeight** member. Accent marks and other diacritical characters can occur in this area.

dfExternalLeading

Type: **WORD**

The amount of extra leading that the application adds between rows.

dfItalic

Type: **BYTE**

An italic font if not equal to zero.

dfUnderline

Type: **BYTE**

An underlined font if not equal to zero.

dfStrikeOut

Type: **BYTE**

A strikeout font if not equal to zero.

dfWeight

Type: **WORD**

The weight of the font in the range 0 through 1000. For example, 400 is roman and 700 is bold. If this value is zero, a default weight is used. For additional defined values, see the description of the LOGFONT structure.

dfCharSet

Type: **BYTE**

The character set of the font. For predefined values, see the description of the LOGFONT structure.

dfPixWidth

Type: **WORD**

The width of the grid on which a vector font was digitized. For raster fonts, if the member is not equal to zero, it represents the width for all the characters in the bitmap. If the member is equal to zero, the font has variable-width characters.

dfPixHeight

Type: **WORD**

The height of the character bitmap for raster fonts or the height of the grid on which a vector font was digitized.

dfPitchAndFamily

Type: **BYTE**

The pitch and the family of the font. For additional information, see the description of the LOGFONT structure.

dfAvgWidth

Type: **WORD**

The average width of characters in the font (generally defined as the width of the letter x). This value does not include the overhang required for bold or italic characters.

dfMaxWidth

Type: **WORD**

The width of the widest character in the font.

dfFirstChar

Type: **BYTE**

The first character code defined in the font.

dfLastChar

Type: **BYTE**

The last character code defined in the font.

dfDefaultChar

Type: **BYTE**

The character to substitute for characters not in the font.

dfBreakChar

Type: **BYTE**

The character that will be used to define word breaks for text justification.

dfWidthBytes

Type: **WORD**

The number of bytes in each row of the bitmap. This value is always even so that the rows start on word boundaries. For vector fonts, this member has no meaning.

dfDevice

Type: **DWORD**

The offset in the file to a null-terminated string that specifies a device name. For a generic font, this value is zero.

dfFace

Type: **DWORD**

The offset in the file to a null-terminated string that names the typeface.

dfReserved

Type: **DWORD**

This member is reserved.

szDeviceName

Type: **CHAR**

The name of the device if this font file is designated for a specific device.

szFaceName

Type: **CHAR**

The typeface name of the font.

Remarks

There is one **FONTDIRENTRY** structure for every font in the .res file. Applications that generate .res files with font resources must also add to the file a **FONTDIRENTRY**

structure for each font.

Font declarations can be mixed with other resource declarations in the .RC file because fonts do not need to be contiguous in the .res file.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

DIRENTRY

FONTGROUPHDR

Conceptual

Resources

Other Resources

LOGFONT

Feedback

FONTGROUPHDR structure

Article • 12/11/2020

Contains the information necessary for an application to access a specific font. The structure definition provided here is for explanation only; it is not present in any standard header file.

Syntax

```
C++
```

```
typedef struct {
  WORD     NumberOfFonts;
  DIRENTRY DE;
} FONTGROUPHDR;
```

Members

NumberOfFonts

Type: **WORD**

The number of individual fonts associated with this resource.

DE

Type: DIRENTRY

A structure that contains a unique ordinal identifier for each font in the resource. The **DE** member is a placeholder for the variable-length array of DIRENTRY structures.

Remarks

The **FONTGROUPHDR** structure follows the data for the individual fonts in the .Res file. The resource compiler automatically adds the **FONTGROUPHDR** structure, generally as the last entry in the file.

Requirements

Requirement	Value

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

DIRENTRY

FONTDIRENTRY

Conceptual

Resources

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

ICONRESDIR structure

Article • 06/17/2023

Contains the dimensions and color format of an individual icon image in a resource group.

The structure definition provided here is for explanation only; it is not present in any standard header file.

Syntax

C++

```
typedef struct {
  BYTE Width;
  BYTE Height;
  BYTE ColorCount;
  BYTE reserved;
} ICONRESDIR;
```

Members

Width

Type: **BYTE**

The width of the icon, in pixels.

The value 0 is accepted as representing a width of 256.

Height

Type: **BYTE**

The height of the icon, in pixels.

The value 0 is accepted as representing a height of 256.

ColorCount

Type: **BYTE**

The number of colors in the icon.

Acceptable values are 2, 8, and 16. The value 0 means that the number of colors deduced from **BitCount** and **Planes** in the RESDIR structure.

reserved

Type: **BYTE**

Reserved; must be set to the same value as that of the reserved field in the icon file header.

Remarks

The **ICONRESDIR** structure is passed in the RESDIR structure if the **RESDIR** structure describes an icon.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

RESDIR

Conceptual

Resources

Feedback

IndexedResourceQualifier structure (resourceindexer.h)

Article02/22/2024

Represents the context under which a resource is appropriate.

Syntax

```
C++
```

```
typedef struct {
  PWSTR name;
  PWSTR value;
} IndexedResourceQualifier;
```

Members

`name`

The name of the qualifier, such as "language", "contrast", or "scale".

`value`

The value of the qualifier. You should preserve the case of the qualifier value from the first instance of the qualifier discovered during indexing.

The following values are examples of the qualifier values:

- "100", "140", or "180" for scale.
- "fr-FR" for language.
- "high" for contrast.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]

Requirement	Value
Header	resourceindexer.h

See also

DestroyIndexedResults

IndexFilePath

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

LOCALHEADER structure

Article • 12/11/2020

Contains the x- and y-coordinates of a hotspot associated with the cursor identified by a RESDIR structure. The structure definition provided here is for explanation only; it is not present in any standard header file.

Syntax

C++

```
typedef struct {
  WORD xHotSpot;
  WORD yHotSpot;
} LOCALHEADER;
```

Members

xHotSpot

Type: **WORD**

The x-coordinate of the cursor hot spot, in pixels.

yHotSpot

Type: **WORD**

The y-coordinate of the cursor hot spot, in pixels.

Remarks

The **LOCALHEADER** structure is the first data written to the RT_CURSOR resource if a RESDIR structure contains information about a cursor.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

CURSORDIR

RESDIR

Conceptual

Resources

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

MENUHEADER structure

Article • 12/11/2020

Contains version information for the menu resource. The structure definition provided here is for explanation only; it is not present in any standard header file.

Syntax

```
C++
```

```
typedef struct {
  WORD wVersion;
  WORD cbHeaderSize;
} MENUHEADER;
```

Members

wVersion

Type: **WORD**

The version number of the menu template. This member must be equal to zero to indicate that this is an RT_MENU created with a standard menu template.

cbHeaderSize

Type: **WORD**

The size of the menu template header. This value is zero for menus you create with a standard menu template.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

MENUEX_TEMPLATE_HEADER

MENUEX_TEMPLATE_ITEM

MENUITEMTEMPLATE

MENUITEMTEMPLATEHEADER

NORMALMENUITEM

POPUPMENUITEM

Conceptual

Resources

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

MENUHELPID structure

Article • 12/11/2020

Contains the final data written to the RT_MENU resource for a menu or submenu if the **resInfo** member of the POPUPMENUITEM structure is set to **MFR_POPUP**. The structure definition provided here is for explanation only; it is not present in any standard header file.

Syntax

C++

```
typedef struct {
  DWORD helpID;
} MENUHELPID;
```

Members

helpID

Type: **DWORD**

The identifier used to identify the menu during WM_HELP processing.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

MENUHEADER

POPUPMENUITEM

Conceptual

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

MESSAGE_RESOURCE_BLOCK structure (winnt.h)

Article02/22/2024

Contains information about message strings with identifiers in the range indicated by the **LowId** and **HighId** members.

Syntax

```
C++
```

```
typedef struct _MESSAGE_RESOURCE_BLOCK {
  DWORD LowId;
  DWORD HighId;
  DWORD OffsetToEntries;
} MESSAGE_RESOURCE_BLOCK, *PMESSAGE_RESOURCE_BLOCK;
```

Members

`LowId`

Type: **DWORD**

The lowest message identifier contained within this structure.

`HighId`

Type: **DWORD**

The highest message identifier contained within this structure.

`OffsetToEntries`

Type: **DWORD**

The offset, in bytes, from the beginning of the MESSAGE_RESOURCE_DATA structure to the MESSAGE_RESOURCE_ENTRY structures in this **MESSAGE_RESOURCE_BLOCK**. The **MESSAGE_RESOURCE_ENTRY** structures contain the message strings.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winnt.h (include Windows.h)

See also

Conceptual

MESSAGE_RESOURCE_DATA

MESSAGE_RESOURCE_ENTRY

Reference

Resources

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MESSAGE_RESOURCE_DATA structure (winnt.h)

Article02/22/2024

Contains information about formatted text for display as an error message or in a message box in a message table resource.

Syntax

```
C++
```

```
typedef struct _MESSAGE_RESOURCE_DATA {
  DWORD                   NumberOfBlocks;
  MESSAGE_RESOURCE_BLOCK Blocks[1];
} MESSAGE_RESOURCE_DATA, *PMESSAGE_RESOURCE_DATA;
```

Members

`NumberOfBlocks`

Type: **DWORD**

The number of MESSAGE_RESOURCE_BLOCK structures.

`Blocks[1]`

Type: **MESSAGE_RESOURCE_BLOCK[1]**

An array of structures. The array is the size indicated by the **NumberOfBlocks** member.

Remarks

A **MESSAGE_RESOURCE_DATA** structure can contain one or more MESSAGE_RESOURCE_BLOCK structures, which can each contain one or more MESSAGE_RESOURCE_ENTRY structures.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winnt.h (include Windows.h)

See also

Conceptual

MESSAGE_RESOURCE_BLOCK

MESSAGE_RESOURCE_ENTRY

Reference

Resources

Feedback

MESSAGE_RESOURCE_ENTRY structure (winnt.h)

Article02/22/2024

Contains the error message or message box display text for a message table resource.

Syntax

```
C++
```

```
typedef struct _MESSAGE_RESOURCE_ENTRY {
  WORD Length;
  WORD Flags;
  BYTE Text[1];
} MESSAGE_RESOURCE_ENTRY, *PMESSAGE_RESOURCE_ENTRY;
```

Members

`Length`

Type: **WORD**

The length, in bytes, of the **MESSAGE_RESOURCE_ENTRY** structure.

`Flags`

Type: **WORD**

Indicates that the string is encoded in Unicode, if equal to the value 0x0001. Indicates that the string is encoded in ANSI, if equal to the value 0x0000.

`Text[1]`

Type: **BYTE[1]**

Pointer to an array that contains the error message or message box display text.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winnt.h (include Windows.h)

See also

Conceptual

MESSAGE_RESOURCE_BLOCK

MESSAGE_RESOURCE_DATA

Reference

Resources

Feedback

NEWHEADER structure

Article • 12/11/2020

Contains the number of icon or cursor components in a resource group. The structure definition provided here is for explanation only; it is not present in any standard header file.

Syntax

```
C++
```

```
typedef struct {
  WORD Reserved;
  WORD ResType;
  WORD ResCount;
} NEWHEADER, *NEWHEADER;
```

Members

Reserved

Type: **WORD**

Reserved; must be zero.

ResType

Type: **WORD**

The resource type. This member must have one of the following values.

Value	Meaning
RES_CURSOR 2	Cursor resource type.
RES_ICON 1	Icon resource type.

ResCount

Type: **WORD**

The number of icon or cursor components in the resource group.

Remarks

One or more RESDIR structures immediately follow the **NEWHEADER** structure in the .res file. The **ResCount** member specifies the number of **RESDIR** structures.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

RESDIR

Conceptual

Resources

Feedback

NORMALMENUITEM structure

Article • 12/11/2020

Contains information about each item in a menu resource that does not open a menu or a submenu. The structure definition provided here is for explanation only; it is not present in any standard header file.

Syntax

```
C++
```

```
typedef struct {
  WORD    resInfo;
  szOrOrd menuText;
} NORMALMENUITEM;
```

Members

resInfo

Type: **WORD**

The type of menu item. This member can be one of the following values.

Value	Meaning
MFR_END 0x80	The menu item is the last in this submenu or menu resource; this flag is used internally by the system.
MFR_POPUP 0x01	The menu item opens a menu or a submenu; the flag is used internally by the system.

menuText

Type: **szOrOrd**

A null-terminated Unicode string that contains the text for this menu item. There is no fixed limit on the size of this string.

Remarks

There is one **NORMALMENUITEM** structure for each menu item that does not open a menu or a submenu. Indicate the last menu item on a menu by setting the **resInfo** member to **MFR_END**.

A menu separator is a special type of menu item that is inactive but appears as a dividing bar between two active menu items. Indicate a menu separator by leaving the **menuText** member empty.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

MENUHEADER

MENUITEMINFO

POPUPMENUITEM

Conceptual

Resources

Feedback

POPUPMENUITEM structure

Article • 12/11/2020

Contains information about the menu items in a menu resource that open a menu or a submenu. The structure definition provided here is for explanation only; it is not present in any standard header file.

Syntax

C++

```
typedef struct {
  DWORD   type;
  DWORD   state;
  DWORD   id;
  WORD    resInfo;
  szOrOrd menuText;
} POPUPMENUITEM;
```

Members

type

Type: **DWORD**

Describes the menu item. Some of the values this member can have include those shown in the list below.

In addition to the values shown, this member can also be a combination of the type values listed with the **fType** member of the MENUITEMINFO structure. The type values are those that begin with MFT_. To use these predefined MFT_* type values, include the following statement in your .rc file:

```
#include "winuser.h"
```

Value	Meaning
MF_END 0x80	The menu item is the last on the menu; the flag is used internally by the system.
MF_POPUP 0x01	The menu item opens a menu or a submenu; the flag is used internally by the system.

state

Type: **DWORD**

Describes the menu item. This member can be a combination of the state values listed with the **dwState** member of the MENUITEMINFO structure. The state values are those that begin with MFS_. To use these predefined MFS_* state values, include the following statement in your .rc file:

```
#include "winuser.h"
```

id

Type: **DWORD**

A numeric expression that identifies the menu item that is passed in the WM_COMMAND message.

resInfo

Type: **WORD**

A set of bit flags that specify the type of menu item. This member can be one of the following values.

Value	Meaning
MFR_END 0x80	The menu item is the last in this submenu or menu resource; this flag is used internally by the system.
MFR_POPUP 0x01	The menu item opens a menu or a submenu; the flag is used internally by the system.

menuText

Type: **szOrOrd**

A null-terminated Unicode string that contains the text for this menu item. There is no fixed limit on the size of this string.

Remarks

There is one **POPUPMENUITEM** structure for each menu item that opens a menu or a submenu. Identify this type of menu item by setting the **type** member to **MF_POPUP** and by setting the **MFR_POPUP** bit in the **resInfo** member to 0x0001. In this case, the final data written to the RT_MENU resource for the menu or submenu is the

MENUHELPID structure. **MENUHELPID** contains a numeric expression that identifies the menu during WM_HELP processing.

Additionally, every **POPUPMENUITEM** structure that has the **MFR_POPUP** bit set in the **resInfo** member will be followed by a MENUHELPID structure plus an additional number of **POPUPMENUITEM** structures, one for each menu item in that submenu. The last **POPUPMENUITEM** structure in the submenu will have the **MFR_END** bit set in the **resInfo** member. To find the end of the resource, look for a matching **MFR_END** for every **MFR_POPUP** plus one additional **MFR_END** that matches the outermost set of menu items.

Indicate the last menu item by setting the **type** member to **MF_END**. Because you can nest submenus, there can be multiple levels of **MF_END**. In these instances, the menu items are sequential.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

MENUHEADER

MENUHELPID

MENUITEMINFO

NORMALMENUITEM

Conceptual

Resources

Feedback

Get help at Microsoft Q&A

RESDIR structure

Article • 06/17/2023

Contains information about an individual icon or cursor component in a RT_GROUP_ICON or RT_GROUP_CURSOR resource group.

There is one **RESDIR** structure for each group component.

The structure definition provided here is for explanation only; it is not present in any standard header file.

Syntax

```
C++
```

```
typedef struct {
  union
  {
      ICONRESDIR   Icon;
      CURSORDIR    Cursor;
  };
  WORD       Planes;
  WORD       BitCount;
  DWORD      BytesInRes;
  WORD       IconCursorId;
} RESDIR;
```

Members

Icon

Type: **ICONRESDIR**

The width, height, and color count of the indicated icon.

Cursor

Type: **CURSORDIR**

The width and height of the indicated cursor.

Planes

Type: **WORD**

The number of color planes in the icon or cursor bitmap.

BitCount

Type: **WORD**

The number of bits per pixel in the icon or cursor bitmap.

BytesInRes

Type: **DWORD**

The size of the resource, in bytes.

IconCursorId

Type: **WORD**

Unique ordinal identifier of the RT_ICON icon or RT_CURSOR cursor resource.

Remarks

One or more **RESDIR** structures immediately follow the **NEWHEADER** structure in the .res file. The **ResCount** member of the **NEWHEADER** structure specifies the number of **RESDIR** structures. Note that the **RESDIR** structure consists of either an **ICONRESDIR** structure or a **CURSORDIR** structure followed by the **Planes**, **BitCount**, **BytesInRes**, and **IconCursorId** members.

If the **RESDIR** structure contains information about a cursor, the first two **WORD**s the resource compiler writes to the RT_CURSOR resource are the **xHotSpot** and **yHotSpot** members of the **LOCALHEADER** structure.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

CURSORDIR

ICONRESDIR

LOCALHEADER

NEWHEADER

Conceptual

Resources

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

RESOURCEHEADER structure

Article • 12/11/2020

Contains information about the resource header itself and the data specific to this resource. This structure is not a true C-language structure, because it contains variable-length members. The structure definition provided here is for explanation only; it is not present in any standard header file.

Syntax

C++

```
typedef struct {
  DWORD DataSize;
  DWORD HeaderSize;
  DWORD TYPE;
  DWORD NAME;
  DWORD DataVersion;
  WORD  MemoryFlags;
  WORD  LanguageId;
  DWORD Version;
  DWORD Characteristics;
} RESOURCEHEADER;
```

Members

DataSize

Type: **DWORD**

The size, in bytes, of the data that follows the resource header for this particular resource. It does not include any file padding between this resource and any resource that follows it in the resource file.

HeaderSize

Type: **DWORD**

The size, in bytes, of the resource header data that follows.

TYPE

Type: **DWORD**

The resource type. The **TYPE** member can either be a numeric value or a null-terminated Unicode string that specifies the name of the type. See the following Remarks section for a description of **Name** or **Ordinal** type members.

If the **TYPE** member is a numeric value, it can specify either a standard or a user-defined resource type. If the member is a string, then it is a user-defined resource type. For a list of the predefined resource types, see Resource Types.

Values less than 256 are reserved for system use.

NAME

Type: **DWORD**

A name that identifies the particular resource. The **NAME** member, like the **TYPE** member, can either be a numeric value or a null-terminated Unicode string. See the following Remarks section for a description of **Name** or **Ordinal** type members.

You do not need to add padding for **DWORD** alignment between the **TYPE** and **NAME** members because they contain **WORD** data. However, you may need to add a **WORD** of padding after the **NAME** member to align the rest of the header on **DWORD** boundaries.

DataVersion

Type: **DWORD**

A predefined resource data version. This will determine which version of the resource data the application should use.

MemoryFlags

Type: **WORD**

A set of attribute flags that can describe the state of the resource. Modifiers in the .RC script file assign these attributes to the resource. The script identifiers can assign the following flag values.

Applications do not use any of these attributes. The attributes are permitted in the script for backward compatibility with existing scripts, but they are ignored. Resources are loaded when the corresponding module is loaded, and are freed when the module is unloaded.

MOVEABLE (0x0010)

FIXED (~MOVEABLE)

PURE (0x0020)

IMPURE (~PURE)

PRELOAD (0x0040)

LOADONCALL (~PRELOAD)

DISCARDABLE (0x1000)

LanguageId

Type: **WORD**

The language for the resource or set of resources. Set the value for this member with the optional LANGUAGE resource definition statement. The parameters are constants from the Winnt.h file.

Each resource includes a language identifier so the system or application can select a language appropriate for the current locale of the system. If there are multiple resources of the same type and name that differ only in the language of the strings within the resources, you will need to specify a **LanguageId** for each one.

Version

Type: **DWORD**

A user-defined version number for the resource data that tools can use to read and write resource files. Set this value with the optional VERSION resource definition statement.

Characteristics

Type: **DWORD**

Specifies user-defined information about the resource that tools can use to read and write resource files. Set this value with the optional CHARACTERISTICS resource definition statement.

Remarks

A variable type member is called a **Name** or **Ordinal** member, and it is used in most places in the resource file where an identifier appears. The first **WORD** of a **Name** or **Ordinal** type member indicates whether the member is a numeric value or a string. If the first **WORD** in the member is equal to the value 0xffff, which is an invalid Unicode

character, then the following **WORD** is a type number. Otherwise, the member contains a Unicode string and the first **WORD** in the member is the first character in the name string. For additional information about resource definition statements, see Resource-Definition Statements.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Conceptual

Resources

Other Resources

CHARACTERISTICS Statement

LANGUAGE Statement

VERSION Statement

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Resource Types

Article • 03/22/2021

The following are the predefined resource types.

Constant/value	Description
RT_ACCELERATOR MAKEINTRESOURCE(9)	Accelerator table.
RT_ANICURSOR MAKEINTRESOURCE(21)	Animated cursor.
RT_ANIICON MAKEINTRESOURCE(22)	Animated icon.
RT_BITMAP MAKEINTRESOURCE(2)	Bitmap resource.
RT_CURSOR MAKEINTRESOURCE(1)	Hardware-dependent cursor resource.
RT_DIALOG MAKEINTRESOURCE(5)	Dialog box.
RT_DLGINCLUDE MAKEINTRESOURCE(17)	Allows a resource editing tool to associate a string with an .rc file. Typically, the string is the name of the header file that provides symbolic names. The resource compiler parses the string but otherwise ignores the value. For example, `1 DLGINCLUDE "MyFile.h"`
RT_FONT MAKEINTRESOURCE(8)	Font resource.
RT_FONTDIR MAKEINTRESOURCE(7)	Font directory resource.
RT_GROUP_CURSOR MAKEINTRESOURCE((ULONG_PTR)(RT_CURSOR) + 11)	Hardware-independent cursor resource.
RT_GROUP_ICON MAKEINTRESOURCE((ULONG_PTR)(RT_ICON) + 11)	Hardware-independent icon resource.
RT_HTML MAKEINTRESOURCE(23)	HTML resource.

Constant/value	Description
RT_ICON MAKEINTRESOURCE(3)	Hardware-dependent icon resource.
RT_MANIFEST MAKEINTRESOURCE(24)	Side-by-Side Assembly Manifest.
RT_MENU MAKEINTRESOURCE(4)	Menu resource.
RT_MESSAGETABLE MAKEINTRESOURCE(11)	Message-table entry.
RT_PLUGPLAY MAKEINTRESOURCE(19)	Plug and Play resource.
RT_RCDATA MAKEINTRESOURCE(10)	Application-defined resource (raw data).
RT_STRING MAKEINTRESOURCE(6)	String-table entry.
RT_VERSION MAKEINTRESOURCE(16)	Version resource.
RT_VXD MAKEINTRESOURCE(20)	VXD.

Requirements

Requirement	Value
Header	Winuser.h

Feedback

Carets

Article • 11/19/2022

A *caret* is a blinking line, block, or bitmap in the client area of a window. The caret typically indicates the place at which text or graphics will be inserted.

The following illustration shows some common variations in the appearance of the caret.

Applications can create a caret, change its blink time, and display, hide, or relocate the caret.

In This Section

Name	Description
About Carets	Discusses carets.
Using Carets	Code samples that show how to perform tasks related to carets.
Caret Reference	Contains the API reference.

Caret Functions

Name	Description
CreateCaret	Creates a new shape for the system caret and assigns ownership of the caret to the specified window. The caret shape can be a line, a block, or a bitmap.
DestroyCaret	Destroys the caret's current shape, frees the caret from the window, and removes the caret from the screen.
GetCaretBlinkTime	Retrieves the time required to invert the caret's pixels. The user can set this value.
GetCaretPos	Copies the caret's position to the specified POINT structure.
HideCaret	Removes the caret from the screen. Hiding a caret does not destroy its current shape or invalidate the insertion point.
SetCaretBlinkTime	Sets the caret blink time to the specified number of milliseconds. The blink time is the elapsed time, in milliseconds, required to invert the caret's pixels.

Name	Description
SetCaretPos	Moves the caret to the specified coordinates. If the window that owns the caret was created with the **CS_OWNDC** class style, then the specified coordinates are subject to the mapping mode of the device context associated with that window.
ShowCaret	Makes the caret visible on the screen at the caret's current position. When the caret becomes visible, it begins flashing automatically.

Feedback

About Carets

Article • 08/19/2020

The system provides one caret per message queue. A window should create a caret only when it has the keyboard focus or is active. The window should destroy the caret before losing the keyboard focus or becoming inactive. For more information on keyboard input, see Keyboard Input.

Use the **CreateCaret** function to specify the parameters for a caret. The system forms a caret by inverting the pixel color within the rectangle specified by the caret's position, width, and height. The width and height are specified in logical units; therefore, the appearance of a caret is subject to the window's mapping mode.

The following topics are discussed in this section.

- Caret Visibility
- Caret Blink Time
- Caret Position
- Removing a Caret

Caret Visibility

After the caret is defined, use the **ShowCaret** function to make the caret visible. When the caret appears, it automatically begins flashing. To display a solid caret, the system inverts every pixel in the rectangle; to display a gray caret, the system inverts every other pixel; to display a bitmap caret, the system inverts only the white bits of the bitmap.

Caret Blink Time

The elapsed time, in milliseconds, required to invert the caret is called the *blink time*. The caret will blink as long as the thread that owns the message queue has a message pump processing the messages.

The user can set the blink time of the caret using the Control Panel and applications should respect the settings that the user has chosen. An application can determine the caret's blink time by using the **GetCaretBlinkTime** function. If you are writing an application that allows the user to adjust the blink time, such as a Control Panel applet, use the **SetCaretBlinkTime** function to set the rate of the blink time to a specified number of milliseconds.

The *flash time* is the elapsed time, in milliseconds, required to display, invert, and restore the caret's display. The flash time of a caret is twice as much as the blink time.

Caret Position

You can determine the position of the caret using the GetCaretPos function. The position, in client coordinates, is copied to a structure specified by a parameter in **GetCaretPos**. An application can move a caret in a window by using the SetCaretPos function. A window can move a caret only if it already owns the caret. **SetCaretPos** can move the caret whether it is visible or not.

Removing a Caret

You can temporarily remove a caret by hiding it, or you can permanently remove the caret by destroying it. To hide the caret, use the HideCaret function. This is useful when your application must redraw the screen while processing a message, but must keep the caret out of the way. When the application finishes drawing, it can display the caret again by using the ShowCaret function. Hiding the caret does not destroy its shape or invalidate the insertion point. Hiding the caret is cumulative; that is, if the application calls **HideCaret** five times, it must also call **ShowCaret** five times before the caret will reappear.

To remove the caret from the screen and destroy its shape, use using the DestroyCaret function. **DestroyCaret** destroys the caret only if the window involved in the current task owns the caret.

Using Carets

Article • 08/19/2020

This section has code samples for the following tasks:

- Creating and Displaying a Caret
- Hiding a Caret
- Destroying a Caret
- Adjusting the Blink Time
- Processing Keyboard Input

Creating and Displaying a Caret

Upon receiving the keyboard focus, the window should create and display the caret. Use the **CreateCaret** function to create a caret in the given window. You can then call **SetCaretPos** to set the current position of the caret and **ShowCaret** to make the caret visible.

The system sends the **WM_SETFOCUS** message to the window receiving keyboard focus; therefore, an application should create and display the caret while processing this message.

```
HWND hwnd,       // window handle
int x;           // horizontal coordinate of cursor
int y;           // vertical coordinate of cursor
int nWidth;      // width of cursor
int nHeight;     // height of cursor
char *lpszChar;  // pointer to character

    case WM_SETFOCUS:

    // Create a solid black caret.
        CreateCaret(hwnd, (HBITMAP) NULL, nWidth, nHeight);

    // Adjust the caret position, in client coordinates.
        SetCaretPos(x, y);

    // Display the caret.
        ShowCaret(hwnd);

        break;
```

To create a caret based on a bitmap, you must specify a bitmap handle when using CreateCaret. You can use a graphics application to create the bitmap and a resource compiler to add the bitmap to your application's resources. Your application can then use the LoadBitmap function to load the bitmap handle. For example, you could replace the **CreateCaret** line in the preceding example with the following lines to create a bitmap caret.

```
// Load the application-defined caret resource.

    hCaret = LoadBitmap(hinst, MAKEINTRESOURCE(120));

// Create a bitmap caret.

    CreateCaret(hwnd, hCaret, 0, 0);
```

Alternatively, you can use the CreateBitmap or CreateDIBitmap function to retrieve the handle of the caret bitmap. For more information about bitmaps, see Bitmaps.

If your application specifies a bitmap handle, CreateCaret ignores the width and height parameters. The bitmap defines the size of the caret.

Hiding a Caret

Whenever your application redraws a screen while processing a message other than WM_PAINT, it must make the caret invisible by using the HideCaret function. When your application is finished drawing, redisplay the caret by using the ShowCaret function. If your application processes the **WM_PAINT** message, it is not necessary to hide and redisplay the caret, because this function does this automatically.

The following code sample shows how to have your application hide the caret while drawing a character on the screen and while processing the WM_CHAR message.

```
HWND hwnd,   // window handle
HDC hdc;     // device context

    case WM_CHAR:
        switch (wParam)
        {
            case 0x08:

             // Process a backspace.
```

```
                break;

            case 0x09:

             // Process a tab.

                break;

            case 0x0D:

             // Process a carriage return.

                break;

            case 0x1B:

             // Process an escape.

                break;

            case 0x0A:

             // Process a linefeed.

                break;

            default:
                // Hide the caret.
                HideCaret(hwnd);

                // Draw the character on the screen.

                hdc = GetDC(hwnd);
                SelectObject(hdc,
                    GetStockObject(SYSTEM_FIXED_FONT));

                TextOut(hdc, x, y, lpszChar, 1);

                ReleaseDC(hwnd, hdc);

                // Display the caret.

                ShowCaret(hwnd);

        }
```

If your application calls the HideCaret function several times without calling ShowCaret, the caret will not be displayed until the application also calls **ShowCaret** the same number of times.

Destroying a Caret

When a window loses the keyboard focus, the system sends the WM_KILLFOCUS message to the window. Your application should destroy the caret while processing this message by using the DestroyCaret function. The following code shows how to destroy a caret in a window that no longer has the keyboard focus.

```
case WM_KILLFOCUS:

// The window is losing the keyboard focus, so destroy the caret.

    DestroyCaret();

    break;
```

Adjusting the Blink Time

In 16-bit Windows, a Windows-based application could call the GetCaretBlinkTime function to save the current blink time, then call the SetCaretBlinkTime function to adjust the blink time during its processing of the WM_SETFOCUS message. The application would restore the saved blink time for the use of other applications by calling **SetCaretBlinkTime** during its processing of the WM_KILLFOCUS message. However, this technique does not work in multithreaded environments. Specifically, the deactivation of one application is not synchronized with the activation of another application, so that if one application hangs, another application can still be activated.

Applications should respect the blink time chosen by the user. The SetCaretBlinkTime function should only be called by an application that allows the user to set the blink time.

Processing Keyboard Input

The following example demonstrates how to use a caret in a simple text editor. The example updates the caret position as the user types printable characters and uses various keys to move through the client area.

```
#define TEXTMATRIX(x, y) *(pTextMatrix + (y * nWindowCharsX) + x)
// Global variables.
HINSTANCE hinst;                // current instance
HBITMAP hCaret;                 // caret bitmap
HDC hdc;                        // device context
PAINTSTRUCT ps;                 // client area paint info
```

```
static char *pTextMatrix = NULL;  // points to text matrix
static int  nCharX,               // width of char. in logical units
            nCharY,               // height of char. in logical units
            nWindowX,             // width of client area
            nWindowY,             // height of client area
            nWindowCharsX,        // width of client area
            nWindowCharsY,        // height of client area
            nCaretPosX,           // x-position of caret
            nCaretPosY;           // y-position of caret
static UINT uOldBlink;            // previous blink rate
int x, y;                         // coordinates for text matrix
TEXTMETRIC tm;                    // font information

LONG APIENTRY MainWndProc(
    HWND hwnd,            // window handle
    UINT message,         // type of message
    UINT wParam,          // additional information
    LONG lParam)          // additional information
{

    switch (message)
    {
        case WM_CREATE:
        // Select a fixed-width system font, and get its text metrics.

            hdc = GetDC(hwnd);
            SelectObject(hdc,
                GetStockObject(SYSTEM_FIXED_FONT));
            GetTextMetrics(hdc, &tm);
            ReleaseDC(hwnd, hdc);

        // Save the avg. width and height of characters.

            nCharX = tm.tmAveCharWidth;
            nCharY = tm.tmHeight;

            return 0;

        case WM_SIZE:
        // Determine the width of the client area, in pixels
        // and in number of characters.

            nWindowX = LOWORD(lParam);
            nWindowCharsX = max(1, nWindowX/nCharX);

            // Determine the height of the client area, in
            // pixels and in number of characters.

            nWindowY = HIWORD(lParam);
            nWindowCharsY = max(1, nWindowY/nCharY);

            // Clear the buffer that holds the text input.

            if (pTextMatrix != NULL)
                free(pTextMatrix);
```

```
            // If there is enough memory, allocate space for the
            // text input buffer.

            pTextMatrix = malloc(nWindowCharsX * nWindowCharsY);

            if (pTextMatrix == NULL)
                ErrorHandler("Not enough memory.");
            else
                for (y = 0; y < nWindowCharsY; y++)
                    for (x = 0; x < nWindowCharsX; x++)
                        TEXTMATRIX(x, y) = ' ';

            // Move the caret to the origin.

            SetCaretPos(0, 0);

            return 0;

        case WM_KEYDOWN:
            switch (wParam)
            {
                case VK_HOME:       // Home
                    nCaretPosX = 0;
                    break;

                case VK_END:        // End
                    nCaretPosX = nWindowCharsX - 1;
                    break;

                case VK_PRIOR:      // Page Up
                    nCaretPosY = 0;
                    break;

                case VK_NEXT:       // Page Down
                    nCaretPosY = nWindowCharsY -1;
                    break;

                case VK_LEFT:       // Left arrow
                    nCaretPosX = max(nCaretPosX - 1, 0);
                    break;

                case VK_RIGHT:      // Right arrow
                    nCaretPosX = min(nCaretPosX + 1,
                        nWindowCharsX - 1);
                    break;

                case VK_UP:         // Up arrow
                    nCaretPosY = max(nCaretPosY - 1, 0);
                    break;

                case VK_DOWN:       // Down arrow
                    nCaretPosY = min(nCaretPosY + 1,
                        nWindowCharsY - 1);
                    break;
```

```
            case VK_DELETE:     // Delete

            // Move all the characters that followed the
            // deleted character (on the same line) one
            // space back (to the left) in the matrix.

                for (x = nCaretPosX; x < nWindowCharsX; x++)
                    TEXTMATRIX(x, nCaretPosY) =
                        TEXTMATRIX(x + 1, nCaretPosY);

                // Replace the last character on the
                // line with a space.

                TEXTMATRIX(nWindowCharsX - 1,
                    nCaretPosY) = ' ';

                // The application will draw outside the
                // WM_PAINT message processing, so hide the caret.

                HideCaret(hwnd);

                // Redraw the line, adjusted for the
                // deleted character.

                hdc = GetDC(hwnd);
                SelectObject(hdc,
                    GetStockObject(SYSTEM_FIXED_FONT));

                TextOut(hdc, nCaretPosX * nCharX,
                    nCaretPosY * nCharY,
                    &TEXTMATRIX(nCaretPosX, nCaretPosY),
                    nWindowCharsX - nCaretPosX);

                ReleaseDC(hwnd, hdc);

                // Display the caret.

                ShowCaret(hwnd);

                break;
        }

        // Adjust the caret position based on the
        // virtual-key processing.

        SetCaretPos(nCaretPosX * nCharX,
            nCaretPosY * nCharY);

        return 0;

    case WM_CHAR:
        switch (wParam)
        {
            case 0x08:          // Backspace
```

```
                // Move the caret back one space, and then
                // process this like the DEL key.

                    if (nCaretPosX > 0)
                    {
                        nCaretPosX--;
                        SendMessage(hwnd, WM_KEYDOWN,
                            VK_DELETE, 1L);
                    }
                    break;

                case 0x09:          // Tab
                // Tab stops exist every four spaces, so add
                // spaces until the user hits the next tab.

                    do
                    {
                        SendMessage(hwnd, WM_CHAR, ' ', 1L);
                    } while (nCaretPosX % 4 != 0);
                    break;

                case 0x0D:          // Carriage return
                // Go to the beginning of the next line.
                // The bottom line wraps around to the top.

                    nCaretPosX = 0;

                    if (++nCaretPosY == nWindowCharsY)
                        nCaretPosY = 0;
                    break;

                case 0x1B:        // Escape
                case 0x0A:        // Linefeed
                    MessageBeep((UINT) -1);
                    break;

                default:
                // Add the character to the text buffer.

                    TEXTMATRIX(nCaretPosX, nCaretPosY) =
                        (char) wParam;

                // The application will draw outside the
                // WM_PAINT message processing, so hide the caret.

                    HideCaret(hwnd);

                // Draw the character on the screen.

                    hdc = GetDC(hwnd);
                    SelectObject(hdc,
                        GetStockObject(SYSTEM_FIXED_FONT));

                    TextOut(hdc, nCaretPosX * nCharX,
                        nCaretPosY * nCharY,
```

```
                    &TEXTMATRIX(nCaretPosX, nCaretPosY), 1);

                ReleaseDC(hwnd, hdc);

                // Display the caret.

                ShowCaret(hwnd);

                // Prepare to wrap around if you reached the
                // end of the line.

                if (++nCaretPosX == nWindowCharsX)
                {
                    nCaretPosX = 0;
                    if (++nCaretPosY == nWindowCharsY)
                        nCaretPosY = 0;
                }
                break;
        }

        // Adjust the caret position based on the
        // character processing.

        SetCaretPos(nCaretPosX * nCharX,
            nCaretPosY * nCharY);

        return 0;

    case WM_PAINT:
    // Draw all the characters in the buffer, line by line.

        hdc = BeginPaint(hwnd, &ps);

        SelectObject(hdc,
            GetStockObject(SYSTEM_FIXED_FONT));

        for (y = 0; y < nWindowCharsY; y++)
            TextOut(hdc, 0, y * nCharY, &TEXTMATRIX(0, y),
                nWindowCharsX);

        EndPaint(hwnd, &ps);

    case WM_SETFOCUS:
    // The window has the input focus. Load the
    // application-defined caret resource.

        hCaret = LoadBitmap(hinst, MAKEINTRESOURCE(120));

        // Create the caret.

        CreateCaret(hwnd, hCaret, 0, 0);

        // Adjust the caret position.

        SetCaretPos(nCaretPosX * nCharX,
```

```
                nCaretPosY * nCharY);

            // Display the caret.

            ShowCaret(hwnd);

            break;

        case WM_KILLFOCUS:
        // The window is losing the input focus,
        // so destroy the caret.

            DestroyCaret();

            break;

        default:
            return DefWindowProc(hwnd, message, wParam, lParam);

    }

    return NULL;
}
```

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Caret Reference

Article • 04/27/2021

In This Section

- Caret Functions

Feedback

Was this page helpful? Yes No

Caret Functions

Article • 04/27/2021

In This Section

- CreateCaret
- DestroyCaret
- GetCaretBlinkTime
- GetCaretPos
- HideCaret
- SetCaretBlinkTime
- SetCaretPos
- ShowCaret

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

CreateCaret function (winuser.h)

Article05/07/2024

Creates a new shape for the system caret and assigns ownership of the caret to the specified window. The caret shape can be a line, a block, or a bitmap.

Syntax

```
C++

BOOL CreateCaret(
  [in]           HWND    hWnd,
  [in, optional] HBITMAP hBitmap,
  [in]           int     nWidth,
  [in]           int     nHeight
);
```

Parameters

`[in] hWnd`

Type: **HWND**

A handle to the window that owns the caret.

`[in, optional] hBitmap`

Type: **HBITMAP**

A handle to the bitmap that defines the caret shape. If this parameter is **NULL**, the caret is solid. If this parameter is `(HBITMAP) 1`, the caret is gray. If this parameter is a bitmap handle, the caret is the specified bitmap. The bitmap handle must have been created by the CreateBitmap, CreateDIBitmap, or LoadBitmap function. The caret is drawn to the screen via the XOR operation.

If *hBitmap* is a bitmap handle, **CreateCaret** ignores the *nWidth* and *nHeight* parameters; the bitmap defines its own width and height. The application should not delete the *hBitmap* until the caret is destroyed or replaced by another caret.

`[in] nWidth`

Type: **int**

The width of the caret, in logical units. If this parameter is zero, the width is set to the system-defined window border width. If *hBitmap* is a bitmap handle, **CreateCaret** ignores this parameter.

`[in] nHeight`

Type: **int**

The height of the caret, in logical units. If this parameter is zero, the height is set to the system-defined window border height. If *hBitmap* is a bitmap handle, **CreateCaret** ignores this parameter.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The *nWidth* and *nHeight* parameters specify the caret's width and height, in logical units; the exact width and height, in pixels, depend on the window's mapping mode.

CreateCaret automatically destroys the previous caret shape, if any, regardless of the window that owns the caret. The caret is hidden until the application calls the ShowCaret function to make the caret visible.

The system provides one caret per queue. A window should create a caret only when it has the keyboard focus or is active. The window should destroy the caret before losing the keyboard focus or becoming inactive.

DPI Virtualization

This API does not participate in DPI virtualization. The width and height parameters are interpreted as logical sizes in terms of the window in question. The calling thread is not taken into consideration.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-caret-l1-1-0 (introduced in Windows 8)

See also

Carets

Conceptual

CreateBitmap

CreateDIBitmap

DestroyCaret

GetSystemMetrics

HideCaret

LoadBitmap

Other Resources

Reference

ShowCaret

Feedback

Was this page helpful? Yes No

DestroyCaret function (winuser.h)

Article02/22/2024

Destroys the caret's current shape, frees the caret from the window, and removes the caret from the screen.

Syntax

```
C++

BOOL DestroyCaret();
```

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

DestroyCaret destroys the caret only if a window in the current task owns the caret. If a window that is not in the current task owns the caret, **DestroyCaret** does nothing and returns **FALSE**.

The system provides one caret per queue. A window should create a caret only when it has the keyboard focus or is active. The window should destroy the caret before losing the keyboard focus or becoming inactive.

For an example, see Destroying a Caret

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-caret-l1-1-0 (introduced in Windows 8)

See also

Carets

Conceptual

CreateCaret

HideCaret

Reference

ShowCaret

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetCaretBlinkTime function (winuser.h)

Article02/22/2024

Retrieves the time required to invert the caret's pixels. The user can set this value.

Syntax

```
C++

UINT GetCaretBlinkTime();
```

Return value

Type: **UINT**

If the function succeeds, the return value is the blink time, in milliseconds.

A return value of **INFINITE** indicates that the caret does not blink.

A return value is zero indicates that the function has failed. To get extended error information, call GetLastError.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-caret-l1-1-0 (introduced in Windows 8)

See also

Carets

Conceptual

Reference

SetCaretBlinkTime

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetCaretPos function (winuser.h)

Article02/22/2024

Copies the caret's position to the specified POINT structure.

Syntax

```
C++

BOOL GetCaretPos(
  [out] LPPOINT lpPoint
);
```

Parameters

`[out] lpPoint`

Type: **LPPOINT**

A pointer to the POINT structure that is to receive the client coordinates of the caret.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The caret position is always given in the client coordinates of the window that contains the caret.

DPI Virtualization

This API does not participate in DPI virtualization. The returned values are interpreted as logical sizes in terms of the window in question. The calling thread is not taken into consideration.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-caret-l1-1-0 (introduced in Windows 8)

See also

Carets

Conceptual

Other Resources

POINT

Reference

SetCaretPos

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

HideCaret function (winuser.h)

Article02/22/2024

Removes the caret from the screen. Hiding a caret does not destroy its current shape or invalidate the insertion point.

Syntax

```
C++

BOOL HideCaret(
  [in, optional] HWND hWnd
);
```

Parameters

`[in, optional] hWnd`

Type: **HWND**

A handle to the window that owns the caret. If this parameter is **NULL**, **HideCaret** searches the current task for the window that owns the caret.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

HideCaret hides the caret only if the specified window owns the caret. If the specified window does not own the caret, **HideCaret** does nothing and returns **FALSE**.

Hiding is cumulative. If your application calls **HideCaret** five times in a row, it must also call ShowCaret five times before the caret is displayed.

For an example, see Hiding a Caret.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-caret-l1-1-0 (introduced in Windows 8)

See also

Carets

Conceptual

CreateCaret

DestroyCaret

GetCaretPos

Reference

SetCaretPos

ShowCaret

Feedback

Was this page helpful? Yes No

SetCaretBlinkTime function (winuser.h)

Article02/22/2024

Sets the caret blink time to the specified number of milliseconds. The blink time is the elapsed time, in milliseconds, required to invert the caret's pixels.

Syntax

```
C++

BOOL SetCaretBlinkTime(
  [in] UINT uMSeconds
);
```

Parameters

`[in] uMSeconds`

Type: **UINT**

The new blink time, in milliseconds.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The user can set the blink time using the Control Panel. Applications should respect the setting that the user has chosen. The **SetCaretBlinkTime** function should only be used by application that allow the user to set the blink time, such as a Control Panel applet.

If you change the blink time, subsequently activated applications will use the modified blink time, even if you restore the previous blink time when you lose the keyboard focus or become inactive. This is due to the multithreaded environment, where deactivation of

your application is not synchronized with the activation of another application. This feature allows the system to activate another application even if the current application is not responding.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-caret-l1-1-0 (introduced in Windows 8)

See also

Carets

Conceptual

GetCaretBlinkTime

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

SetCaretPos function (winuser.h)

Article10/13/2021

Moves the caret to the specified coordinates. If the window that owns the caret was created with the **CS_OWNDC** class style, then the specified coordinates are subject to the mapping mode of the device context associated with that window.

Syntax

```
C++

BOOL SetCaretPos(
  [in] int X,
  [in] int Y
);
```

Parameters

`[in] X`

Type: **int**

The new x-coordinate of the caret.

`[in] Y`

Type: **int**

The new y-coordinate of the caret.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

SetCaretPos moves the caret whether the caret is hidden.

The system provides one caret per queue. A window should create a caret only when it has the keyboard focus or is active. The window should destroy the caret before losing the keyboard focus or becoming inactive. A window can set the caret position only if it owns the caret.

DPI Virtualization

This API does not participate in DPI virtualization. The provided position is interpreted as logical coordinates in terms of the window associated with the caret. The calling thread is not taken into consideration.

Examples

For an example, see Creating and Displaying a Caret.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-caret-l1-1-0 (introduced in Windows 8)

See also

Carets

Conceptual

GetCaretPos

HideCaret

Reference

ShowCaret

Feedback

ShowCaret function (winuser.h)

Article02/22/2024

Makes the caret visible on the screen at the caret's current position. When the caret becomes visible, it begins flashing automatically.

Syntax

```
C++

BOOL ShowCaret(
  [in, optional] HWND hWnd
);
```

Parameters

`[in, optional] hWnd`

Type: **HWND**

A handle to the window that owns the caret. If this parameter is **NULL**, **ShowCaret** searches the current task for the window that owns the caret.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

ShowCaret shows the caret only if the specified window owns the caret, the caret has a shape, and the caret has not been hidden two or more times in a row. If one or more of these conditions is not met, **ShowCaret** does nothing and returns **FALSE**.

Hiding is cumulative. If your application calls HideCaret five times in a row, it must also call **ShowCaret** five times before the caret reappears.

The system provides one caret per queue. A window should create a caret only when it has the keyboard focus or is active. The window should destroy the caret before losing the keyboard focus or becoming inactive.

Examples

For an example, see Creating and Displaying a Caret.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-caret-l1-1-0 (introduced in Windows 8)

See also

Carets

Conceptual

CreateCaret

DestroyCaret

GetCaretPos

HideCaret

Reference

SetCaretPos

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Cursors

Article • 08/23/2019

A cursor is a small picture whose location on the screen is controlled by a pointing device, such as a mouse, pen, or trackball. In the remainder of this overview, the term mouse refers to any pointing device.

When the user moves the mouse, the system moves the cursor accordingly. The cursor functions enable applications to create, load, display, animate, move, confine, and destroy cursors.

In This Section

Name	Description
About Cursors	Discusses the standard cursors.
Using Cursors	Discusses how to perform tasks related to cursors.
Cursor Reference	Contains the API reference.

Cursor Functions

Name	Description
ClipCursor	Confines the cursor to a rectangular area on the screen. If a subsequent cursor position (set by the SetCursorPos function or the mouse) lies outside the rectangle, the system automatically adjusts the position to keep the cursor inside the rectangular area.
CopyCursor	Copies the specified cursor.
CreateCursor	Creates a cursor having the specified size, bit patterns, and hot spot.
DestroyCursor	Destroys a cursor and frees any memory the cursor occupied. Do not use this function to destroy a shared cursor.
GetClipCursor	Retrieves the screen coordinates of the rectangular area to which the cursor is confined.
GetCursor	Retrieves a handle to the current cursor.
GetCursorInfo	Retrieves information about the global cursor.
GetCursorPos	Retrieves the cursor's position, in screen coordinates.

Name	Description
GetPhysicalCursorPos	Retrieves the position of the cursor in physical coordinates.
LoadCursor	Loads the specified cursor resource from the executable (.EXE) file associated with an application instance.
LoadCursorFromFile	Creates a cursor based on data contained in a file.
SetCursor	Sets the cursor shape.
SetCursorPos	Moves the cursor to the specified screen coordinates. If the new coordinates are not within the screen rectangle set by the most recent ClipCursor function call, the system automatically adjusts the coordinates so that the cursor stays within the rectangle.
SetPhysicalCursorPos	Sets the position of the cursor in physical coordinates.
SetSystemCursor	Enables an application to customize the system cursors. It replaces the contents of the system cursor specified by the *id* parameter with the contents of the cursor specified by the *hcur* parameter and then destroys *hcur*.
ShowCursor	Displays or hides the cursor.

Cursor Notifications

Name	Description
WM_SETCURSOR	Sent to a window if the mouse causes the cursor to move within a window and mouse input is not captured.

Cursor Structures

Name	Description
CURSORINFO	Contains global cursor information.

About Cursors

Article • 03/13/2024

Windows provides a set of **standard cursors** that can be used by applications. The following cursor identifiers are defined in WinUser.h:

Expand table

Value	Meaning
IDC_ARROW MAKEINTRESOURCE(32512)	 Normal select
IDC_IBEAM MAKEINTRESOURCE(32513)	 Text select
IDC_WAIT MAKEINTRESOURCE(32514)	 Busy
IDC_CROSS MAKEINTRESOURCE(32515)	 Precision select
IDC_UPARROW MAKEINTRESOURCE(32516)	 Alternate select
IDC_SIZENWSE MAKEINTRESOURCE(32642)	 Diagonal resize 1
IDC_SIZENESW MAKEINTRESOURCE(32643)	 Diagonal resize 2
IDC_SIZEWE MAKEINTRESOURCE(32644)	 Horizontal resize
IDC_SIZENS MAKEINTRESOURCE(32645)	 Vertical resize
IDC_SIZEALL MAKEINTRESOURCE(32646)	 Move
IDC_NO MAKEINTRESOURCE(32648)	 Unavailable
IDC_HAND MAKEINTRESOURCE(32649)	 Link select
IDC_APPSTARTING MAKEINTRESOURCE(32650)	 Working in background

Value	Meaning
IDC_HELP MAKEINTRESOURCE(32651)	 Help select
IDC_PIN MAKEINTRESOURCE(32671)	 Location select
IDC_PERSON MAKEINTRESOURCE(32672)	 Person select

A number of additional cursors are also available that do not have identifiers defined in WinUser.h (or are considered obsolete):

Expand table

Value	Meaning
MAKEINTRESOURCE(32631)	 A pen cursor.
MAKEINTRESOURCE(32652)	 A scrolling cursor with arrows pointing north and south.
MAKEINTRESOURCE(32653)	 A scrolling cursor with arrows pointing west and east.
MAKEINTRESOURCE(32654)	 A scrolling cursor with arrows pointing north, south, east, and west.
MAKEINTRESOURCE(32655)	 A scrolling cursor with an arrow pointing north.
MAKEINTRESOURCE(32656)	 A scrolling cursor with an arrow pointing south.
MAKEINTRESOURCE(32657)	 A scrolling cursor with an arrow pointing west.
MAKEINTRESOURCE(32658)	 A scrolling cursor with an arrow pointing east.
MAKEINTRESOURCE(32659)	 A scrolling cursor with arrows pointing north and west.
MAKEINTRESOURCE(32660)	 A scrolling cursor with arrows pointing north and east.
MAKEINTRESOURCE(32661)	 A scrolling cursor with arrows pointing south and west.
MAKEINTRESOURCE(32662)	 A scrolling cursor with arrows pointing south and east.

Value	Meaning
MAKEINTRESOURCE(32663)	 An arrow cd cursor.

See Guidelines for information on using standard cursors.

Each standard cursor has a corresponding default image associated with it. The user or an application can replace the default image associated with any standard cursor at any time. An application replaces a default image by using the **SetSystemCursor** function.

An application can use the **GetIconInfo** function to retrieve the current image for a cursor and can draw the cursor by using the **DrawIconEx** function.

Custom cursors are designed for use in a specific application and can be any design the developer defines. The following illustration shows several custom cursors.

Cursors can be either monochrome or color, and either static or animated. The type of cursor used on a particular computer system depends on the system's display. Old displays such as VGA do not support color or animated cursors. New displays, whose display drivers use the device-independent bitmap (DIB) engine, do support them.

Cursors and icons are similar and can be used interchangeably in many situations. The only difference between them is that an image specified as a cursor must be in the format that the display can support. For example, a cursor must be monochrome for a VGA display.

This overview provides information on the following topics:

- The Hot Spot
- The Mouse and the Cursor
- Cursor Creation
- Cursor Location and Appearance
- Cursor Confinement
- Cursor Destruction
- Cursor Duplication
- The Window Class Cursor

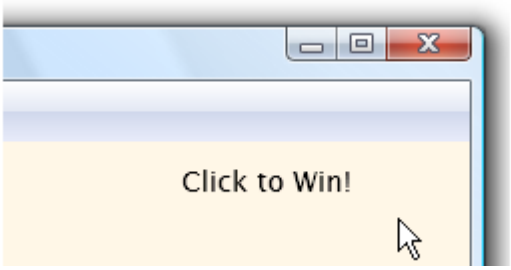
The Hot Spot

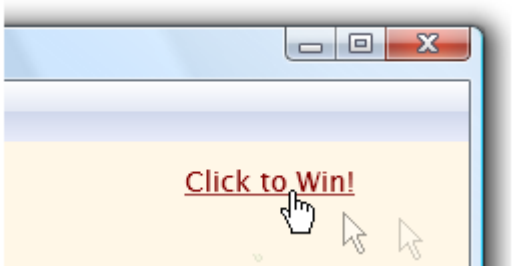
In the cursor, a pixel called the *hot spot* marks the exact screen location that is affected by a mouse event, such as clicking a mouse button. Typically, the hot spot is the focal point of the cursor. The system tracks and recognizes this point as the position of the cursor. For example, typical hot spots are the pixel at the tip of an arrow-shaped cursor and the pixel in the middle of a crosshair-shaped cursor. The following images shows two cursors from a drawing program, in which hot spots are associated with the tip of the brush and the crosshair of the paint can.

When a mouse input event occurs, the mouse driver translates the event into an appropriate mouse message that includes the coordinates of the hot spot. The system sends the mouse message to the window that contains the hot spot or to the window that is capturing mouse input. For more information, see Mouse Input.

The Mouse and the Cursor

The system reflects the movement of the mouse by moving the cursor on the screen accordingly. As the cursor moves over different parts of windows or into different windows, the system (or an application) changes the appearance of the cursor. For example, when the cursor crosses over a hyperlink, the system changes the cursor from an arrow to a hand.





If the system does not have a mouse, the system displays and moves the cursor only when the user chooses certain system commands, such as those used to size or move a window. To provide the user with a method of displaying and moving the cursor when a mouse is not available, an application can use the cursor functions to simulate mouse movement. Given this simulation capability, the user can use the arrow keys to move the cursor.

Cursor Creation

Because standard cursors are predefined, it is not necessary to create them. To use a standard cursor, an application retrieves a cursor handle by using the **LoadCursor** or **LoadImage** function. A *cursor handle* is a unique value of the **HCURSOR** type that identifies a standard or custom cursor.

To create a custom cursor for an application, you typically use a graphics application and include the cursor as a resource in the application's resource-definition file. At run time, call **LoadCursor** to retrieve the cursor handle. Cursor resources contain data for several different display devices. The **LoadCursor** function automatically selects the most appropriate data for the current display device. To load a cursor directly from a .CUR or .ANI file, use the **LoadCursorFromFile** function.

You can also create a custom cursor at run time by using the **CreateIconIndirect** function, which creates a cursor based on the content of an **ICONINFO** structure. The **GetIconInfo** function fills this structure with hot spot coordinates and information concerning the associated mask and color.

Applications should implement custom cursors as resources and use **LoadCursor**, **LoadCursorFromFile**, or **LoadImage** rather than create the cursor at run time. Using cursor resources avoids device dependence, simplifies localization, and enables applications to share cursor designs.

The **CreateIconFromResourceEx** function enables an application to create icons and cursors based on resource data. **CreateIconFromResourceEx** creates a cursor based on binary resource data from other executable (.exe) files or DLLs. It must be preceded by calls to the **LookupIconIdFromDirectoryEx** function, as well as several resource functions. **LookupIconIdFromDirectoryEx** identifies the most appropriate cursor data for the current display device. For more information about resource functions, see Resources.

Cursor Location and Appearance

The system automatically displays a cursor for the mouse and updates its position on the screen. You can obtain current screen coordinates of the cursor and move the cursor to any location on the screen by using the **GetCursorPos** and **SetCursorPos** functions, respectively.

You can also retrieve the handle to the current cursor by using the **GetCursor** function, and you can set the cursor by using the **SetCursor** function. After you call **SetCursor**, the appearance of the cursor does not change until either the mouse moves, the cursor is explicitly set to a different cursor, or a system command is executed.

When the user moves the mouse, the system redraws the cursor at the new location. The system automatically redraws the cursor design associated with the window to which the cursor is pointing.

You can hide and redisplay the cursor, without changing the cursor design, by using the **ShowCursor** function. This function uses an internal counter to determine when to hide or display the cursor. An attempt to show the cursor increments the counter; an attempt to hide the cursor decrements the counter. The cursor is visible only if this counter is greater than or equal to zero.

The **GetCursorInfo** function gets the following information for the global cursor: whether the cursor is hidden or shown, the handle to the cursor, and the coordinates of the cursor.

Cursor Confinement

You can confine the cursor to a rectangular area on the screen by using the **ClipCursor** function. This is useful for when the user must respond to a certain event within the confined area of the rectangle. For example, you might use **ClipCursor** to confine the cursor to a modal dialog box, preventing the user from interacting with other windows until the dialog box is closed.

The **GetClipCursor** function retrieves the screen coordinates of the rectangular area to which the cursor is temporarily confined. When it is necessary to confine the cursor, you can also use this function to save the coordinates of the original area in which the cursor can move. Then, you can restore the cursor to the original area when the new confinement is no longer necessary.

Cursor Destruction

You can destroy the cursor handle and free the memory the cursor used by calling the **DestroyCursor** function. However, this function has no effect on a shared cursor. A shared cursor is valid as long as the module from which it was loaded remains in memory. The following functions obtain a shared cursor:

- **LoadCursor**
- **LoadCursorFromFile**
- **LoadImage** (if you use the **LR_SHARED** flag)
- **CopyImage** (if you use the **LR_COPYRETURNORG** flag and the *hImage* is a shared cursor)

When you no longer need a cursor you created by using the **CreateIconIndirect** function, you should destroy the cursor. The **DestroyIcon** function destroys the cursor handle and frees any memory the cursor used. Use this function only on cursors that were created with **CreateIconIndirect**.

Cursor Duplication

The **CopyCursor** function copies a cursor handle. This enables application or DLL code to retrieve the handle to a cursor owned by another module. Then, if the other module is freed, the module that copied the cursor can still use the cursor design.

For information on how to add, remove, or replace cursor resources in executable files, see Resources.

The Window Class Cursor

When you register a window class, using the **RegisterClass** function, you can assign it a default cursor, known as the *class cursor*. After the application registers the window class, each window of that class has the specified class cursor.

To override the class cursor, process the **WM_SETCURSOR** message. You can also replace a class cursor by using the **SetClassLong** function. This function changes the default window settings for all windows of a specified class. For more information, see Class Cursor.

Using Cursors

Article • 03/28/2023

This section discusses the following topics.

- Creating a Cursor
- Getting a Cursor size
- Displaying a Cursor
- Confining a Cursor
- Using Cursor Functions to Create a Mousetrap
- Using the Keyboard to Move the Cursor

Creating a Cursor

The following example creates two cursor handles: one for the standard hourglass cursor and one for a custom cursor included as a resource in the application's resource-definition file.

C++

```
HINSTANCE hinst;            // handle to current instance
HCURSOR hCurs1, hCurs2;     // cursor handles

// Create a standard hourglass cursor.

hCurs1 = LoadCursor(NULL, IDC_WAIT);

// Create a custom cursor based on a resource.

hCurs2 = LoadCursor(hinst, MAKEINTRESOURCE(240));
```

Applications should implement custom cursors as resources and use **LoadCursor**, **LoadCursorFromFile**, or **LoadImage** rather than create the cursor at run time. Using cursor resources avoids device dependence, simplifies localization, and enables applications to share cursor designs.

The following example uses the **CreateCursor** function to create a custom monochrome cursor at run time. The example is included here to illustrate how the system interprets cursor masks.

C++

```
HINSTANCE hinst;            // handle to current instance
HCURSOR hCurs1, hCurs2;     // cursor handles
```

```
HCURSOR hCurs3;             // cursor handle

// Yin-shaped cursor AND mask

BYTE ANDmaskCursor[] =
{
    0xFF, 0xFC, 0x3F, 0xFF,   // line 1
    0xFF, 0xC0, 0x1F, 0xFF,   // line 2
    0xFF, 0x00, 0x3F, 0xFF,   // line 3
    0xFE, 0x00, 0xFF, 0xFF,   // line 4

    0xF7, 0x01, 0xFF, 0xFF,   // line 5
    0xF0, 0x03, 0xFF, 0xFF,   // line 6
    0xF0, 0x03, 0xFF, 0xFF,   // line 7
    0xE0, 0x07, 0xFF, 0xFF,   // line 8

    0xC0, 0x07, 0xFF, 0xFF,   // line 9
    0xC0, 0x0F, 0xFF, 0xFF,   // line 10
    0x80, 0x0F, 0xFF, 0xFF,   // line 11
    0x80, 0x0F, 0xFF, 0xFF,   // line 12

    0x80, 0x07, 0xFF, 0xFF,   // line 13
    0x00, 0x07, 0xFF, 0xFF,   // line 14
    0x00, 0x03, 0xFF, 0xFF,   // line 15
    0x00, 0x00, 0xFF, 0xFF,   // line 16

    0x00, 0x00, 0x7F, 0xFF,   // line 17
    0x00, 0x00, 0x1F, 0xFF,   // line 18
    0x00, 0x00, 0x0F, 0xFF,   // line 19
    0x80, 0x00, 0x0F, 0xFF,   // line 20

    0x80, 0x00, 0x07, 0xFF,   // line 21
    0x80, 0x00, 0x07, 0xFF,   // line 22
    0xC0, 0x00, 0x07, 0xFF,   // line 23
    0xC0, 0x00, 0x0F, 0xFF,   // line 24

    0xE0, 0x00, 0x0F, 0xFF,   // line 25
    0xF0, 0x00, 0x1F, 0xFF,   // line 26
    0xF0, 0x00, 0x1F, 0xFF,   // line 27
    0xF8, 0x00, 0x3F, 0xFF,   // line 28

    0xFE, 0x00, 0x7F, 0xFF,   // line 29
    0xFF, 0x00, 0xFF, 0xFF,   // line 30
    0xFF, 0xC3, 0xFF, 0xFF,   // line 31
    0xFF, 0xFF, 0xFF, 0xFF    // line 32
};

// Yin-shaped cursor XOR mask

BYTE XORmaskCursor[] =
{
    0x00, 0x00, 0x00, 0x00,   // line 1
    0x00, 0x03, 0xC0, 0x00,   // line 2
    0x00, 0x3F, 0x00, 0x00,   // line 3
```

```
    0x00, 0xFE, 0x00, 0x00,   // line 4

    0x0E, 0xFC, 0x00, 0x00,   // line 5
    0x07, 0xF8, 0x00, 0x00,   // line 6
    0x07, 0xF8, 0x00, 0x00,   // line 7
    0x0F, 0xF0, 0x00, 0x00,   // line 8

    0x1F, 0xF0, 0x00, 0x00,   // line 9
    0x1F, 0xE0, 0x00, 0x00,   // line 10
    0x3F, 0xE0, 0x00, 0x00,   // line 11
    0x3F, 0xE0, 0x00, 0x00,   // line 12

    0x3F, 0xF0, 0x00, 0x00,   // line 13
    0x7F, 0xF0, 0x00, 0x00,   // line 14
    0x7F, 0xF8, 0x00, 0x00,   // line 15
    0x7F, 0xFC, 0x00, 0x00,   // line 16

    0x7F, 0xFF, 0x00, 0x00,   // line 17
    0x7F, 0xFF, 0x80, 0x00,   // line 18
    0x7F, 0xFF, 0xE0, 0x00,   // line 19
    0x3F, 0xFF, 0xE0, 0x00,   // line 20

    0x3F, 0xC7, 0xF0, 0x00,   // line 21
    0x3F, 0x83, 0xF0, 0x00,   // line 22
    0x1F, 0x83, 0xF0, 0x00,   // line 23
    0x1F, 0x83, 0xE0, 0x00,   // line 24

    0x0F, 0xC7, 0xE0, 0x00,   // line 25
    0x07, 0xFF, 0xC0, 0x00,   // line 26
    0x07, 0xFF, 0xC0, 0x00,   // line 27
    0x01, 0xFF, 0x80, 0x00,   // line 28

    0x00, 0xFF, 0x00, 0x00,   // line 29
    0x00, 0x3C, 0x00, 0x00,   // line 30
    0x00, 0x00, 0x00, 0x00,   // line 31
    0x00, 0x00, 0x00, 0x00    // line 32
};

// Create a custom cursor at run time.

hCurs3 = CreateCursor( hinst,   // app. instance
             19,                // horizontal position of hot spot
             2,                 // vertical position of hot spot
             32,                // cursor width
             32,                // cursor height
             ANDmaskCursor,     // AND mask
             XORmaskCursor );   // XOR mask
```

To create the cursor, CreateCursor applies the following truth table to the **AND** and **XOR** masks.

AND mask	XOR mask	Display
0	0	Black
0	1	White
1	0	Screen
1	1	Reverse screen

For more information, see Bitmaps.

Follow these steps to create an alpha blended cursor or icon at run time:

- Complete a **BITMAPV5HEADER** structure, as in the code example following these steps, to define a 32 bits per pixel (BPP) alpha blended DIB.
- Call the **CreateDIBSection** function to create a DIB section based on the **BITMAPV5HEADER** structure that you completed.
- Use the bitmap and alpha information that you want for your alpha blended cursor or icon to complete the DIB section.
- Complete an **ICONINFO** structure.
- Place an empty monochrome bitmap in the **hbmMask** field, and then place the alpha blended DIB section in the **hbmColor** field.
- Call the **CreateIconIndirect** function to create the alpha blended cursor or icon.

The following code demonstrates how to create an alpha blended cursor. You can use the same code to create an alpha blended icon by changing the **fIcon** member of the **ICONINFO** structure to **TRUE**:

C++

```
HCURSOR CreateAlphaCursor(void)
{
    HDC hMemDC;
    DWORD dwWidth, dwHeight;
    BITMAPV5HEADER bi;
    HBITMAP hBitmap, hOldBitmap;
    void *lpBits;
    DWORD x,y;
    HCURSOR hAlphaCursor = NULL;

    dwWidth  = 32;  // width of cursor
    dwHeight = 32;  // height of cursor

    ZeroMemory(&bi,sizeof(BITMAPV5HEADER));
    bi.bV5Size           = sizeof(BITMAPV5HEADER);
    bi.bV5Width           = dwWidth;
    bi.bV5Height          = dwHeight;
    bi.bV5Planes = 1;
```

```
    bi.bV5BitCount = 32;
    bi.bV5Compression = BI_BITFIELDS;
    // The following mask specification specifies a supported 32 BPP
    // alpha format for Windows XP.
    bi.bV5RedMask   =  0x00FF0000;
    bi.bV5GreenMask =  0x0000FF00;
    bi.bV5BlueMask  =  0x000000FF;
    bi.bV5AlphaMask =  0xFF000000;

    HDC hdc;
    hdc = GetDC(NULL);

    // Create the DIB section with an alpha channel.
    hBitmap = CreateDIBSection(hdc, (BITMAPINFO *)&bi, DIB_RGB_COLORS,
        (void **)&lpBits, NULL, (DWORD)0);

    hMemDC = CreateCompatibleDC(hdc);
    ReleaseDC(NULL,hdc);

    // Draw something on the DIB section.
    hOldBitmap = (HBITMAP)SelectObject(hMemDC, hBitmap);
    PatBlt(hMemDC,0,0,dwWidth,dwHeight,WHITENESS);
    SetTextColor(hMemDC,RGB(0,0,0));
    SetBkMode(hMemDC,TRANSPARENT);
    TextOut(hMemDC,0,9,"rgba",4);
    SelectObject(hMemDC, hOldBitmap);
    DeleteDC(hMemDC);

    // Create an empty mask bitmap.
    HBITMAP hMonoBitmap = CreateBitmap(dwWidth,dwHeight,1,1,NULL);

    // Set the alpha values for each pixel in the cursor so that
    // the complete cursor is semi-transparent.
    DWORD *lpdwPixel;
    lpdwPixel = (DWORD *)lpBits;
    for (x=0;x<dwWidth;x++)
       for (y=0;y<dwHeight;y++)
       {
           // Clear the alpha bits
           *lpdwPixel &= 0x00FFFFFF;
           // Set the alpha bits to 0x9F (semi-transparent)
           *lpdwPixel |= 0x9F000000;
           lpdwPixel++;
       }

    ICONINFO ii;
    ii.fIcon = FALSE;  // Change fIcon to TRUE to create an alpha icon
    ii.xHotspot = 0;
    ii.yHotspot = 0;
    ii.hbmMask = hMonoBitmap;
    ii.hbmColor = hBitmap;

    // Create the alpha cursor with the alpha DIB section.
    hAlphaCursor = CreateIconIndirect(&ii);
```

```
    DeleteObject(hBitmap);
    DeleteObject(hMonoBitmap);

    return hAlphaCursor;
}
```

Before closing, you must use the **DestroyCursor** function to destroy any cursors you created with **CreateCursor** or **CreateIconIndirect**. It is not necessary to destroy cursors created by other functions.

Getting a Cursor size

See Getting the Icon size.

Displaying a Cursor

The system automatically displays the class cursor (the cursor associated with the window to which the cursor is pointing). You can assign a class cursor while registering a window class. The following example illustrates this by assigning a cursor handle to the **hCursor** member of the **WNDCLASS** structure identified by the *wc* parameter.

C++

```
WNDCLASS  wc;

// Fill the window class structure with parameters that
// describe the main window.

wc.style = NULL;                        // class style(s)
wc.lpfnWndProc = (WNDPROC) MainWndProc; // window procedure
wc.cbClsExtra = 0;           // no per-class extra data
wc.cbWndExtra = 0;           // no per-window extra data
wc.hInstance = hinst;        // application that owns the class
wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);     // class icon
wc.hCursor = LoadCursor(hinst, MAKEINTRESOURCE(230)); // class cursor
wc.hbrBackground = GetStockObject(WHITE_BRUSH); // class background
wc.lpszMenuName =  "GenericMenu";               // class menu
wc.lpszClassName = "GenericWClass"              // class name

// Register the window class.

return RegisterClass(&wc);
```

When the window class is registered, the cursor identified by 230 in the application's resource-definition file is the default cursor for all windows based on the class.

Your application can change the design of the cursor by using the SetCursor function and specifying a different cursor handle. However, when the cursor moves, the system redraws the class cursor at the new location. To prevent the class cursor from being redrawn, you must process the WM_SETCURSOR message. Each time the cursor moves and mouse input is not captured, the system sends this message to the window in which the cursor is moving.

You can specify different cursors for different conditions while processing WM_SETCURSOR. For example, the following example shows how to display the cursor whenever the cursor moves over the icon of a minimized application.

```
C++
```

```
case WM_SETCURSOR:

    // If the window is minimized, draw the hCurs3 cursor.
    // If the window is not minimized, draw the default
    // cursor (class cursor).

    if (IsIconic(hwnd))
    {
        SetCursor(hCurs3);
        break;
    }
```

When the window is not minimized, the system displays the class cursor.

You can replace a class cursor by using the SetClassLong function. This function changes the default window settings for all windows of a specified class. The following example replaces the existing class cursor with the `hCurs2` cursor.

```
C++
```

```
// Change the cursor for window class represented by hwnd.

SetClassLongPtr(hwnd,    // window handle
    GCLP_HCURSOR,        // change cursor
    (LONG_PTR) hCurs2);  // new cursor
```

For more information, see Window Classes and Mouse Input.

Confining a Cursor

The following example confines the cursor to the application's window and then restores the cursor to its previous window. The example uses the GetClipCursor function

to record the area in which the cursor can move and the ClipCursor function to confine and restore the cursor.

```
C++
```

```
RECT rcClip;           // new area for ClipCursor
RECT rcOldClip;        // previous area for ClipCursor

// Record the area in which the cursor can move.

GetClipCursor(&rcOldClip);

// Get the dimensions of the application's window.

GetWindowRect(hwnd, &rcClip);

// Confine the cursor to the application's window.

ClipCursor(&rcClip);

   //
   // Process input from the confined cursor.
   //

// Restore the cursor to its previous area.

ClipCursor(&rcOldClip);
```

Because there is only one cursor at a time available in the system, an application that confines the cursor must restore the cursor before relinquishing control to another window.

Using Cursor Functions to Create a Mousetrap

The following example uses the SetCursorPos, GetCursorPos, CreateCursor, LoadCursor, and SetCursor functions to create a simple mousetrap. It also uses cursor and timer functions to monitor the cursor's position every 10 seconds. If the cursor position has not changed in the last 10 seconds and the application's main window is minimized, the application changes the cursor and moves it to the mousetrap icon.

An example for a similar mousetrap is included in Icons. It uses the LoadCursor and LoadIcon functions instead of the more device-dependent CreateCursor and CreateIcon functions.

```
C++
```

```
HICON hIcon1;               // icon handles
POINT ptOld;                // previous cursor location
HCURSOR hCurs1;             // cursor handle


// The following cursor masks are defined in a code
// example that appears earlier in this section.

// Yin-shaped cursor AND and XOR masks

BYTE ANDmaskCursor[] = ...
BYTE XORmaskCursor[] = ...

// Yang-shaped icon AND mask

BYTE ANDmaskIcon[] = {0xFF, 0xFF, 0xFF, 0xFF,  // line 1
                      0xFF, 0xFF, 0xC3, 0xFF,  // line 2
                      0xFF, 0xFF, 0x00, 0xFF,  // line 3
                      0xFF, 0xFE, 0x00, 0x7F,  // line 4

                      0xFF, 0xFC, 0x00, 0x1F,  // line 5
                      0xFF, 0xF8, 0x00, 0x0F,  // line 6
                      0xFF, 0xF8, 0x00, 0x0F,  // line 7
                      0xFF, 0xF0, 0x00, 0x07,  // line 8

                      0xFF, 0xF0, 0x00, 0x03,  // line 9
                      0xFF, 0xE0, 0x00, 0x03,  // line 10
                      0xFF, 0xE0, 0x00, 0x01,  // line 11
                      0xFF, 0xE0, 0x00, 0x01,  // line 12

                      0xFF, 0xF0, 0x00, 0x01,  // line 13
                      0xFF, 0xF0, 0x00, 0x00,  // line 14
                      0xFF, 0xF8, 0x00, 0x00,  // line 15
                      0xFF, 0xFC, 0x00, 0x00,  // line 16

                      0xFF, 0xFF, 0x00, 0x00,  // line 17
                      0xFF, 0xFF, 0x80, 0x00,  // line 18
                      0xFF, 0xFF, 0xE0, 0x00,  // line 19
                      0xFF, 0xFF, 0xE0, 0x01,  // line 20

                      0xFF, 0xFF, 0xF0, 0x01,  // line 21
                      0xFF, 0xFF, 0xF0, 0x01,  // line 22
                      0xFF, 0xFF, 0xF0, 0x03,  // line 23
                      0xFF, 0xFF, 0xE0, 0x03,  // line 24

                      0xFF, 0xFF, 0xE0, 0x07,  // line 25
                      0xFF, 0xFF, 0xC0, 0x0F,  // line 26
                      0xFF, 0xFF, 0xC0, 0x0F,  // line 27
                      0xFF, 0xFF, 0x80, 0x1F,  // line 28

                      0xFF, 0xFF, 0x00, 0x7F,  // line 29
                      0xFF, 0xFC, 0x00, 0xFF,  // line 30
                      0xFF, 0xF8, 0x03, 0xFF,  // line 31
                      0xFF, 0xFC, 0x3F, 0xFF}; // line 32
```

```
// Yang-shaped icon XOR mask

BYTE XORmaskIcon[] = {0x00, 0x00, 0x00, 0x00,  // line 1
                      0x00, 0x00, 0x00, 0x00,  // line 2
                      0x00, 0x00, 0x00, 0x00,  // line 3
                      0x00, 0x00, 0x00, 0x00,  // line 4

                      0x00, 0x00, 0x00, 0x00,  // line 5
                      0x00, 0x00, 0x00, 0x00,  // line 6
                      0x00, 0x00, 0x00, 0x00,  // line 7
                      0x00, 0x00, 0x38, 0x00,  // line 8

                      0x00, 0x00, 0x7C, 0x00,  // line 9
                      0x00, 0x00, 0x7C, 0x00,  // line 10
                      0x00, 0x00, 0x7C, 0x00,  // line 11
                      0x00, 0x00, 0x38, 0x00,  // line 12

                      0x00, 0x00, 0x00, 0x00,  // line 13
                      0x00, 0x00, 0x00, 0x00,  // line 14
                      0x00, 0x00, 0x00, 0x00,  // line 15
                      0x00, 0x00, 0x00, 0x00,  // line 16

                      0x00, 0x00, 0x00, 0x00,  // line 17
                      0x00, 0x00, 0x00, 0x00,  // line 18
                      0x00, 0x00, 0x00, 0x00,  // line 19
                      0x00, 0x00, 0x00, 0x00,  // line 20

                      0x00, 0x00, 0x00, 0x00,  // line 21
                      0x00, 0x00, 0x00, 0x00,  // line 22
                      0x00, 0x00, 0x00, 0x00,  // line 23
                      0x00, 0x00, 0x00, 0x00,  // line 24

                      0x00, 0x00, 0x00, 0x00,  // line 25
                      0x00, 0x00, 0x00, 0x00,  // line 26
                      0x00, 0x00, 0x00, 0x00,  // line 27
                      0x00, 0x00, 0x00, 0x00,  // line 28

                      0x00, 0x00, 0x00, 0x00,  // line 29
                      0x00, 0x00, 0x00, 0x00,  // line 30
                      0x00, 0x00, 0x00, 0x00,  // line 31
                      0x00, 0x00, 0x00, 0x00}; // line 32

hIcon1 = CreateIcon(hinst, // handle to app. instance
             32,           // icon width
             32,           // icon height
             1,            // number of XOR planes
             1,            // number of bits per pixel
             ANDmaskIcon,  // AND mask
             XORmaskIcon); // XOR mask

hCurs1 = CreateCursor(hinst, // handle to app. instance
             19,             // horizontal position of hot spot
             2,              // vertical position of hot spot
             32,             // cursor width
```

```
             32,             // cursor height
             ANDmaskCursor,  // AND mask
             XORmaskCursor); // XOR mask

// Fill in the window class structure.

WNDCLASS  wc;

wc.hIcon = hIcon1;                        // class icon
wc.hCursor = LoadCursor(NULL, IDC_ARROW); // class cursor

//
// Register the window class and perform
// other application initialization.
//

// Set a timer for the mousetrap.

GetCursorPos(&ptOld);

SetTimer(hwnd, IDT_CURSOR, 10000, (TIMERPROC) NULL);

LONG APIENTRY MainWndProc(
    HWND hwnd,          // window handle
    UINT message,       // type of message
    UINT wParam,        // additional information
    LONG lParam)        // additional information
{

    HDC hdc;            // handle to device context
    POINT pt;           // current cursor location
    RECT rc;            // minimized window location

    switch (message)
    {
        //
        // Process other messages.
        //
        case WM_TIMER:
        // If the window is minimized, compare the
        // current cursor position with the one 10
        // seconds before. If the cursor position has
        // not changed, move the cursor to the icon.

            if (IsIconic(hwnd))
            {
                GetCursorPos(&pt);

                if ((pt.x == ptOld.x) && (pt.y == ptOld.y))
                {
                    GetWindowRect(hwnd, &rc);
                    SetCursorPos(rc.left + 20, rc.top + 4);

                    // Note that the additional constants
                    // (20 and 4) are application-specific
```

```
                    // values to align the yin-shaped cursor
                    // and the yang-shaped icon.

                }
                else
                {
                    ptOld.x = pt.x;
                    ptOld.y = pt.y;
                }
            }

            return 0;

        case WM_SETCURSOR:
        // If the window is minimized, draw hCurs1.
        // If the window is not minimized, draw the
        // default cursor (class cursor).

            if (IsIconic(hwnd))
            {
                SetCursor(hCurs1);
                break;
            }

        case WM_DESTROY:
        // Destroy timer.

            KillTimer(hwnd, IDT_CURSOR);

            PostQuitMessage(0);
            break;
    }
}
```

Using the Keyboard to Move the Cursor

Because the system does not require a mouse, an application should be able to simulate mouse actions with the keyboard. The following example shows how to achieve this by using the GetCursorPos and SetCursorPos functions and by processing input from the arrow keys.

C++

```
HCURSOR hCurs1, hCurs2;    // cursor handles

POINT pt;                  // cursor location
RECT rc;                   // client area coordinates
static int repeat = 1;     // repeat key counter

//
```

```
// Other declarations and initialization.
//

switch (message)
{
//
// Process other messages.
//

    case WM_KEYDOWN:

        if (wParam != VK_LEFT && wParam != VK_RIGHT &&
        wParam != VK_UP && wParam != VK_DOWN)
        {
            break;
        }

        GetCursorPos(&pt);

        // Convert screen coordinates to client coordinates.

        ScreenToClient(hwnd, &pt);

        switch (wParam)
        {
        // Move the cursor to reflect which
        // arrow keys are pressed.

            case VK_LEFT:               // left arrow
                pt.x -= repeat;
                break;

            case VK_RIGHT:              // right arrow
                pt.x += repeat;
                break;

            case VK_UP:                 // up arrow
                pt.y -= repeat;
                break;

            case VK_DOWN:               // down arrow
                pt.y += repeat;
                break;

            default:
                return 0;
        }

        repeat++;           // Increment repeat count.

        // Keep the cursor in the client area.

        GetClientRect(hwnd, &rc);

        if (pt.x >= rc.right)
```

```
        {
            pt.x = rc.right - 1;
        }
        else
        {
            if (pt.x < rc.left)
            {
                pt.x = rc.left;
            }
        }

        if (pt.y >= rc.bottom)
            pt.y = rc.bottom - 1;
        else
            if (pt.y < rc.top)
                pt.y = rc.top;

        // Convert client coordinates to screen coordinates.

        ClientToScreen(hwnd, &pt);
        SetCursorPos(pt.x, pt.y);
        return 0;


    case WM_KEYUP:

        repeat = 1;            // Clear repeat count.
        return 0;

}
```

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Cursor Reference

Article • 04/27/2021

In This Section

- Cursor Functions
- Cursor Notifications
- Cursor Structures

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Cursor Functions

Article • 04/27/2021

In This Section

- ClipCursor
- CopyCursor
- CreateCursor
- DestroyCursor
- GetClipCursor
- GetCursor
- GetCursorInfo
- GetCursorPos
- GetPhysicalCursorPos
- LoadCursor
- LoadCursorFromFile
- SetCursor
- SetCursorPos
- SetPhysicalCursorPos
- SetSystemCursor
- ShowCursor

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

ClipCursor function (winuser.h)

Article02/22/2024

Confines the cursor to a rectangular area on the screen. If a subsequent cursor position (set by the SetCursorPos function or the mouse) lies outside the rectangle, the system automatically adjusts the position to keep the cursor inside the rectangular area.

Syntax

```
C++
```

```
BOOL ClipCursor(
  [in, optional] const RECT *lpRect
);
```

Parameters

`[in, optional] lpRect`

Type: **const RECT***

A pointer to the structure that contains the screen coordinates of the upper-left and lower-right corners of the confining rectangle. If this parameter is **NULL**, the cursor is free to move anywhere on the screen.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The cursor is a shared resource. If an application confines the cursor, it must release the cursor by using **ClipCursor** before relinquishing control to another application.

The calling process must have **WINSTA_WRITEATTRIBUTES** access to the window station.

Examples

For an example, see Confining a Cursor.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

Cursors

GetClipCursor

GetCursorPos

Other Resources

RECT

Reference

SetCursorPos

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CopyCursor macro (winuser.h)

Article02/22/2024

Copies the specified cursor.

Syntax

```
C++
```

```
void CopyCursor(
  [in]  pcur
);
```

Parameters

`[in] pcur`

Type: **HCURSOR**

A handle to the cursor to be copied.

Return value

None

Remarks

CopyCursor enables an application or DLL to obtain the handle to a cursor shape owned by another module. Then if the other module is freed, the application is still able to use the cursor shape.

Before closing, an application must call the DestroyCursor function to free any system resources associated with the cursor.

Do not use the **CopyCursor** function for animated cursors. Instead, use the CopyImage function.

CopyCursor is implemented as a call to the CopyIcon function.

```
syntax
```

```
#define CopyCursor(pcur) ((HCURSOR)CopyIcon((HICON)(pcur)))
```

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

See also

Conceptual

CopyIcon

CopyImage

Cursors

DestroyCursor

GetCursor

Reference

SetCursor

ShowCursor

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CreateCursor function (winuser.h)

Article01/26/2024

Creates a monochrome cursor having the specified size, bit patterns, and hot spot.

To create a colored cursor at run time you can use the **CreateIconIndirect** function, which creates a cursor based on the content of an **ICONINFO** structure.

Syntax

```
C++

HCURSOR CreateCursor(
  [in, optional] HINSTANCE  hInst,
  [in]           int        xHotSpot,
  [in]           int        yHotSpot,
  [in]           int        nWidth,
  [in]           int        nHeight,
  [in]           const VOID *pvANDPlane,
  [in]           const VOID *pvXORPlane
);
```

Parameters

`[in, optional] hInst`

Type: **HINSTANCE**

A handle to the current instance of the application creating the cursor.

`[in] xHotSpot`

Type: **int**

The horizontal position of the cursor's hot spot.

`[in] yHotSpot`

Type: **int**

The vertical position of the cursor's hot spot.

`[in] nWidth`

Type: **int**

The width of the cursor, in pixels.

`[in] nHeight`

Type: **int**

The height of the cursor, in pixels.

`[in] pvANDPlane`

Type: **const VOID***

An array of bytes that contains the bit values for the AND mask of the cursor, as in a monochrome bitmap. See remarks.

`[in] pvXORPlane`

Type: **const VOID***

An array of bytes that contains the bit values for the XOR mask of the cursor, as in a monochrome bitmap. See remarks.

Return value

Type: **HCURSOR**

If the function succeeds, the return value is a handle to the cursor.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

To determine the nominal size of a cursor, use the GetSystemMetrics function, specifying the **SM_CXCURSOR** or **SM_CYCURSOR** value. Also, you can use the DPI-aware version of this API, see (GetSystemMetricsForDpi)(/windows/win32/api/winuser/nf-winuser-getsystemmetricsfordpi). For more information see High DPI Desktop Application Development on Windows.

For more information about *pvANDPlane* and *pvXORPlane* parameters see description of *lpBits* parameter of CreateBitmap function.

CreateCursor applies the following truth table to the AND and XOR bitmasks:

Expand table

AND bitmask	XOR bitmask	Display
0	0	Black
0	1	White
1	0	Screen
1	1	Reverse screen

Before closing, an application must call the DestroyCursor function to free any system resources associated with the cursor.

DPI Virtualization

This API does not participate in DPI virtualization. The output returned is in terms of physical coordinates, and is not affected by the DPI of the calling thread. Note that the cursor created may still be scaled to match the DPI of any given window it is drawn into.

Examples

For an example, see Creating a Cursor.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

CreateIcon

CreateIconIndirect

DestroyCursor

GetSystemMetrics

SetCursor

Cursors

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

DestroyCursor function (winuser.h)

Article02/22/2024

Destroys a cursor and frees any memory the cursor occupied. Do not use this function to destroy a shared cursor.

Syntax

```
C++

BOOL DestroyCursor(
  [in] HCURSOR hCursor
);
```

Parameters

`[in] hCursor`

Type: **HCURSOR**

A handle to the cursor to be destroyed. The cursor must not be in use.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The **DestroyCursor** function destroys a nonshared cursor. Do not use this function to destroy a shared cursor. A shared cursor is valid as long as the module from which it was loaded remains in memory. The following functions obtain a shared cursor:

- LoadCursor
- LoadCursorFromFile
- LoadImage (if you use the **LR_SHARED** flag)

- CopyImage (if you use the **LR_COPYRETURNORG** flag and the *hImage* parameter is a shared cursor)

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

CopyCursor

CopyImage

CreateCursor

Cursors

LoadCursor

LoadCursorFromFile

LoadImage

Reference

Feedback

Was this page helpful?

GetClipCursor function (winuser.h)

Article02/22/2024

Retrieves the screen coordinates of the rectangular area to which the cursor is confined.

Syntax

```
C++

BOOL GetClipCursor(
  [out] LPRECT lpRect
);
```

Parameters

`[out] lpRect`

Type: **LPRECT**

A pointer to a RECT structure that receives the screen coordinates of the confining rectangle. The structure receives the dimensions of the screen if the cursor is not confined to a rectangle.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The cursor is a shared resource. If an application confines the cursor with the ClipCursor function, it must later release the cursor by using **ClipCursor** before relinquishing control to another application.

The calling process must have **WINSTA_READATTRIBUTES** access to the window station.

Examples

For an example, see Confining a Cursor.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

ClipCursor

Conceptual

Cursors

GetCursorPos

Other Resources

RECT

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetCursor function (winuser.h)

Article02/22/2024

Retrieves a handle to the current cursor.

To get information on the global cursor, even if it is not owned by the current thread, use GetCursorInfo.

Syntax

```
C++

HCURSOR GetCursor();
```

Return value

Type: **HCURSOR**

The return value is the handle to the current cursor. If there is no cursor, the return value is **NULL**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

Cursors

GetCursorInfo

Reference

SetCursor

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetCursorInfo function (winuser.h)

Article02/22/2024

Retrieves information about the global cursor.

Syntax

```
C++

BOOL GetCursorInfo(
  [in, out] PCURSORINFO pci
);
```

Parameters

`[in, out] pci`

Type: **PCURSORINFO**

A pointer to a CURSORINFO structure that receives the information. Note that you must set the **cbSize** member to `sizeof(CURSORINFO)` before calling this function.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

CURSORINFO

Conceptual

Cursors

GetGUIThreadInfo

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetCursorPos function (winuser.h)

Article11/19/2022

Retrieves the position of the mouse cursor, in screen coordinates.

Syntax

```
C++

BOOL GetCursorPos(
  [out] LPPOINT lpPoint
);
```

Parameters

`[out] lpPoint`

Type: **LPPOINT**

A pointer to a POINT structure that receives the screen coordinates of the cursor.

Return value

Type: **BOOL**

Returns nonzero if successful or zero otherwise. To get extended error information, call GetLastError.

Remarks

The cursor position is always specified in screen coordinates and is not affected by the mapping mode of the window that contains the cursor.

The calling process must have **WINSTA_READATTRIBUTES** access to the window station.

The input desktop must be the current desktop when you call **GetCursorPos**. Call OpenInputDesktop to determine whether the current desktop is the input desktop. If it is not, call SetThreadDesktop with the **HDESK** returned by **OpenInputDesktop** to switch to that desktop.

Examples

For an example, see Using the Keyboard to Move the Cursor.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-window-l1-1-2 (introduced in Windows 10, version 10.0.10240)

See also

ClipCursor

Conceptual

Cursors

GetCursorInfo

GetMessagePos

Other Resources

POINT

Reference

SetCursor

SetCursorPos

ShowCursor

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetPhysicalCursorPos function (winuser.h)

Article02/22/2024

Retrieves the position of the cursor in physical coordinates.

Syntax

```
C++

BOOL GetPhysicalCursorPos(
  [out] LPPOINT lpPoint
);
```

Parameters

`[out] lpPoint`

Type: **LPPOINT**

The position of the cursor, in physical coordinates.

Return value

Type: **BOOL**

TRUE if successful; otherwise **FALSE**.

GetLastError can be called to get more information about any error that is generated.

Remarks

For a description of the difference between logical coordinates and physical coordinates, see PhysicalToLogicalPoint.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-gui-l1-1-1 (introduced in Windows 8.1)

See also

ClipCursor

Conceptual

Cursors

GetCursorPos

Reference

SetCaretPos

SetCursor

SetCursorPos

SetPhysicalCursorPos

ShowCursor

Feedback

LoadCursorA function (winuser.h)

Article06/20/2023

Loads the specified cursor resource from the executable (.EXE) file associated with an application instance.

> ⓘ **Note**
>
> This function has been superseded by the **LoadImage** function (with **LR_DEFAULTSIZE** and **LR_SHARED** flags set).

Syntax

```
C++

HCURSOR LoadCursorA(
  [in, optional] HINSTANCE hInstance,
  [in]           LPCSTR    lpCursorName
);
```

Parameters

`[in, optional] hInstance`

Type: **HINSTANCE**

A handle to the module of either a DLL or executable (.exe) file that contains the cursor to be loaded. For more information, see GetModuleHandle.

To load a predefined system cursor, set this parameter to **NULL**.

`[in] lpCursorName`

Type: **LPCTSTR**

If *hInstance* is non-**NULL**, *lpCursorName* specifies the cursor resource either by name or ordinal. This ordinal must be packaged by using the MAKEINTRESOURCE macro.

If *hInstance* is **NULL**, *lpCursorName* specifies the identifier (beginning with the IDC_ prefix) of a predefined system cursor to load.

Return value

Type: **HCURSOR**

If the function succeeds, the return value is the handle to the newly loaded cursor.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

The **LoadCursor** function loads the cursor resource only if it has not been loaded; otherwise, it retrieves the handle to the existing resource. This function returns a valid cursor handle only if the *lpCursorName* parameter is a pointer to a cursor resource. If *lpCursorName* is a pointer to any type of resource other than a cursor (such as an icon), the return value is not **NULL**, even though it is not a valid cursor handle.

The **LoadCursor** function searches the cursor resource most appropriate for the cursor for the current display device. The cursor resource can be a color or monochrome bitmap.

DPI Virtualization

This API does not participate in DPI virtualization. The output returned is not affected by the DPI of the calling thread.

Examples

For an example, see Creating a Cursor.

> ⓘ **Note**
>
> The winuser.h header defines LoadCursor as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

Cursors

LoadImage

MAKEINTRESOURCE

IS_INTRESOURCE

Reference

SetCursor

SetCursorPos

ShowCursor

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

LoadCursorFromFileA function (winuser.h)

Article07/20/2023

Creates a cursor based on data contained in a file.

> ⓘ **Note**
>
> This function has been superseded by the **LoadImage** function (with **LR_DEFAULTSIZE** and **LR_LOADFROMFILE** flags set).

Syntax

```
C++

HCURSOR LoadCursorFromFileA(
  [in] LPCSTR lpFileName
);
```

Parameters

`[in] lpFileName`

Type: **LPCTSTR**

The source of the file data to be used to create the cursor. The data in the file must be in either .CUR or .ANI format.

If the high-order word of *lpFileName* is nonzero, it is a pointer to a string that is a fully qualified name of a file containing cursor data.

Return value

Type: **HCURSOR**

If the function is successful, the return value is a handle to the new cursor.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError. **GetLastError** may return the following value.

Expand table

Return code	Description
ERROR_FILE_NOT_FOUND	The specified file cannot be found.

Remarks

DPI Virtualization

This API does not participate in DPI virtualization. The output returned is not affected by the DPI of the calling thread.

> **Note**
>
> The winuser.h header defines LoadCursorFromFile as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

Cursors

LoadCursor

Reference

SetCursor

SetSystemCursor

Feedback

SetCursor function (winuser.h)

Article02/02/2023

Sets the cursor shape.

Syntax

```
C++

HCURSOR SetCursor(
  [in, optional] HCURSOR hCursor
);
```

Parameters

`[in, optional] hCursor`

Type: **HCURSOR**

A handle to the cursor.

The cursor must have been created by either the CreateCursor or the CreateIconIndirect function, or loaded by either the LoadCursor or the LoadImage function.

If this parameter is **NULL**, the cursor is removed from the screen.

Return value

Type: **HCURSOR**

The return value is the handle to the previous cursor, if there was one.

If there was no previous cursor, the return value is **NULL**.

Remarks

The cursor is set only if the new cursor is different from the previous cursor; otherwise, the function returns immediately.

The cursor is a shared resource. A window should set the cursor shape only when the cursor is in its client area or when the window is capturing mouse input. In systems

without a mouse, the window should restore the previous cursor before the cursor leaves the client area or before it relinquishes control to another window.

If your application must set the cursor while it is in a window, make sure the class cursor for the specified window's class is set to **NULL**. If the class cursor is not **NULL**, the system restores the class cursor each time the mouse is moved.

The cursor is not shown on the screen if the internal cursor display count is less than zero. This occurs if the application uses the ShowCursor function to hide the cursor more times than to show the cursor.

Examples

For an example, see Displaying a Cursor.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

Cursors

Creating a Cursor

CreateCursor

CreateIconIndirect

GetCursor

GetSystemMetrics

LoadCursor

LoadImage

Reference

SetCursorPos

ShowCursor

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

SetCursorPos function (winuser.h)

Article02/22/2024

Moves the cursor to the specified screen coordinates. If the new coordinates are not within the screen rectangle set by the most recent ClipCursor function call, the system automatically adjusts the coordinates so that the cursor stays within the rectangle.

Syntax

```
C++

BOOL SetCursorPos(
  [in] int X,
  [in] int Y
);
```

Parameters

`[in] X`

Type: **int**

The new x-coordinate of the cursor, in screen coordinates.

`[in] Y`

Type: **int**

The new y-coordinate of the cursor, in screen coordinates.

Return value

Type: **BOOL**

Returns nonzero if successful or zero otherwise. To get extended error information, call GetLastError.

Remarks

The cursor is a shared resource. A window should move the cursor only when the cursor is in the window's client area.

The calling process must have **WINSTA_WRITEATTRIBUTES** access to the window station.

The input desktop must be the current desktop when you call **SetCursorPos**. Call OpenInputDesktop to determine whether the current desktop is the input desktop. If it is not, call SetThreadDesktop with the **HDESK** returned by **OpenInputDesktop** to switch to that desktop.

Examples

For an example, see Using the Keyboard to Move the Cursor.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-window-l1-1-2 (introduced in Windows 10, version 10.0.10240)

See also

ClipCursor

Conceptual

Cursors

GetCursorPos

Reference

SetCaretPos

SetCursor

ShowCursor

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

SetPhysicalCursorPos function (winuser.h)

Article02/22/2024

Sets the position of the cursor in physical coordinates.

Syntax

```
C++
```

```
BOOL SetPhysicalCursorPos(
  [in] int X,
  [in] int Y
);
```

Parameters

`[in] X`

Type: **int**

The new x-coordinate of the cursor, in physical coordinates.

`[in] Y`

Type: **int**

The new y-coordinate of the cursor, in physical coordinates.

Return value

Type: **BOOL**

TRUE if successful; otherwise **FALSE**.

Remarks

For a description of the difference between logical coordinates and physical coordinates, see PhysicalToLogicalPoint.

GetLastError can be called to get more information about any error that is generated.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-gui-l1-1-1 (introduced in Windows 8.1)

See also

ClipCursor

Conceptual

Cursors

GetCursorPos

GetPhysicalCursorPos

Reference

SetCaretPos

SetCursor

SetCursorPos

ShowCursor

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

SetSystemCursor function (winuser.h)

Article02/22/2024

Enables an application to customize the system cursors. It replaces the contents of the system cursor specified by the *id* parameter with the contents of the cursor specified by the *hcur* parameter and then destroys *hcur*.

Syntax

```
C++

BOOL SetSystemCursor(
  [in] HCURSOR hcur,
  [in] DWORD   id
);
```

Parameters

`[in] hcur`

Type: **HCURSOR**

A handle to the cursor. The function replaces the contents of the system cursor specified by *id* with the contents of the cursor handled by *hcur*.

The system destroys *hcur* by calling the DestroyCursor function. Therefore, *hcur* cannot be a cursor loaded using the LoadCursor function. To specify a cursor loaded from a resource, copy the cursor using the CopyCursor function, then pass the copy to **SetSystemCursor**.

`[in] id`

Type: **DWORD**

The system cursor to replace with the contents of *hcur*. This parameter can be one of the following values.

Expand table

Value	Meaning
OCR_NORMAL `32512`	Normal select
OCR_IBEAM `32513`	Text select
OCR_WAIT `32514`	Busy
OCR_CROSS `32515`	Precision select
OCR_UP `32516`	Alternate select
OCR_SIZENWSE `32642`	Diagonal resize 1
OCR_SIZENESW `32643`	Diagonal resize 2
OCR_SIZEWE `32644`	Horizontal resize
OCR_SIZENS `32645`	Vertical resize
OCR_SIZEALL `32646`	Move
OCR_NO `32648`	Unavailable
OCR_HAND `32649`	Link select
OCR_APPSTARTING `32650`	Working in background

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

For an application to use any of the OCR_ constants, the constant **OEMRESOURCE** must be defined before the Windows.h header file is included.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

Cursors

DestroyCursor

LoadCursor

LoadCursorFromFile

Reference

SetCursor

Feedback

Was this page helpful? Yes No

ShowCursor function (winuser.h)

Article02/22/2024

Displays or hides the cursor.

Syntax

```
C++
```

```
int ShowCursor(
  [in] BOOL bShow
);
```

Parameters

`[in] bShow`

Type: **BOOL**

If *bShow* is **TRUE**, the display count is incremented by one. If *bShow* is **FALSE**, the display count is decremented by one.

Return value

Type: **int**

The return value specifies the new display counter.

Remarks

Windows 8: Call GetCursorInfo to determine the cursor visibility.

This function sets an internal display counter that determines whether the cursor should be displayed. The cursor is displayed only if the display count is greater than or equal to 0. If a mouse is installed, the initial display count is 0. If no mouse is installed, the display count is –1.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

ClipCursor

Conceptual

Cursors

GetCursorPos

Reference

SetCursor

SetCursorPos

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Cursor Notifications

Article • 04/27/2021

In This Section

- WM_SETCURSOR

Feedback

WM_SETCURSOR message

Article • 12/11/2020

Sent to a window if the mouse causes the cursor to move within a window and mouse input is not captured.

```
C++

#define WM_SETCURSOR                    0x0020
```

Parameters

wParam

A handle to the window that contains the cursor.

lParam

The low-order word of *lParam* specifies the hit-test result for the cursor position. See the return values for WM_NCHITTEST for possible values.

The high-order word of *lParam* specifies the mouse window message which triggered this event, such as WM_MOUSEMOVE. When the window enters menu mode, this value is zero.

Return value

If an application processes this message, it should return **TRUE** to halt further processing or **FALSE** to continue.

Remarks

The **DefWindowProc** function passes the **WM_SETCURSOR** message to a parent window before processing. If the parent window returns **TRUE**, further processing is halted. Passing the message to a window's parent window gives the parent window control over the cursor's setting in a child window. The **DefWindowProc** function also uses this message to set the cursor to an arrow if it is not in the client area, or to the registered class cursor if it is in the client area. If the low-order word of the *lParam* parameter is **HTERROR** and the high-order word of *lParam* specifies that one of the mouse buttons is pressed, **DefWindowProc** calls the **MessageBeep** function.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Reference

DefWindowProc

HIWORD

LOWORD

Conceptual

Cursors

Feedback

Was this page helpful? Yes No

Cursor Structures

Article • 04/27/2021

In This Section

- CURSORINFO

Feedback

Was this page helpful? Yes No

CURSORINFO structure (winuser.h)

Article02/22/2024

Contains global cursor information.

Syntax

```
C++
```

```
typedef struct tagCURSORINFO {
  DWORD   cbSize;
  DWORD   flags;
  HCURSOR hCursor;
  POINT   ptScreenPos;
} CURSORINFO, *PCURSORINFO, *LPCURSORINFO;
```

Members

`cbSize`

Type: **DWORD**

The size of the structure, in bytes. The caller must set this to `sizeof(CURSORINFO)`.

`flags`

Type: **DWORD**

The cursor state. This parameter can be one of the following values.

Expand table

Value	Meaning
0	The cursor is hidden.
CURSOR_SHOWING 0x00000001	The cursor is showing.
CURSOR_SUPPRESSED 0x00000002	**Windows 8**: The cursor is suppressed. This flag indicates that the system is not drawing the cursor because the user is providing input through touch or pen instead of the mouse.

`hCursor`

Type: **HCURSOR**

A handle to the cursor.

`ptScreenPos`

Type: **POINT**

A structure that receives the screen coordinates of the cursor.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

Cursors

GetCursorInfo

POINT

Reference

Feedback

Icons (Menus and Other Resources)

Article • 12/10/2020

An *icon* is a picture that consists of a bitmap image combined with a mask to create transparent areas in the picture. The term icon can refer to either of the following:

- A single icon image. This is a resource of type **RT_ICON**.
- A group of images, from which the system or an application can choose the most appropriate icon based on size and color depth. This is a resource of type **RT_GROUP_ICON**.

In This Section

Name	Description
About Icons	Discusses icons.
Using Icons	Discusses how to perform tasks related to icons.
Icon Reference	Contains the API reference.

Icon Functions

Name	Description
CopyIcon	Copies the specified icon from another module to the current module.
CreateIcon	Creates an icon that has the specified size, colors, and bit patterns.
CreateIconFromResource	Creates an icon or cursor from resource bits describing the icon.
CreateIconFromResourceEx	Creates an icon or cursor from resource bits describing the icon.
CreateIconIndirect	Creates an icon or cursor from an ICONINFO structure.
DestroyIcon	Destroys an icon and frees any memory the icon occupied.
DrawIcon	Draws an icon or cursor into the specified device context.
DrawIconEx	Draws an icon or cursor into the specified device context, performing the specified raster operations, and stretching or compressing the icon or cursor as specified.
DuplicateIcon	Creates a duplicate of a specified icon.

Name	Description
ExtractAssociatedIcon	Retrieves a handle to an indexed icon found in a file or an icon found in an associated executable file.
ExtractIcon	Retrieves a handle to an icon from the specified executable file,DLL, or icon file.
ExtractIconEx	Creates an array of handles to large or small icons extracted from the specified executable file, DLL, or icon file.
GetIconInfo	Retrieves information about the specified icon or cursor.
GetIconInfoEx	Retrieves information about the specified icon or cursor. GetIconInfoEx extends GetIconInfo by using the newer ICONINFOEX structure.
LoadIcon	Loads the specified icon resource from the executable (.exe) file associated with an application instance.
LookupIconIdFromDirectory	Searches through icon or cursor data for the icon or cursor that best fits the current display device.
LookupIconIdFromDirectoryEx	Searches through icon or cursor data for the icon or cursor that best fits the current display device.
PrivateExtractIcons	Creates an array of handles to icons that are extracted from a specified file.

Icon Structures

Name	Description
ICONINFO	Contains information about an icon or a cursor.
ICONINFOEX	Contains information about an icon or a cursor. Extends ICONINFO. Used by GetIconInfoEx.
ICONMETRICS	Contains the scalable metrics associated with icons. This structure is used with the SystemParametersInfo function when the **SPI_GETICONMETRICS** or **SPI_SETICONMETRICS** action is specified.

Get help at Microsoft Q&A

About Icons

Article • 06/22/2023

The system uses icons throughout the user interface to represent objects such as files, folders, shortcuts, applications, and documents. The icon functions enable applications to create, load, display, arrange, animate, and destroy icons. For information on specifying icons for file types, see **ExtractIcon**.

This overview provides information on the following topics:

Icon Hot Spot

One of the pixels in an icon is designated as the hot spot, which is the point the system tracks and recognizes as the position of the icon. An icon's hot spot is typically the pixel located at the center of the icon. If you use the **CreateIconIndirect** function to create an icon, you can specify any pixel to be the hot spot.

Icon Types

The operating system provides a set of standard icons that are available for any application to use at any time. The software development kit (SDK) header files contain identifiers for the **system icons** — the identifiers begin with the **IDI_** prefix.

Value	Meaning

Value	Meaning
IDI_APPLICATION MAKEINTRESOURCE(32512)	Default application icon
IDI_ERROR MAKEINTRESOURCE(32513)	Error icon
IDI_QUESTION MAKEINTRESOURCE(32514)	Question mark icon
IDI_WARNING MAKEINTRESOURCE(32515)	Warning icon
IDI_INFORMATION MAKEINTRESOURCE(32516)	Information icon
IDI_WINLOGO MAKEINTRESOURCE(32517)	Windows logo icon
IDI_SHIELD MAKEINTRESOURCE(32518)	Security shield icon

See Guidelines for information on recommended usage of standard icons.

Also, starting with Windows Vista, an additional set of **standard system shell icons** is available through the SHGetStockIconInfo method.

Custom icons are designed for use in a particular application and can be any design. User can load custom icons from files or create them at run-time. Following are several custom icons.

Icon Sizes

The system uses four icon sizes:

- System small
- System large
- Shell small
- Shell large
- Jumbo (starting Windows Vista)

The *system small icon* is displayed in the window caption.

See Icon scaling for recommendations on preferred icon sizes for your application.

To change the size of the system small icon

1. From Control Panel, click **Display**, then click the **Appearance** tab.
2. Select **Caption Buttons** from the **Item** list, then set the **Size** field.

To retrieve the size of the system small icon

- Call the **GetSystemMetrics** function with **SM_CXSMICON** and **SM_CYSMICON**.

The *system large icon* is mainly used by applications, but it is also displayed in the Alt+Tab dialog. The **CreateIconFromResource**, **DrawIcon**, **ExtractAssociatedIcon**, **ExtractIcon**, **ExtractIconEx**, and **LoadIcon** functions all use system large icons. The size of the system large icon is defined by the video driver, therefore it cannot be changed.

To retrieve the size of the system large icon

- Call **GetSystemMetrics** with **SM_CXICON** and **SM_CYICON**.

The **CreateIcon**, **CreateIconFromResourceEx**, **CreateIconIndirect**, and **SHGetFileInfo** functions can be used to work with icons in sizes other than system large.

The *shell small icon* is used in the Windows Explorer and the common dialogs. Currently, this defaults to the system small size.

To retrieve the size of the shell small icon

1. Use the SHGetFileInfo function with `SHGFI_SHELLICONSIZE | SHGFI_SMALLICON` to retrieve a handle to the system image list.
2. Then call the **ImageList_GetIconSize** function to get the icon size.

The shell large icon is used on the desktop.

To change the size of the large icon

1. From Control Panel , click **Display**, then click the **Appearance** tab,
2. Select **Icon** from the **Item** list, then set the **Size** field (this size is stored in the registry, under **HKEY_CURRENT_USER\Control Panel, Desktop\WindowMetrics\Shell Icon Size**).
3. Click the **Plus!** tab and then select the **Use Large Icons** check box.

To retrieve the size of the shell large icon

1. Use the **SHGetFileInfo** function with **SHGFI_SHELLICONSIZE** to retrieve a handle to the system image list.
2. Then call the **ImageList_GetIconSize** function to get the icon size.

When filling in the **WNDCLASSEX** structure to be used in registering your window class, set the **hIcon** member to the system large icon (usually 32x32) and the **hIconSm** member to the system small icon (usually 16x16). For more information about class icons, see Class Icons.

Icon Creation

Standard icons are predefined, so it is not necessary to create them. To use a standard icon, an application can obtain its handle by using the **LoadImage** function. An *icon handle* is a unique value of the **HICON** type that identifies a standard or custom icon.

To create a custom icon for an application, you would typically use a graphics application and include the ICON Resource in the application's resource-definition file. At run-time, you can call **LoadIcon** or **LoadImage** to retrieve a handle to the icon. An icon resource can contain a group of images for several different display devices. **LoadIcon** and **LoadImage** automatically select the most appropriate icon from the group for the current display device.

An application can also create a custom icon at run-time by using the **CreateIconIndirect** function, which creates an icon based on the contents of an **ICONINFO** structure. The **GetIconInfo** function fills the structure with the hot-spot coordinates and information about the bitmask bitmap and color bitmap for the icon.

Applications should implement custom icons as resources and should use **LoadIcon** or **LoadImage**, rather than create the icon at run-time. Using icon resources avoids device dependence, simplifies localization, and enables applications to share icon shapes.

The **CreateIconFromResourceEx** function enables an application to browse through the system's resources and create icons and cursors based on resource data. **CreateIconFromResourceEx** creates an icon based on binary resource data from other executable files or DLLs. An application must precede this function with calls to the **LookupIconIdFromDirectoryEx** function and several of the resource functions. **LookupIconIdFromDirectoryEx** returns the identifier of the most appropriate icon data for the current display device.

Icon Display

You can retrieve the image for an icon by using the **GetIconInfo** function, and can draw it by using the **DrawIconEx** function. To draw the default image for a icon, specify the **DI_COMPAT** flag in the call to **DrawIconEx**. If you do not specify the **DI_COMPAT** flag, **DrawIconEx** draws the icon using the image that the user specified.

When the system displays an icon, it must extract the appropriate icon image from the .exe or .dll file. The system uses the following steps to select the icon image:

1. Select the **RT_GROUP_ICON** resource. If more than one such resource exists, the system uses the first resource listed in the resource scrip.
2. Select the appropriate **RT_ICON** image from the **RT_GROUP_ICON** resource. If more than one image exists, the system uses the following criteria to choose an image:

    - The image closest in size to the requested size is chosen.
    - If two or more images of that size are present, the one that matches the color depth of the display is chosen.
    - If no images exactly match the color depth of the display, the image with the greatest color depth that does not exceed the color depth of the display is chosen. If all exceed the color depth, the one with the lowest color depth is chosen.

> ⓘ **Note**
>
> The system treats all color depths of 8 or more bpp as equal. Therefore, there is no advantage of including a 16x16 256-color image and a 16x16 16-color image in the same resource—the system will simply choose the first one it encounters. When the display is in 8-bpp mode, the system will choose a 16-color icon over a 256-color icon, and will display all icons using the system default palette.

To display an animated icon, use a static control as shown in the following code fragment.

```
hIcon = LoadImage(NULL, "ico.ani", IMAGE_ICON, 0, 0, LR_LOADFROMFILE);
SendMessage( hStatic, STM_SETIMAGE, IMAGE_ICON, (LPARAM)(UINT)hIcon);
```

Icon Destruction

When an application no longer needs an icon it created by using the **CreateIconIndirect** function, it should destroy the icon. The **DestroyIcon** function destroys the icon handle and frees any memory used by the icon. Applications should use this function only for icons created with **CreateIconIndirect**; it is not necessary to destroy other icons.

Icon Duplication

The **CopyIcon** function copies an icon handle. This enables an application or DLL to get its own handle to an icon owned by another module. Then, if the other module is freed, the application that copied the icon will still be able to use the icon.

The **CopyImage** function creates a new icon based on the specified source icon. The new icon can be larger or smaller than the source icon.

For information about adding, removing, or replacing icon resources in executable (.exe) files, see Resources.

The **DuplicateIcon** function makes an actual copy of the icon.

Using Icons

Article • 10/26/2022

The following topics describe how to perform certain tasks related to icons:

- Creating an Icon
- Geting the Icon size
- Displaying an Icon
- Sharing Icon Resources

Creating an Icon

To use an icon, your application must get a handle to the icon. The following example shows how to create two different icon handles: one for the standard question icon and one for a custom icon included as a resource in the application's resource-definition file.

```
C++
```

```
HICON hIcon1;   // icon handle
HICON hIcon2;   // icon handle

// Create a standard question icon.

hIcon1 = LoadIcon(NULL, IDI_QUESTION);

// Create a custom icon based on a resource.

hIcon2 = LoadIcon(hinst, MAKEINTRESOURCE(460));

// Create a custom icon at run time.
```

An application should implement custom icons as resources and should use the **LoadIcon** or **LoadImage** function, rather than create the icons at run-time. This approach avoids device dependence, simplifies localization, and enables applications to share icon bitmaps. However, the following example uses **CreateIcon** to create a custom monochrome icon at run-time, based on bitmap bitmasks; it is included to illustrate how the system interprets icon bitmap bitmasks.

```
C++
```

```
HICON hIcon3;      // icon handle

// Yang icon AND bitmask
```

```
BYTE ANDmaskIcon[] = {0xFF, 0xFF, 0xFF, 0xFF,   // line 1
                      0xFF, 0xFF, 0xC3, 0xFF,   // line 2
                      0xFF, 0xFF, 0x00, 0xFF,   // line 3
                      0xFF, 0xFE, 0x00, 0x7F,   // line 4

                      0xFF, 0xFC, 0x00, 0x1F,   // line 5
                      0xFF, 0xF8, 0x00, 0x0F,   // line 6
                      0xFF, 0xF8, 0x00, 0x0F,   // line 7
                      0xFF, 0xF0, 0x00, 0x07,   // line 8

                      0xFF, 0xF0, 0x00, 0x03,   // line 9
                      0xFF, 0xE0, 0x00, 0x03,   // line 10
                      0xFF, 0xE0, 0x00, 0x01,   // line 11
                      0xFF, 0xE0, 0x00, 0x01,   // line 12

                      0xFF, 0xF0, 0x00, 0x01,   // line 13
                      0xFF, 0xF0, 0x00, 0x00,   // line 14
                      0xFF, 0xF8, 0x00, 0x00,   // line 15
                      0xFF, 0xFC, 0x00, 0x00,   // line 16

                      0xFF, 0xFF, 0x00, 0x00,   // line 17
                      0xFF, 0xFF, 0x80, 0x00,   // line 18
                      0xFF, 0xFF, 0xE0, 0x00,   // line 19
                      0xFF, 0xFF, 0xE0, 0x01,   // line 20

                      0xFF, 0xFF, 0xF0, 0x01,   // line 21
                      0xFF, 0xFF, 0xF0, 0x01,   // line 22
                      0xFF, 0xFF, 0xF0, 0x03,   // line 23
                      0xFF, 0xFF, 0xE0, 0x03,   // line 24

                      0xFF, 0xFF, 0xE0, 0x07,   // line 25
                      0xFF, 0xFF, 0xC0, 0x0F,   // line 26
                      0xFF, 0xFF, 0xC0, 0x0F,   // line 27
                      0xFF, 0xFF, 0x80, 0x1F,   // line 28

                      0xFF, 0xFF, 0x00, 0x7F,   // line 29
                      0xFF, 0xFC, 0x00, 0xFF,   // line 30
                      0xFF, 0xF8, 0x03, 0xFF,   // line 31
                      0xFF, 0xFC, 0x3F, 0xFF};  // line 32

// Yang icon XOR bitmask

BYTE XORmaskIcon[] = {0x00, 0x00, 0x00, 0x00,   // line 1
                      0x00, 0x00, 0x00, 0x00,   // line 2
                      0x00, 0x00, 0x00, 0x00,   // line 3
                      0x00, 0x00, 0x00, 0x00,   // line 4

                      0x00, 0x00, 0x00, 0x00,   // line 5
                      0x00, 0x00, 0x00, 0x00,   // line 6
                      0x00, 0x00, 0x00, 0x00,   // line 7
                      0x00, 0x00, 0x38, 0x00,   // line 8

                      0x00, 0x00, 0x7C, 0x00,   // line 9
                      0x00, 0x00, 0x7C, 0x00,   // line 10
                      0x00, 0x00, 0x7C, 0x00,   // line 11
```

```
                        0x00, 0x00, 0x38, 0x00,   // line 12

                        0x00, 0x00, 0x00, 0x00,   // line 13
                        0x00, 0x00, 0x00, 0x00,   // line 14
                        0x00, 0x00, 0x00, 0x00,   // line 15
                        0x00, 0x00, 0x00, 0x00,   // line 16

                        0x00, 0x00, 0x00, 0x00,   // line 17
                        0x00, 0x00, 0x00, 0x00,   // line 18
                        0x00, 0x00, 0x00, 0x00,   // line 19
                        0x00, 0x00, 0x00, 0x00,   // line 20

                        0x00, 0x00, 0x00, 0x00,   // line 21
                        0x00, 0x00, 0x00, 0x00,   // line 22
                        0x00, 0x00, 0x00, 0x00,   // line 23
                        0x00, 0x00, 0x00, 0x00,   // line 24

                        0x00, 0x00, 0x00, 0x00,   // line 25
                        0x00, 0x00, 0x00, 0x00,   // line 26
                        0x00, 0x00, 0x00, 0x00,   // line 27
                        0x00, 0x00, 0x00, 0x00,   // line 28

                        0x00, 0x00, 0x00, 0x00,   // line 29
                        0x00, 0x00, 0x00, 0x00,   // line 30
                        0x00, 0x00, 0x00, 0x00,   // line 31
                        0x00, 0x00, 0x00, 0x00};  // line 32

hIcon3 = CreateIcon(hinst,    // application instance
             32,              // icon width
             32,              // icon height
             1,               // number of XOR planes
             1,               // number of bits per pixel
             ANDmaskIcon,     // AND bitmask
             XORmaskIcon);    // XOR bitmask
```

To create the icon, **CreateIcon** applies the following truth table to the AND and XOR bitmasks.

AND bitmask	XOR bitmask	Display
0	0	Black
0	1	White
1	0	Screen
1	1	Reverse screen

To create a colored icon at run time you must use the **CreateIconIndirect** function, which creates a icon based on the content of an **ICONINFO** structure.

Before closing, your application must use **DestroyIcon** to destroy any icon it created by using **CreateIcon** or **CreateIconIndirect**. It is not necessary to destroy icons created by other functions.

Getting the Icon size

Here is example code how to get the icon size from the **HICON** handle:

C++

```
// Also works for cursors
BOOL GetIconDimensions(__in HICON hico, __out SIZE *psiz)
{
  ICONINFO ii;
  BOOL fResult = GetIconInfo(hico, &ii);
  if (fResult) {
    BITMAP bm;
    fResult = GetObject(ii.hbmMask, sizeof(bm), &bm) == sizeof(bm);
    if (fResult) {
      psiz->cx = bm.bmWidth;
      psiz->cy = ii.hbmColor ? bm.bmHeight : bm.bmHeight / 2;
    }
    if (ii.hbmMask)  DeleteObject(ii.hbmMask);
    if (ii.hbmColor) DeleteObject(ii.hbmColor);
  }
  return fResult;
}
```

Displaying an Icon

Your application can load and create icons to display in the application's client area or child windows. The following example demonstrates how to draw an icon in the client area of the window whose device context (DC) is identified by the *hdc* parameter.

C++

```
HICON hIcon1;   // icon handle
HDC hdc;        // handle to display context

DrawIcon(hdc, 10, 20, hIcon1);
```

The system automatically displays the class icon(s) for a window. Your application can assign class icons while registering a window class. Your application can replace a class icon by using the **SetClassLong** function. This function changes the default window

settings for all windows of a given class. The following example replaces a class icon with the icon whose resource identifier is 480.

```
C++

HINSTANCE hinst;            // handle to current instance
HWND hwnd;                  // main window handle

// Change the icon for hwnd's window class.

SetClassLongPtr(hwnd,          // window handle
    GCLP_HICON,               // changes icon
    (LONG_PTR) LoadIcon(hinst, MAKEINTRESOURCE(480))
   );
```

For more information about window classes, see Window Classes.

Sharing Icon Resources

The following code uses the functions **CreateIconFromResourceEx**, **DrawIcon**, and **LookupIconIdFromDirectoryEx**, and several of the resource functions, to create an icon handle based on icon data from another executable file. Then, it displays the icon in a window.

Security Warning: Using **LoadLibrary** incorrectly can compromise the security of your application by loading the wrong DLL. Refer to the **LoadLibrary** documentation for information on how to correctly load DLLs with different versions of Windows.

```
C++

HICON hIcon1;       // icon handle
HINSTANCE hExe;     // handle to loaded .EXE file
HRSRC hResource;    // handle to FindResource
HRSRC hMem;         // handle to LoadResource
BYTE *lpResource;   // pointer to resource data
int nID;            // ID of resource that best fits current screen

HDC hdc;        // handle to display context

// Load the file from which to copy the icon.
// Note: LoadLibrary should have a fully explicit path.
//
hExe = LoadLibrary("myapp.exe");
if (hExe == NULL)
{
    //Error loading module -- fail as securely as possible
    return;
}
```

```
// Find the icon directory whose identifier is 440.

hResource = FindResource(hExe,
    MAKEINTRESOURCE(440),
    RT_GROUP_ICON);

// Load and lock the icon directory.

hMem = LoadResource(hExe, hResource);

lpResource = LockResource(hMem);

// Get the identifier of the icon that is most appropriate
// for the video display.

nID = LookupIconIdFromDirectoryEx((PBYTE) lpResource, TRUE,
    CXICON, CYICON, LR_DEFAULTCOLOR);

// Find the bits for the nID icon.

hResource = FindResource(hExe,
    MAKEINTRESOURCE(nID),
    MAKEINTRESOURCE(RT_ICON));

// Load and lock the icon.

hMem = LoadResource(hExe, hResource);

lpResource = LockResource(hMem);

// Create a handle to the icon.

hIcon1 = CreateIconFromResourceEx((PBYTE) lpResource,
    SizeofResource(hExe, hResource), TRUE, 0x00030000,
    CXICON, CYICON, LR_DEFAULTCOLOR);

// Draw the icon in the client area.

DrawIcon(hdc, 10, 20, hIcon1);
```

Feedback

Icon Reference

Article • 04/27/2021

In This Section

- Icon Functions
- Icon Structures

Feedback

Was this page helpful? Yes No

Icon Functions

Article • 04/27/2021

In This Section

- CopyIcon
- CreateIcon
- CreateIconFromResource
- CreateIconFromResourceEx
- CreateIconIndirect
- DestroyIcon
- DrawIcon
- DrawIconEx
- DuplicateIcon
- ExtractAssociatedIcon
- ExtractIcon
- ExtractIconEx
- GetIconInfo
- GetIconInfoEx
- LoadIcon
- LookupIconIdFromDirectory
- LookupIconIdFromDirectoryEx
- PrivateExtractIcons

Feedback

CopyIcon function (winuser.h)

Article02/22/2024

Copies the specified icon from another module to the current module.

Syntax

```
C++

HICON CopyIcon(
  [in] HICON hIcon
);
```

Parameters

`[in] hIcon`

Type: **HICON**

A handle to the icon to be copied.

Return value

Type: **HICON**

If the function succeeds, the return value is a handle to the duplicate icon.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

The **CopyIcon** function enables an application or DLL to get its own handle to an icon owned by another module. If the other module is freed, the application icon will still be able to use the icon.

Before closing, an application must call the DestroyIcon function to free any system resources associated with the icon.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-gui-l1-3-0 (introduced in Windows 10, version 10.0.10240)

See also

Conceptual

CopyCursor

DestroyIcon

DrawIcon

DrawIconEx

Icons

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CreateIcon function (winuser.h)

Article01/26/2024

Creates an icon that has the specified size, colors, and bit patterns.

To create a colored icon at run time you can use the **CreateIconIndirect** function, which creates a icon based on the content of an **ICONINFO** structure.

Syntax

```
C++

HICON CreateIcon(
  [in, optional] HINSTANCE  hInstance,
  [in]           int        nWidth,
  [in]           int        nHeight,
  [in]           BYTE       cPlanes,
  [in]           BYTE       cBitsPixel,
  [in]           const BYTE *lpbANDbits,
  [in]           const BYTE *lpbXORbits
);
```

Parameters

`[in, optional] hInstance`

Type: **HINSTANCE**

A handle to the instance of the module creating the icon.

`[in] nWidth`

Type: **int**

The width, in pixels, of the icon. See remarks.

`[in] nHeight`

Type: **int**

The height, in pixels, of the icon. See remarks.

`[in] cPlanes`

Type: **BYTE**

The number of planes in the XOR bitmask of the icon. See remarks.

`[in] cBitsPixel`

Type: **BYTE**

The number of bits-per-pixel in the XOR bitmask of the icon.

`[in] lpbANDbits`

Type: **const BYTE***

An array of bytes that contains the bit values for the AND bitmask of the icon. This bitmask describes a monochrome bitmap. See remarks.

`[in] lpbXORbits`

Type: **const BYTE***

An array of bytes that contains the bit values for the XOR bitmask of the icon. This bitmask describes a monochrome or color bitmap. See remarks.

Return value

Type: **HICON**

If the function succeeds, the return value is a handle to an icon.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

To determine the nominal size of the icon, use the GetSystemMetrics function, specifying the **SM_CXICON** or **SM_CYICON** value. Also, you can use the DPI-aware version of this API, see (GetSystemMetricsForDpi)(/windows/win32/api/winuser/nf-winuser-getsystemmetricsfordpi). For more information see Icon Sizes and High DPI Desktop Application Development on Windows.

For more information about *lpbANDbits* and *lpbXORbits* parameters see description of *lpBits* parameter of CreateBitmap function.

In case of monochrome icon **CreateIcon** applies the following truth table to the AND and XOR bitmasks:

Expand table

AND bitmask	XOR bitmask	Display
0	0	Black
0	1	White
1	0	Screen
1	1	Reverse screen

When you are finished using the icon, destroy it using the DestroyIcon function.

Examples

For an example, see Creating an Icon.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

CreateIconIndirect

Icons

Icon Sizes

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CreateIconFromResource function (winuser.h)

Article01/26/2024

Creates an icon or cursor from resource bits describing the icon.

To specify a desired height or width, use the CreateIconFromResourceEx function.

Syntax

```
C++

HICON CreateIconFromResource(
  [in] PBYTE presbits,
  [in] DWORD dwResSize,
  [in] BOOL  fIcon,
  [in] DWORD dwVer
);
```

Parameters

`[in] presbits`

Type: **PBYTE**

The DWORD-aligned buffer pointer containing the icon or cursor resource bits. These bits are typically loaded by calls to the LookupIconIdFromDirectory, LookupIconIdFromDirectoryEx, and LoadResource functions.

See Cursor and Icon Resources for more info on icon and cursor resource format.

`[in] dwResSize`

Type: **DWORD**

The size, in bytes, of the set of bits pointed to by the *presbits* parameter.

`[in] fIcon`

Type: **BOOL**

Indicates whether an icon or a cursor is to be created. If this parameter is **TRUE**, an icon is to be created. If it is **FALSE**, a cursor is to be created.

The LOCALHEADER structure defines cursor hotspot and is the first data read from the cursor resource bits.

`[in] dwVer`

Type: **DWORD**

The version number of the icon or cursor format for the resource bits pointed to by the *presbits* parameter. The value must be greater than or equal to 0x00020000 and less than or equal to 0x00030000. This parameter is generally set to 0x00030000.

Return value

Type: **HICON**

If the function succeeds, the return value is a handle to the icon or cursor.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

The **CreateIconFromResource**, CreateIconFromResourceEx, CreateIconIndirect, GetIconInfo, LookupIconIdFromDirectory, and LookupIconIdFromDirectoryEx functions allow shell applications and icon browsers to examine and use resources throughout the system.

The **CreateIconFromResource** function calls CreateIconFromResourceEx passing `LR_DEFAULTSIZE|LR_SHARED` as flags.

You should call DestroyIcon for icons or DestroyCursor for cursors created with **CreateIconFromResource**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

CreateIconFromResourceEx

CreateIconIndirect

GetIconInfo

Icons

LoadResource

LookupIconIdFromDirectory

LookupIconIdFromDirectoryEx

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CreateIconFromResourceEx function (winuser.h)

Article01/26/2024

Creates an icon or cursor from resource bits describing the icon.

Syntax

```
C++

HICON CreateIconFromResourceEx(
  [in] PBYTE presbits,
  [in] DWORD dwResSize,
  [in] BOOL  fIcon,
  [in] DWORD dwVer,
  [in] int   cxDesired,
  [in] int   cyDesired,
  [in] UINT  Flags
);
```

Parameters

`[in] presbits`

Type: **PBYTE**

The DWORD-aligned buffer pointer containing the icon (**RT_ICON**) or cursor (**RT_CURSOR**) resource bits. These bits are typically loaded by calls to the LookupIconIdFromDirectoryEx and LoadResource functions.

See Cursor and Icon Resources for more info on icon and cursor resource format.

`[in] dwResSize`

Type: **DWORD**

The size, in bytes, of the set of bits pointed to by the *pbIconBits* parameter.

`[in] fIcon`

Type: **BOOL**

Indicates whether an icon or a cursor is to be created. If this parameter is **TRUE**, an icon is to be created. If it is **FALSE**, a cursor is to be created.

The LOCALHEADER structure defines cursor hotspot and is the first data read from the cursor resource bits.

`[in] dwVer`

Type: **DWORD**

The version number of the icon or cursor format for the resource bits pointed to by the *presbits* parameter. The value must be greater than or equal to 0x00020000 and less than or equal to 0x00030000. This parameter is generally set to 0x00030000.

`[in] cxDesired`

Type: **int**

The width, in pixels, of the icon or cursor. If this parameter is zero and the *Flags* parameter is **LR_DEFAULTSIZE**, the function uses the **SM_CXICON** or **SM_CXCURSOR** system metric value to set the width. If this parameter is zero and **LR_DEFAULTSIZE** is not used, the function uses the actual resource width.

`[in] cyDesired`

Type: **int**

The height, in pixels, of the icon or cursor. If this parameter is zero and the *Flags* parameter is **LR_DEFAULTSIZE**, the function uses the **SM_CYICON** or **SM_CYCURSOR** system metric value to set the height. If this parameter is zero and **LR_DEFAULTSIZE** is not used, the function uses the actual resource height.

`[in] Flags`

Type: **UINT**

A combination of the following values.

Expand table

Value	Meaning
LR_DEFAULTCOLOR 0x00000000	Uses the default color format.
LR_DEFAULTSIZE 0x00000040	Uses the width or height specified by the system metric values for cursors or icons, if the *cxDesired* or *cyDesired*

	values are set to zero. If this flag is not specified and *cxDesired* and *cyDesired* are set to zero, the function uses the actual resource size.
LR_MONOCHROME 0x00000001	Creates a monochrome icon or cursor.
LR_SHARED 0x00008000	Shares the icon or cursor handle if the icon or cursor is created multiple times. If **LR_SHARED** is not set, a second call to **CreateIconFromResourceEx** for the same resource will create the icon or cursor again and return a different handle. When you use this flag, the system will destroy the resource when it is no longer needed. Do not use **LR_SHARED** for icons or cursors that have non-standard sizes, that may change after loading, or that are loaded from a file.

Return value

Type: **HICON**

If the function succeeds, the return value is a handle to the icon or cursor.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

The CreateIconFromResource, **CreateIconFromResourceEx**, CreateIconIndirect, GetIconInfo, and LookupIconIdFromDirectoryEx functions allow shell applications and icon browsers to examine and use resources throughout the system.

You should call DestroyIcon for icons or DestroyCursor for cursors created with **CreateIconFromResourceEx**.

Examples

For an example, see Sharing Icon Resources.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

BITMAPINFOHEADER

Conceptual

CreateIconFromResource

CreateIconIndirect

DestroyIcon

GetIconInfo

Icons

LoadResource

LookupIconIdFromDirectoryEx

Other Resources

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CreateIconIndirect function (winuser.h)

Article06/20/2023

Creates an icon or cursor from an ICONINFO structure.

Syntax

```
C++

HICON CreateIconIndirect(
  [in] PICONINFO piconinfo
);
```

Parameters

`[in] piconinfo`

Type: **PICONINFO**

A pointer to an ICONINFO structure the function uses to create the icon or cursor.

Return value

Type: **HICON**

If the function succeeds, the return value is a handle to the icon or cursor that is created.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

The system copies the bitmaps in the ICONINFO structure before creating the icon or cursor. Because the system may temporarily select the bitmaps in a device context, the **hbmMask** and **hbmColor** members of the **ICONINFO** structure should not already be selected into a device context. The application must continue to manage the original bitmaps and delete them when they are no longer necessary.

When you are finished using the icon, destroy it using the DestroyIcon function.

Examples

For an example, see Creating a Cursor.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-gui-l1-3-0 (introduced in Windows 10, version 10.0.10240)

See also

Conceptual

DestroyIcon

ICONINFO

Icons

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

DestroyIcon function (winuser.h)

Article02/22/2024

Destroys an icon and frees any memory the icon occupied.

Syntax

```
C++

BOOL DestroyIcon(
  [in] HICON hIcon
);
```

Parameters

`[in] hIcon`

Type: **HICON**

A handle to the icon to be destroyed. The icon must not be in use.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

It is only necessary to call **DestroyIcon** for icons and cursors created with the following functions: CreateIconFromResourceEx (if called without the **LR_SHARED** flag), CreateIconIndirect, and CopyIcon. Do not use this function to destroy a shared icon. A shared icon is valid as long as the module from which it was loaded remains in memory. The following functions obtain a shared icon.

- LoadIcon
- LoadImage (if you use the **LR_SHARED** flag)

- CopyImage (if you use the **LR_COPYRETURNORG** flag and the *hImage* parameter is a shared icon)
- CreateIconFromResource
- CreateIconFromResourceEx (if you use the **LR_SHARED** flag)

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-gui-l1-1-0 (introduced in Windows 8)

See also

Conceptual

CopyIcon

CreateIconFromResource

CreateIconFromResourceEx

CreateIconIndirect

Icons

Reference

Feedback

Was this page helpful?

DrawIcon function (winuser.h)

Article02/22/2024

Draws an icon or cursor into the specified device context.

To specify additional drawing options, use the DrawIconEx function.

Syntax

```
C++

BOOL DrawIcon(
  [in] HDC   hDC,
  [in] int   X,
  [in] int   Y,
  [in] HICON hIcon
);
```

Parameters

`[in] hDC`

Type: **HDC**

A handle to the device context into which the icon or cursor will be drawn.

`[in] X`

Type: **int**

The logical x-coordinate of the upper-left corner of the icon.

`[in] Y`

Type: **int**

The logical y-coordinate of the upper-left corner of the icon.

`[in] hIcon`

Type: **HICON**

A handle to the icon to be drawn.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

DrawIcon places the icon's upper-left corner at the location specified by the *X* and *Y* parameters. The location is subject to the current mapping mode of the device context.

DrawIcon draws the icon or cursor using the width and height specified by the system metric values for icons; for more information, see GetSystemMetrics.

The **DrawIcon** function calls DrawIconEx passing `DI_NORMAL|DI_DEFAULTSIZE` as flags.

Examples

For an example, see Displaying an Icon.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-gui-l1-3-1 (introduced in Windows 10, version 10.0.14393)

See also

Conceptual

CreateIcon

DrawIconEx

Icons

LoadIcon

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

DrawIconEx function (winuser.h)

Article03/11/2023

Draws an icon or cursor into the specified device context, performing the specified raster operations, and stretching or compressing the icon or cursor as specified.

Syntax

```
C++

BOOL DrawIconEx(
  [in]           HDC    hdc,
  [in]           int    xLeft,
  [in]           int    yTop,
  [in]           HICON  hIcon,
  [in]           int    cxWidth,
  [in]           int    cyWidth,
  [in]           UINT   istepIfAniCur,
  [in, optional] HBRUSH hbrFlickerFreeDraw,
  [in]           UINT   diFlags
);
```

Parameters

`[in] hdc`

Type: **HDC**

A handle to the device context into which the icon or cursor will be drawn.

`[in] xLeft`

Type: **int**

The logical x-coordinate of the upper-left corner of the icon or cursor.

`[in] yTop`

Type: **int**

The logical y-coordinate of the upper-left corner of the icon or cursor.

`[in] hIcon`

Type: **HICON**

A handle to the icon or cursor to be drawn. This parameter can identify an animated cursor.

`[in] cxWidth`

Type: **int**

The logical width of the icon or cursor. If this parameter is zero and the *diFlags* parameter is **DI_DEFAULTSIZE**, the function uses the **SM_CXICON** system metric value to set the width. If this parameter is zero and **DI_DEFAULTSIZE** is not used, the function uses the actual resource width.

`[in] cyWidth`

Type: **int**

The logical height of the icon or cursor. If this parameter is zero and the *diFlags* parameter is **DI_DEFAULTSIZE**, the function uses the **SM_CYICON** system metric value to set the width. If this parameter is zero and **DI_DEFAULTSIZE** is not used, the function uses the actual resource height.

`[in] istepIfAniCur`

Type: **UINT**

The index of the frame to draw, if *hIcon* identifies an animated cursor. This parameter is ignored if *hIcon* does not identify an animated cursor.

`[in, optional] hbrFlickerFreeDraw`

Type: **HBRUSH**

A handle to a brush that the system uses for flicker-free drawing. If *hbrFlickerFreeDraw* is a valid brush handle, the system creates an offscreen bitmap using the specified brush for the background color, draws the icon or cursor into the bitmap, and then copies the bitmap into the device context identified by *hdc*. If *hbrFlickerFreeDraw* is **NULL**, the system draws the icon or cursor directly into the device context.

`[in] diFlags`

Type: **UINT**

The drawing flags. This parameter can be one of the following values.

Expand table

Value	Meaning
DI_COMPAT 0x0004	This flag is ignored.
DI_DEFAULTSIZE 0x0008	Draws the icon or cursor using the width and height specified by the system metric values for icons, if the *cxWidth* and *cyWidth* parameters are set to zero. If this flag is not specified and *cxWidth* and *cyWidth* are set to zero, the function uses the actual resource size.
DI_IMAGE 0x0002	Draws the icon or cursor using the image. See Remarks.
DI_MASK 0x0001	Draws the icon or cursor using the mask. See Remarks.
DI_NOMIRROR 0x0010	Draws the icon as an unmirrored icon. By default, the icon is drawn as a mirrored icon if *hdc* is mirrored.
DI_NORMAL 0x0003	Combination of **DI_IMAGE** and **DI_MASK**. See Remarks.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The **DrawIconEx** function places the icon's upper-left corner at the location specified by the *xLeft* and *yTop* parameters. The location is subject to the current mapping mode of the device context.

If only one of the **DI_IMAGE** and **DI_MASK** flags is set, then the corresponding bitmap is drawn with the **SRCCOPY** raster operation code.

If both the **DI_IMAGE** and **DI_MASK** flags are set:

- If the icon or cursor is a 32-bit alpha-blended icon or cursor, then the image is drawn with **AC_SRC_OVER** blend function and the mask is ignored.
- For all other icons or cursors, the mask is drawn with the **SRCAND** raster operation code, and the image is drawn with the **SRCINVERT** raster operation code

To duplicate `DrawIcon (hDC, X, Y, hIcon)`, call **DrawIconEx** as follows:

syntax

```
DrawIconEx (hDC, X, Y, hIcon, 0, 0, 0, NULL, DI_NORMAL | DI_COMPAT |
DI_DEFAULTSIZE);
```

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-gui-l1-1-0 (introduced in Windows 8)

See also

Conceptual

CopyImage

DrawIcon

Icons

LoadImage

Reference

BitBlt

AlphaBlend

BLENDFUNCTION

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

DuplicateIcon function (shellapi.h)

Article02/22/2024

Creates a duplicate of a specified icon.

Syntax

```
C++

HICON DuplicateIcon(
  [in] HINSTANCE hInst,
  [in] HICON     hIcon
);
```

Parameters

`[in] hInst`

Type: **HINSTANCE**

`[in] hIcon`

Type: **HICON**

Handle to the icon to be duplicated.

Return value

Type: **HICON**

If successful, the function returns the handle to the new icon that was created; otherwise, **NULL**.

Remarks

When it is no longer needed, the caller is responsible for freeing the icon handle returned by **DuplicateIcon** by calling the DestroyIcon function.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	shellapi.h
DLL	Shell32.dll
API set	ext-ms-win-shell-shell32-l1-2-1 (introduced in Windows 10, version 10.0.10240)

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

ExtractAssociatedIconA function (shellapi.h)

Article02/09/2023

Gets a handle to an icon stored as a resource in a file or an icon stored in a file's associated executable file.

Syntax

```
C++

HICON ExtractAssociatedIconA(
  [in]      HINSTANCE hInst,
  [in, out] LPSTR     pszIconPath,
  [in, out] WORD      *piIcon
);
```

Parameters

`[in] hInst`

Type: **HINSTANCE**

A handle to the instance of the calling application.

`[in, out] pszIconPath`

Type: **LPTSTR**

Pointer to a string that, on entry, specifies the full path and file name of the file that contains the icon. The function extracts the icon handle from that file, or from an executable file associated with that file.

When this function returns, if the icon handle was obtained from an executable file (either an executable file pointed to by *lpIconPath* or an associated executable file) the function stores the full path and file name of that executable in the buffer pointed to by this parameter.

`[in, out] piIcon`

Type: **LPWORD**

Pointer to a **WORD** value that, on entry, specifies the index of the icon whose handle is to be obtained.

When the function returns, if the icon handle was obtained from an executable file (either an executable file pointed to by *lpIconPath* or an associated executable file), this value points to the icon's index in that file.

Return value

Type: **HICON**

If the function succeeds, the return value is an icon handle. If the icon is extracted from an associated executable file, the function stores the full path and file name of the executable file in the string pointed to by *lpIconPath*, and stores the icon's identifier in the **WORD** pointed to by *lpiIcon*.

If the function fails, the return value is **NULL**.

Remarks

When it is no longer needed, the caller is responsible for freeing the icon handle returned by **ExtractAssociatedIcon** by calling the DestroyIcon function.

The **ExtractAssociatedIcon** function first looks for the indexed icon in the file specified by *lpIconPath*. If the function cannot obtain the icon handle from that file, and the file has an associated executable file, it looks in that executable file for an icon. Associations with executable files are based on file name extensions and are stored in the per-user part of the registry.

> **Note**
>
> The shellapi.h header defines ExtractAssociatedIcon as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	shellapi.h
DLL	Shell32.dll
API set	ext-ms-win-shell-shell32-l1-2-1 (introduced in Windows 10, version 10.0.10240)

See also

ExtractAssociatedIconEx

ExtractIcon

ExtractIconEx

Feedback

ExtractIconA function (shellapi.h)

Article02/09/2023

Gets a handle to an icon from the specified executable file, DLL, or icon file.

To retrieve an array of handles to large or small icons, use the ExtractIconEx function.

Syntax

```
C++

HICON ExtractIconA(
  [in] HINSTANCE hInst,
  [in] LPCSTR    pszExeFileName,
       UINT      nIconIndex
);
```

Parameters

`[in] hInst`

Type: **HINSTANCE**

Handle to the instance of the application that calls the function.

`[in] pszExeFileName`

Type: **LPCTSTR**

Pointer to a null-terminated string that specifies the name of an executable file, DLL, or icon file.

`nIconIndex`

Type: **UINT**

Specifies the zero-based index of the icon to retrieve. For example, if this value is 0, the function returns a handle to the first icon in the specified file.

If this value is -1, the function returns the total number of icons in the specified file. If the file is an executable file or DLL, the return value is the number of RT_GROUP_ICON resources. If the file is an .ICO file, the return value is 1.

If this value is a negative number not equal to –1, the function returns a handle to the icon in the specified file whose resource identifier is equal to the absolute value of *nIconIndex*. For example, you should use –3 to extract the icon whose resource identifier is 3. To extract the icon whose resource identifier is 1, use the ExtractIconEx function.

Return value

Type: **HICON**

The return value is a handle to an icon. If the file specified was not an executable file, DLL, or icon file, the return is 1. If no icons were found in the file, the return value is **NULL**.

Remarks

When it is no longer needed, you must destroy the icon handle returned by **ExtractIcon** by calling the DestroyIcon function.

> **Note**
>
> The shellapi.h header defines ExtractIcon as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	shellapi.h

Requirement	Value
DLL	Shell32.dll
API set	ext-ms-win-shell-shell32-l1-2-1 (introduced in Windows 10, version 10.0.10240)

See also

ExtractAssociatedIcon

ExtractAssociatedIconEx

ExtractIconEx

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

ExtractIconExA function (shellapi.h)

Article02/22/2024

The **ExtractIconEx** function creates an array of handles to large or small icons extracted from the specified executable file, DLL, or icon file.

Syntax

```
C++

UINT ExtractIconExA(
  [in]  LPCSTR lpszFile,
  [in]  int    nIconIndex,
  [out] HICON  *phiconLarge,
  [out] HICON  *phiconSmall,
        UINT   nIcons
);
```

Parameters

`[in] lpszFile`

Type: **LPCTSTR**

Pointer to a null-terminated string that specifies the name of an executable file, DLL, or icon file from which icons will be extracted.

`[in] nIconIndex`

Type: **int**

Specifies the zero-based index of the first icon to extract. For example, if this value is zero, the function extracts the first icon in the specified file.

If this value is –1 and *phiconLarge* and *phiconSmall* are both **NULL**, the function returns the total number of icons in the specified file. If the file is an executable file or DLL, the return value is the number of RT_GROUP_ICON resources. If the file is an .ico file, the return value is 1.

If this value is a negative number and either *phiconLarge* or *phiconSmall* is not **NULL**, the function begins by extracting the icon whose resource identifier is equal to the absolute

value of *nIconIndex*. For example, use -3 to extract the icon whose resource identifier is 3.

`[out] phiconLarge`

Type: **HICON***

Pointer to an array of icon handles that receives handles to the large icons extracted from the file. If this parameter is **NULL**, no large icons are extracted from the file.

`[out] phiconSmall`

Type: **HICON***

Pointer to an array of icon handles that receives handles to the small icons extracted from the file. If this parameter is **NULL**, no small icons are extracted from the file.

`nIcons`

Type: **UINT**

The number of icons to extract from the file.

Return value

Type: **UINT**

If the *nIconIndex* parameter is -1 and both the *phiconLarge* and *phiconSmall* parameters are **NULL**, then the return value is the number of icons contained in the specified file.

If the *nIconIndex* parameter is any value other than -1 and either *phiconLarge* or *phiconSmall* is not **NULL**, the return value is the number of icons successfully extracted from the file.

> ⓘ **Note**
>
> If the function encounters an error, it returns **UINT_MAX**. In this case, you can call **GetLastError** to retrieve the error code. For example, this function returns **UINT_MAX** if the file specified by *lpszFile* cannot be found while the *nIconIndex* parameter is any value other than -1 and either *phiconLarge* or *phiconSmall* is not **NULL**. In this case, **GetLastError** returns **ERROR_FILE_NOT_FOUND** (2).

Remarks

When they are no longer needed, you must destroy all icons extracted by **ExtractIconEx** by calling the DestroyIcon function.

To retrieve the dimensions of the large and small icons, use this function with the SM_CXICON, SM_CYICON, SM_CXSMICON, and SM_CYSMICON flags.

> **Note**
>
> The shellapi.h header defines ExtractIconEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	shellapi.h
DLL	Shell32.dll
API set	ext-ms-win-shell-shell32-l1-2-1 (introduced in Windows 10, version 10.0.10240)

See also

ExtractAssociatedIcon

ExtractAssociatedIconEx

ExtractIcon

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetIconInfo function (winuser.h)

Article06/20/2023

Retrieves information about the specified icon or cursor.

Syntax

```
C++

BOOL GetIconInfo(
  [in]  HICON     hIcon,
  [out] PICONINFO piconinfo
);
```

Parameters

`[in] hIcon`

Type: **HICON**

A handle to the icon or cursor.

To retrieve information about a standard icon or cursor, specify the identifier beginning with the IDI_ prefix or the identifier beginning with the IDC_ prefix in this parameter.

`[out] piconinfo`

Type: **PICONINFO**

A pointer to an ICONINFO structure. The function fills in the structure's members.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero and the function fills in the members of the specified ICONINFO structure.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

GetIconInfo creates bitmaps for the **hbmMask** and **hbmColor** or members of ICONINFO. The calling application must manage these bitmaps and delete them with DeleteObject call when they are no longer necessary.

DPI Virtualization

This API does not participate in DPI virtualization. The output returned is not affected by the DPI of the calling thread.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-gui-l1-1-0 (introduced in Windows 8)

See also

Conceptual

Bitmaps

Icons

DeleteObject

GetObject

BITMAP

CreateIcon

CreateIconFromResource

CreateIconIndirect

DestroyIcon

DrawIcon

DrawIconEx

LoadIcon

LookupIconIdFromDirectory

ICONINFO

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetIconInfoExA function (winuser.h)

Article02/22/2024

Retrieves information about the specified icon or cursor. **GetIconInfoEx** extends GetIconInfo by using the newer ICONINFOEX structure.

Syntax

```
C++

BOOL GetIconInfoExA(
  [in]      HICON        hicon,
  [in, out] PICONINFOEXA piconinfo
);
```

Parameters

`[in] hicon`

Type: **HICON**

A handle to the icon or cursor.

To retrieve information about a standard icon or cursor, specify the identifier beginning with the IDI_ prefix or the identifier beginning with the IDC_ prefix in this parameter.

`[in, out] piconinfo`

Type: **PICONINFOEX**

When this method returns, contains a pointer to an ICONINFOEX structure. The function fills in the structure's members.

Return value

Type: **BOOL**

TRUE indicates success, **FALSE** indicates failure.

Remarks

GetIconInfoEx creates bitmaps for the **hbmMask** and **hbmColor** or members of ICONINFOEX. The calling application must manage these bitmaps and delete them with DeleteObject call when they are no longer necessary.

DPI Virtualization

This API does not participate in DPI virtualization. The output returned is not affected by the DPI of the calling thread.

> **Note**
>
> The winuser.h header defines GetIconInfoEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

Bitmaps

Icons

DeleteObject

GetObject

BITMAP

CreateIcon

CreateIconFromResource

CreateIconIndirect

DestroyIcon

DrawIcon

DrawIconEx

LoadIcon

LookupIconIdFromDirectory

ICONINFO

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

LoadIconA function (winuser.h)

Article07/20/2023

Loads the specified icon resource from the executable (.exe) file associated with an application instance.

> ⓘ **Note**
>
> This function has been superseded by the **LoadImage** function (with **LR_DEFAULTSIZE** and **LR_SHARED** flags set).

Syntax

```
C++

HICON LoadIconA(
  [in, optional] HINSTANCE hInstance,
  [in]           LPCSTR    lpIconName
);
```

Parameters

`[in, optional] hInstance`

Type: **HINSTANCE**

A handle to the module of either a DLL or executable (.exe) file that contains the icon to be loaded. For more information, see GetModuleHandle.

To load a predefined system icon, set this parameter to **NULL**.

`[in] lpIconName`

Type: **LPCTSTR**

If *hInstance* is non-**NULL**, *lpIconName* specifies the icon resource either by name or ordinal. This ordinal must be packaged by using the MAKEINTRESOURCE macro.

If *hInstance* is **NULL**, *lpIconName* specifies the identifier (beginning with the IDI_ prefix) of a predefined system icon to load.

Return value

Type: **HICON**

If the function succeeds, the return value is a handle to the newly loaded icon.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

LoadIcon loads the icon resource only if it has not been loaded; otherwise, it retrieves a handle to the existing resource. The function searches the icon resource for the icon most appropriate for the current display. The icon resource can be a color or monochrome bitmap.

LoadIcon can only load an icon whose size conforms to the **SM_CXICON** and **SM_CYICON** system metric values. Use the LoadImage function to load icons of other sizes.

> **Note**
>
> The winuser.h header defines LoadIcon as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

Requirement	Value
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-gui-l1-1-0 (introduced in Windows 8)

See also

Conceptual

CreateIcon

Icons

LoadImage

MAKEINTRESOURCE

IS_INTRESOURCE

Feedback

LookupIconIdFromDirectory function (winuser.h)

Article05/24/2023

Searches through icon (**RT_GROUP_ICON**) or cursor (**RT_GROUP_CURSOR**) resource data for the icon or cursor that best fits the current display device.

To specify a desired height or width, use the LookupIconIdFromDirectoryEx function. This function calls it by passing zero in the **cxDesired/cyDesired** parameters.

Syntax

```
C++

int LookupIconIdFromDirectory(
  [in] PBYTE presbits,
  [in] BOOL  fIcon
);
```

Parameters

`[in] presbits`

Type: **PBYTE**

The icon or cursor directory data. Because this function does not validate the resource data, it causes a general protection (GP) fault or returns an undefined value if *presbits* is not pointing to valid resource data.

`[in] fIcon`

Type: **BOOL**

Indicates whether an icon or a cursor is sought. If this parameter is **TRUE**, the function is searching for an icon; if the parameter is **FALSE**, the function is searching for a cursor.

Return value

Type: **int**

If the function succeeds, the return value is an integer resource identifier for the icon (**RT_ICON**) or cursor (**RT_CURSOR**) that best fits the current display device.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

A resource file of type **RT_GROUP_ICON** (**RT_GROUP_CURSOR** indicates cursors) contains icon (or cursor) data in several device-dependent and device-independent formats. **LookupIconIdFromDirectory** searches the resource file for the icon (or cursor) that best fits the current display device and returns its integer identifier. The FindResource and FindResourceEx functions use the MAKEINTRESOURCE macro with this identifier to locate the resource in the module.

The icon directory is loaded from a resource file with resource type **RT_GROUP_ICON** (or **RT_GROUP_CURSOR** for cursors), and an integer resource name for the specific icon to be loaded. **LookupIconIdFromDirectory** returns an integer identifier that is the resource name of the icon that best fits the current display device.

The LoadIcon, LoadCursor, and LoadImage functions use this function to search the specified resource data for the icon or cursor that best fits the current display device.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

CreateIconFromResource

CreateIconIndirect

FindResource

FindResourceEx

GetIconInfo

Icons

LoadCursor

LoadIcon

LoadImage

LookupIconIdFromDirectoryEx

MAKEINTRESOURCE

Reference

Feedback

LookupIconIdFromDirectoryEx function (winuser.h)

Article02/22/2024

Searches through icon (**RT_GROUP_ICON**) or cursor (**RT_GROUP_CURSOR**) resource data for the icon or cursor that best fits the current display device.

If more than one image exists in resource group, this method uses the following criteria to choose an image:

- The image closest but not exceed the requested size is selected.
- If two or more images of that size are present, the one that matches the color depth of the display is chosen.
- If no images exactly match the color depth of the display, the image with the greatest color depth that does not exceed the color depth of the display is chosen. If all exceed the color depth, the one with the lowest color depth is chosen.

Syntax

C++

```
int LookupIconIdFromDirectoryEx(
  [in] PBYTE presbits,
  [in] BOOL  fIcon,
  [in] int   cxDesired,
  [in] int   cyDesired,
  [in] UINT  Flags
);
```

Parameters

`[in] presbits`

Type: **PBYTE**

The icon or cursor directory data. Because this function does not validate the resource data, it causes a general protection (GP) fault or returns an undefined value if *presbits* is not pointing to valid resource data.

`[in] fIcon`

Type: **BOOL**

Indicates whether an icon or a cursor is sought. If this parameter is **TRUE**, the function is searching for an icon; if the parameter is **FALSE**, the function is searching for a cursor.

`[in] cxDesired`

Type: **int**

The desired width, in pixels, of the icon. If this parameter is zero, the function uses the **SM_CXICON** or **SM_CXCURSOR** system metric value.

`[in] cyDesired`

Type: **int**

The desired height, in pixels, of the icon. If this parameter is zero, the function uses the **SM_CYICON** or **SM_CYCURSOR** system metric value.

`[in] Flags`

Type: **UINT**

A combination of the following values.

Expand table

Value	Meaning
LR_DEFAULTCOLOR 0x00000000	Uses the default color format.
LR_MONOCHROME 0x00000001	Creates a monochrome icon or cursor.

Return value

Type: **int**

If the function succeeds, the return value is an integer resource identifier for the icon (**RT_ICON**) or cursor (**RT_CURSOR**) that best fits the current display device.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

A resource file of type **RT_GROUP_ICON** (**RT_GROUP_CURSOR** indicates cursors) contains icon (or cursor) data in several device-dependent and device-independent formats. **LookupIconIdFromDirectoryEx** searches the resource file for the icon (or cursor) that best fits the current display device and returns its integer identifier. The FindResource and FindResourceEx functions use the MAKEINTRESOURCE macro with this identifier to locate the resource in the module.

The icon directory is loaded from a resource file with resource type **RT_GROUP_ICON** (or **RT_GROUP_CURSOR** for cursors), and an integer resource name for the specific icon (**RT_ICON**) or cursor (**RT_CURSOR**) to be loaded. LoadResource and CreateIconFromResourceEx functions may be used to create a corresponding icon or cursor.

The LoadIcon, LoadImage, and LoadCursor functions use this function to search the specified resource data for the icon or cursor that best fits the current display device. LoadIconWithScaleDown uses alternative search criteria for a best fit.

Examples

For an example, see Sharing Icon Resources.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

CreateIconFromResourceEx

CreateIconIndirect

FindResource

FindResourceEx

GetIconInfo

Icons

LoadCursor

LoadIcon

LoadImage

LookupIconIdFromDirectory

MAKEINTRESOURCE

Reference

Feedback

PrivateExtractIconsA function (winuser.h)

Article02/09/2023

[This function is not intended for general use. It may be altered or unavailable in subsequent versions of Windows.]

Creates an array of handles to icons that are extracted from a specified file.

Syntax

```
C++

UINT PrivateExtractIconsA(
  [in]            LPCSTR szFileName,
  [in]            int    nIconIndex,
  [in]            int    cxIcon,
  [in]            int    cyIcon,
  [out, optional] HICON  *phicon,
  [out, optional] UINT   *piconid,
  [in]            UINT   nIcons,
  [in]            UINT   flags
);
```

Parameters

`[in] szFileName`

Type: **LPCTSTR**

The path and name of the file from which the icon(s) are to be extracted.

`[in] nIconIndex`

Type: **int**

The zero-based index of the first icon to extract. For example, if this value is zero, the function extracts the first icon in the specified file.

`[in] cxIcon`

Type: **int**

The horizontal icon size wanted. See Remarks.

`[in] cyIcon`

Type: **int**

The vertical icon size wanted. See Remarks.

`[out, optional] phicon`

Type: **HICON***

A pointer to the returned array of icon handles.

`[out, optional] piconid`

Type: **UINT***

A pointer to a returned resource identifier for the icon that best fits the current display device. The returned identifier is 0xFFFFFFFF if the identifier is not available for this format. The returned identifier is 0 if the identifier cannot otherwise be obtained.

`[in] nIcons`

Type: **UINT**

The number of icons to extract from the file. This parameter is only valid when extracting from .exe and .dll files.

`[in] flags`

Type: **UINT**

Specifies flags that control this function. These flags are the LR_* flags used by the LoadImage function.

Return value

Type: **UINT**

If the *phicon* parameter is **NULL** and this function succeeds, then the return value is the number of icons in the file. If the function fails then the return value is 0.

If the *phicon* parameter is not **NULL** and the function succeeds, then the return value is the number of icons extracted. Otherwise, the return value is 0xFFFFFFFF if the file is not found.

Remarks

This function extracts from executable (.exe), DLL (.dll), icon (.ico), cursor (.cur), animated cursor (.ani), and bitmap (.bmp) files. Extractions from Windows 3.x 16-bit executables (.exe or .dll) are also supported.

The *cxIcon* and *cyIcon* parameters specify the size of the icons to extract. Two sizes can be extracted by putting the first size in the LOWORD of the parameter and the second size in the HIWORD. For example, `MAKELONG(24, 48)` for both the cxIcon and cyIcon parameters would extract both 24 and 48 size icons.

You must destroy all icons extracted by **PrivateExtractIcons** by calling the DestroyIcon function.

This function was not included in the SDK headers and libraries until Windows XP Service Pack 1 (SP1) and Windows Server 2003. If you do not have a header file and import library for this function, you can call the function using LoadLibrary and GetProcAddress.

> **Note**
>
> The winuser.h header defines PrivateExtractIcons as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h

Requirement	Value
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-misc-l1-5-1 (introduced in Windows 10, version 10.0.14393)

See also

Conceptual

DestroyIcon

ExtractIcon

ExtractIconEx

Icons

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Icon Structures

Article • 04/27/2021

In This Section

- ICONINFO
- ICONINFOEX
- ICONMETRICS

Feedback

ICONINFO structure (winuser.h)

Article02/22/2024

Contains information about an icon or a cursor.

Syntax

```
C++
```

```
typedef struct _ICONINFO {
  BOOL    fIcon;
  DWORD   xHotspot;
  DWORD   yHotspot;
  HBITMAP hbmMask;
  HBITMAP hbmColor;
} ICONINFO;
```

Members

`fIcon`

Type: **BOOL**

Specifies whether this structure defines an icon or a cursor. A value of **TRUE** specifies an icon; **FALSE** specifies a cursor.

`xHotspot`

Type: **DWORD**

The x-coordinate of a cursor's hot spot. If this structure defines an icon, the hot spot is always in the center of the icon, and this member is ignored.

`yHotspot`

Type: **DWORD**

The y-coordinate of the cursor's hot spot. If this structure defines an icon, the hot spot is always in the center of the icon, and this member is ignored.

`hbmMask`

Type: **HBITMAP**

A handle to the icon monochrome mask bitmap.

`hbmColor`

Type: **HBITMAP**

A handle to the icon color bitmap.

Remarks

For monochrome icons, the **hbmMask** is twice the height of the icon (with the AND mask on top and the XOR mask on the bottom), and **hbmColor** is **NULL**. Also, in this case the height should be an even multiple of two.

For color icons, the **hbmMask** and **hbmColor** bitmaps are the same size, each of which is the size of the icon.

You can use a GetObject function to get contents of **hbmMask** and **hbmColor** in the BITMAP structure. The bitmap bits can be obtained with call to GetDIBits on the bitmaps in this structure.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

CreateIconIndirect

Icons

Bitmaps

GetObject

GetDIBits

BITMAP

Reference

Feedback

ICONINFOEXA structure (winuser.h)

Article05/24/2023

Contains information about an icon or a cursor. Extends ICONINFO. Used by GetIconInfoEx.

Syntax

```
C++

typedef struct _ICONINFOEXA {
  DWORD   cbSize;
  BOOL    fIcon;
  DWORD   xHotspot;
  DWORD   yHotspot;
  HBITMAP hbmMask;
  HBITMAP hbmColor;
  WORD    wResID;
  CHAR    szModName[MAX_PATH];
  CHAR    szResName[MAX_PATH];
} ICONINFOEXA, *PICONINFOEXA;
```

Members

`cbSize`

Type: **DWORD**

The size, in bytes, of this structure.

`fIcon`

Type: **BOOL**

Specifies whether this structure defines an icon or a cursor. A value of **TRUE** specifies an icon; **FALSE** specifies a cursor.

`xHotspot`

Type: **DWORD**

The x-coordinate of a cursor's hot spot. If this structure defines an icon, the hot spot is always in the center of the icon, and this member is ignored.

`yHotspot`

Type: **DWORD**

The y-coordinate of the cursor's hot spot. If this structure defines an icon, the hot spot is always in the center of the icon, and this member is ignored.

`hbmMask`

Type: **HBITMAP**

A handle to the icon monochrome mask bitmap.

`hbmColor`

Type: **HBITMAP**

A handle to the icon color bitmap.

`wResID`

Type: **WORD**

Resource identifier of the resource in **szModName** module. If the icon or cursor was loaded by name, then **wResID** is zero and **szResName** contains the resource name.

You can use MAKEINTRESOURCE(wResID) macro to convert resource identifier to a resource name type compatible with the resource-management functions.

`szModName[MAX_PATH]`

Type: **TCHAR[MAX_PATH]**

Name of the module from which an icon or a cursor was loaded.

You can use GetModuleHandle function to convert it to the module handle compatible with the resource-management functions.

`szResName[MAX_PATH]`

Type: **TCHAR[MAX_PATH]**

Resource name of the resource in **szModName** module.

Remarks

For monochrome icons, the **hbmMask** is twice the height of the icon (with the AND mask on top and the XOR mask on the bottom), and **hbmColor** is **NULL**. Also, in this case the height should be an even multiple of two.

For color icons, the **hbmMask** and **hbmColor** bitmaps are the same size, each of which is the size of the icon.

You can use a GetObject function to get contents of **hbmMask** and **hbmColor** in the BITMAP structure. The bitmap bits can be obtained with call to GetDIBits on the bitmaps in this structure.

ICONINFOEX is an extended version of **ICONINFO** structure with additional **szModName/szResName/wResID** members that can be used to query an icon or cursor resource bits. These bits are typically loaded by calls to the FindResource, LoadResource, LockResource and LookupIconIdFromDirectoryEx functions.

> **Note**
>
> The winuser.h header defines ICONINFOEX as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

CreateIconIndirect

GetIconInfo

Icons

Bitmaps

GetObject

BITMAP

GetDIBits

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

ICONMETRICSA structure (winuser.h)

Article02/22/2024

Contains the scalable metrics associated with icons. This structure is used with the SystemParametersInfo function when the **SPI_GETICONMETRICS** or **SPI_SETICONMETRICS** action is specified.

Syntax

C++

```
typedef struct tagICONMETRICSA {
  UINT     cbSize;
  int      iHorzSpacing;
  int      iVertSpacing;
  int      iTitleWrap;
  LOGFONTA lfFont;
} ICONMETRICSA, *PICONMETRICSA, *LPICONMETRICSA;
```

Members

`cbSize`

Type: **UINT**

The size of the structure, in bytes.

`iHorzSpacing`

Type: **int**

The horizontal space, in pixels, for each arranged icon.

`iVertSpacing`

Type: **int**

The vertical space, in pixels, for each arranged icon.

`iTitleWrap`

Type: **int**

If this member is nonzero, icon titles wrap to a new line. If this member is zero, titles do not wrap.

`lfFont`

Type: **LOGFONT**

The font to use for icon titles.

Remarks

> **Note**
>
> The winuser.h header defines ICONMETRICS as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

Icons

Other Resources

SystemParametersInfo

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Fonts and text metrics

Article • 09/22/2023

This topic discusses the outline fonts provided by Windows, font metric values that may change between versions of Windows, and guidance for how to use font metrics in your desktop apps.

- For info specific to font metrics in DirectWrite, see Text Metrics.
- For details on managing text in apps using GDI, see the topics in Fonts and Text.

For more detailed information on font usage and type specifications, see the Microsoft typography site.

Available fonts

The outline fonts supplied with Windows are delivered as OpenType fonts with TrueType outlines (Windows also supports OpenType fonts in the CFF format). For lists of all fonts supplied by Windows, see Microsoft typography: fonts by product or family. All Windows outline fonts conform to the latest version of the OpenType specification.

For a list of all the current and legacy UI fonts, see Locked font metrics below.

Font modifications

To assure backwards compatibility, fonts are rarely removed from Windows. However, fonts are often modified. Modifications may include adding characters, redrawing existing characters, modifying hints, or adding or modifying support for advanced OpenType features and complex script shaping.

Locked font metrics

Note that some values associated with UI fonts and default fonts used in Microsoft apps are locked. UI fonts are used to render UI elements like captions, dialogs, and menus. Very few changes are made to these fonts, given their high visibility and frequent use. However, because the reported values associated with these fonts are locked, there may be discrepancies between reported and actual font values.

The following reported values are locked for UI and default fonts, and may be inaccurately reported:

- These values from the font's OS/2 table:

    - xAvgCharWidth
    - sTypoLineGap
    - sTypoAscender
    - sTypoDescender
    - usWinAscent
    - usWinDescent
- The unitsPerEm value set in the font's header
- Values from the Vertical Device metrics table (VDMX)
- The advance widths for individual glyphs

Here's a list of the UI fonts shipped with Windows 8.1 (affected by locked values):

Script name	**UI font**
Arabic	Segoe UI
Armenian	Segoe UI
Bangla	Nirmala UI
Bopomofo	Microsoft JhengHei UI
Braille	Segoe UI Symbol
Buginese	Leelawadee UI
Canadian Aboriginal Syllabics	Gadugi
Cherokee	Gadugi
Coptic	Segoe UI Symbol
Chinese (Simplified)	Microsoft YaHei UI
Chinese (Traditional)	Microsoft JhengHei UI
Cyrillic	Segoe UI
Devanagari	Nirmala UI
Deseret	Segoe UI Symbol
Ethiopic	Ebrima
Georgian	Segoe UI
Glagolitic	Segoe UI Symbol
Gothic	Segoe UI Symbol
Greek	Segoe UI

Script name	UI font
Gujarati	Nirmala UI
Gurmukhi	Nirmala UI
Hebrew	Segoe UI
Old Italic	Segoe UI Symbol
Javanese	Javanese Text
Japanese	Meiryo UI
Kannada	Mirmala UI
Khmer	Leelawadee UI
Korean	Malgun Gothic
Lao	Leelawadee UI
Latin	Segoe UI
Malayalam	Nirmala UI
Mongolian	Mongolian Baiti
Myanmar	Myanmar Text
N'Ko	Ebrima
Ogham	Segoe UI Symbol
Ol Chiki	Nirmala UI
Old Turkic	Segoe UI Symbol
Odia	Nirmala UI
Osmanya	Ebrima
Phags-pa	Microsoft PhagsPa
Runic	Segoe UI Symbol
Sora Sompeng	Nirmala UI
Sinhala	Nirmala UI
Syriac	Estrangelo Edessa
Tai Le	Microsoft Tai Le

Script name	UI font
New Tai Lue	Microsoft New Tai Lue
Tamil	Nirmala UI
Telugu	Nirmala UI
Tifinagh	Ebrima
Thaana	MV Boli
Thai	Leelawadee UI
Tibetan	Microsoft Himalaya
Vai	Ebrima
Yi	Microsoft Yi Baiti

Here's a list of the legacy UI fonts which are also affected by locked values:

Script name (legacy)	UI font (legacy)
Bangla	Vrinda
Canadian Aboriginal Syllabics	Euphemia
Cherokee	Plantagenet
Chinese (Simplified)	Microsoft YaHei and SimSun
Chinese (Traditional)	MingLiU and Microsoft JhengHei
Devanagari	Mangal
European languages	Tahoma
Gujarati	Shruti
Gurmukhi	Raavi
Japanese	Meiryo and MS Gothic UI
Kannada	Tunga
Khmer	Khmer
Korean	Gulim

Script name (legacy)	UI font (legacy)
Lao	Lao UI
Malayalam	Kartika
Middle Eastern languages	Tahoma
Odia	Kalinga
Sinhalese	Iskoola Pota
Tamil	Latha and Vijaya
Telugu	Gautami
Thai	Leelawadee and Tahoma

These fonts are used as defaults in Microsoft apps and are also affected by locked values:

- Arial
- Calibri
- Cambria
- Consolas
- Courier New
- MS Mincho
- Times New Roman
- Verdana

Dynamic font metrics

Other than the locked metrics listed above, font values are accurately reported. If a font is changed in a new version of Windows, dynamic font values will differ between the new and old. For example, when a glyph is added to a font, values in the font's header may change. Clipping could result if these values (which include xMin, xMax, yMin, and yMax, and report the minimum and maximum bounding box for glyphs in the font) were locked and didn't report true values.

> ⓘ **Important**
>
> If you use dynamic font values in your app (like those in **TEXTMETRIC**), these values will change if fonts are modified in future versions of Windows. Don't use these

actual values in situations where text must stay static.

Guidelines for using font metrics

- Compute screen metrics and font metrics (e.g., average width) when an app is launched, and use these values to lay out your app. This will provide consistently accurate rendering, and your layout will respond to changes in fonts or accommodate font fallback. For an overview of font fallback and font linking, see Globalization Step by Step: Fonts. See Using Font Fallback for Uniscribe-specific info.
  - To compute a base metric, render representative text for your intended language/script.
  - For controls that just contain a single line of unwrapped text, lay them out to fit the full width of the untrimmed text.
  - For controls with multiple lines, get the total length, divide by the character length, and you've got a solid width to work with. Note that this is trickier for complex scripts where a single 'character' to the reader may be multiple code points.
- Use sTypoAscender, sTypoDescender, and unitsPerEm (from the OS/2 table) to calculate vertical spacing. sTypoAscender is used to determine the optimum offset from the top of a text frame to the first baseline and sTypoDescender determines the optimum offset from the bottom of a text frame to the last baseline.
- If you're using DirectWrite, create a layout using **IDWriteTextLayout**. **IDWriteTextLayout** provides **ascender** + **descender** + **lineGap** in natural layout. You can access these specific values with **DWRITE_FONT_METRICS**. For info on this interface, see Text Formatting and Layout.
- If you're using GDI, render off screen, then inspect the layout (for example, the line length or characters per line) and recalculate the final layout parameters used in actual rendering.
- Don't build layouts statically based on particular values for particular versions of fonts. Actual values may change from release to release.

Related topics

Reference

IDWriteTextLayout

DWRITE_FONT_METRICS

TEXTMETRIC

unitsPerEm

OS/2 table

Vertical Device metrics table (VDMX)

Microsoft typography: fonts by product or family

Conceptual

Text Metrics (DirectWrite)

Fonts and Text (GDI)

Microsoft Typography

Feedback

Was this page helpful? Yes No

Keyboard Accelerators

Article • 08/19/2020

A *keyboard accelerator* (or, simply, accelerator) is a keystroke or combination of keystrokes that generates a WM_COMMAND or WM_SYSCOMMAND message for an application.

In This Section

Name	Description
About Keyboard Accelerators	Discusses keyboard accelerators.
Using Keyboard Accelerators	Discusses tasks that are associated with keyboard accelerators.
Keyboard Accelerator Reference	Contains the API reference.

Keyboard Accelerator Functions

Name	Description
CopyAcceleratorTable	Copies the specified accelerator table. This function is used to obtain the accelerator-table data that corresponds to an accelerator-table handle, or to determine the size of the accelerator-table data.
CreateAcceleratorTable	Creates an accelerator table.
DestroyAcceleratorTable	Destroys an accelerator table.
LoadAccelerators	Loads the specified accelerator table.
TranslateAccelerator	Processes accelerator keys for menu commands. The function translates a WM_KEYDOWN or WM_SYSKEYDOWN message to a WM_COMMAND or WM_SYSCOMMAND message (if there is an entry for the key in the specified accelerator table) and then sends the **WM_COMMAND** or **WM_SYSCOMMAND** message directly to the specified window procedure. TranslateAccelerator does not return until the window procedure has processed the message.

Keyboard Accelerator Messages

Name	Description
WM_CHANGEUISTATE	Sent to indicate that the UI state should be changed.

Name	Description
WM_INITMENU	Sent when a menu is about to become active. It occurs when the user clicks an item on the menu bar or presses a menu key. This allows the application to modify the menu before it is displayed. A window receives this message through its WindowProc function.
WM_QUERYUISTATE	Sent to retrieve the UI state for a window.
WM_UPDATEUISTATE	Sent to change the UI state for the specified window and all its child windows.

Keyboard Accelerator Notifications

Name	Description
WM_INITMENUPOPUP	Sent when a drop-down menu or submenu is about to become active. This allows an application to modify the menu before it is displayed, without changing the entire menu.
WM_MENUCHAR	Sent when a menu is active and the user presses a key that does not correspond to any mnemonic or accelerator key. This message is sent to the window that owns the menu.
WM_MENUSELECT	Sent to a menu's owner window when the user selects a menu item.
WM_SYSCHAR	Posted to the window with the keyboard focus when a WM_SYSKEYDOWN message is translated by the TranslateMessage function. It specifies the character code of a system character key that is, a character key that is pressed while the ALT key is down.
WM_SYSCOMMAND	A window receives this message when the user chooses a command from the **Window** menu or when the user chooses the maximize button, minimize button, restore button, or close button.

Keyboard Accelerator Structures

Name	Description
ACCEL	Defines an accelerator key used in an accelerator table.

Get help at Microsoft Q&A

About Keyboard Accelerators

Article • 06/30/2021

Accelerators are closely related to menus — both provide the user with access to an application's command set. Typically, users rely on an application's menus to learn the command set and then switch over to using accelerators as they become more proficient with the application. Accelerators provide faster, more direct access to commands than menus do. At a minimum, an application should provide accelerators for the more commonly used commands. Although accelerators typically generate commands that exist as menu items, they can also generate commands that have no equivalent menu items.

This section covers the following topics.

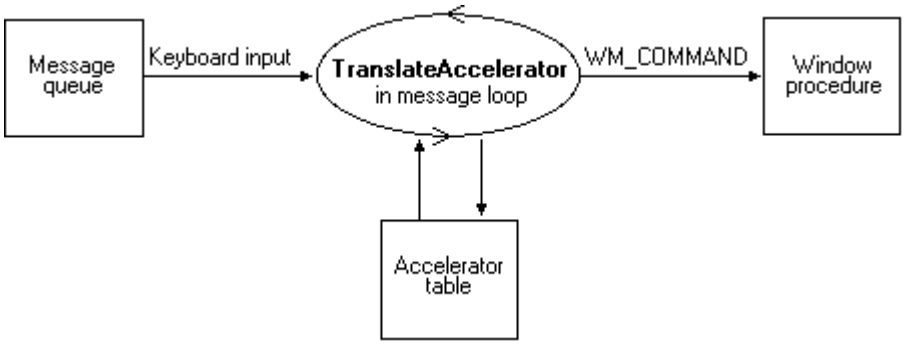
- Accelerator Tables
- Accelerator-Table Creation
- Accelerator Keystroke Assignments
- Accelerators and Menus
- UI State

Accelerator Tables

An accelerator table consists of an array of **ACCEL** structures, each defining an individual accelerator. Each **ACCEL** structure includes the following information:

- The accelerator's keystroke combination.
- The accelerator's identifier.
- Various flags. This includes one that specifies whether the system is to provide visual feedback by highlighting the corresponding menu item, if any, when the accelerator is used

To process accelerator keystrokes for a specified thread, the developer must call the **TranslateAccelerator** function in the message loop associated with the thread's message queue. The **TranslateAccelerator** function monitors keyboard input to the message queue, checking for key combinations that match an entry in the accelerator table. When **TranslateAccelerator** finds a match, it translates the keyboard input (that is, the **WM_KEYUP** and **WM_KEYDOWN** messages) into a **WM_COMMAND** or **WM_SYSCOMMAND** message and then sends the message to the window procedure of the specified window. The following illustration shows how accelerators are processed.



The WM_COMMAND message includes the identifier of the accelerator that caused TranslateAccelerator to generate the message. The window procedure examines the identifier to determine the source of the message and then processes the message accordingly.

Accelerator tables exist at two different levels. The system maintains a single, system-wide accelerator table that applies to all applications. An application cannot modify the system accelerator table. For a description of the accelerators provided by the system accelerator table, see Accelerator Keystroke Assignments.

The system also maintains accelerator tables for each application. An application can define any number of accelerator tables for use with its own windows. A unique 32-bit handle (**HACCEL**) identifies each table. However, only one accelerator table can be active at a time for a specified thread. The handle to the accelerator table passed to the TranslateAccelerator function determines which accelerator table is active for a thread. The active accelerator table can be changed at any time by passing a different accelerator-table handle to **TranslateAccelerator**.

Accelerator-Table Creation

Several steps are required to create an accelerator table for an application. First, a resource compiler is used to create accelerator-table resources and to add them to the application's executable file. At run time, the LoadAccelerators function is used to load the accelerator table into memory and retrieve the handle to the accelerator table. This handle is passed to the TranslateAccelerator function to activate the accelerator table.

An accelerator table can also be created for an application at run time by passing an array of ACCEL structures to the CreateAcceleratorTable function. This method supports user-defined accelerators in the application. Like the LoadAccelerators function, **CreateAcceleratorTable** returns an accelerator-table handle that can be passed to TranslateAccelerator to activate the accelerator table.

The system automatically destroys accelerator tables loaded by LoadAccelerators or created by CreateAcceleratorTable. However, an application can free resources while it

is running by destroying accelerator tables no longer needed by calling the **DestroyAcceleratorTable** function.

An existing accelerator table can be copied and modified. The existing accelerator table is copied by using the **CopyAcceleratorTable** function. After the copy is modified, a handle to the new accelerator table is retrieved by calling **CreateAcceleratorTable**. Finally, the handle is passed to **TranslateAccelerator** to activate the new table.

Accelerator Keystroke Assignments

An ASCII character code or a virtual-key code can be used to define the accelerator. An ASCII character code makes the accelerator case sensitive. Thus, using the ASCII "C" character defines the accelerator as ALT+C rather than ALT+c. However, case-sensitive accelerators can be confusing to use. For example, the ALT+C accelerator will be generated if the CAPS LOCK key is down or if the SHIFT key is down, but not if both are down.

Typically, accelerators don't need to be case sensitive, so most applications use virtual-key codes for accelerators rather than ASCII character codes.

Avoid accelerators that conflict with an application's menu mnemonics, because the accelerator overrides the mnemonic, which can confuse the user. For more information about menu mnemonics, see Menus.

If an application defines an accelerator that is also defined in the system accelerator table, the application-defined accelerator overrides the system accelerator, but only within the context of the application. Avoid this practice, however, because it prevents the system accelerator from performing its standard role in the user interface. The system-wide accelerators are described in the following list:

Accelerator	**Description**
ALT+ESC	Switches to the next application.
ALT+F4	Closes an application or a window.
ALT+HYPHEN	Opens the **Window** menu for a document window.
ALT+PRINT SCREEN	Copies an image in the active window onto the clipboard.
ALT+SPACEBAR	Opens the **Window** menu for the application's main window.
ALT+TAB	Switches to the next application.

Accelerator	Description
CTRL+ESC	Switches to the **Start** menu.
CTRL+F4	Closes the active group or document window.
F1	Starts the application's help file, if one exists.
PRINT SCREEN	Copies an image on the screen onto the clipboard.
SHIFT+ALT+TAB	Switches to the previous application. The user must press and hold down ALT+SHIFT while pressing TAB.

Accelerators and Menus

Using an accelerator is the same as choosing a menu item: Both actions cause the system to send a WM_COMMAND or WM_SYSCOMMAND message to the corresponding window procedure. The **WM_COMMAND** message includes an identifier that the window procedure examines to determine the source of the message. If an accelerator generated the **WM_COMMAND** message, the identifier is that of the accelerator. Similarly, if a menu item generated the **WM_COMMAND** message, the identifier is that of the menu item. Because an accelerator provides a shortcut for choosing a command from a menu, an application usually assigns the same identifier to the accelerator and the corresponding menu item.

An application processes an accelerator WM_COMMAND message in exactly the same way as the corresponding menu item **WM_COMMAND** message. However, the **WM_COMMAND** message contains a flag that specifies whether the message originated from an accelerator or a menu item, in case accelerators must be processed differently from their corresponding menu items. The WM_SYSCOMMAND message does not contain this flag.

The identifier determines whether an accelerator generates a WM_COMMAND or WM_SYSCOMMAND message. If the identifier has the same value as a menu item in the System menu, the accelerator generates a **WM_SYSCOMMAND** message. Otherwise, the accelerator generates a **WM_COMMAND** message.

If an accelerator has the same identifier as a menu item and the menu item is grayed or disabled, the accelerator is disabled and does not generate a WM_COMMAND or WM_SYSCOMMAND message. Also, an accelerator does not generate a command message if the corresponding window is minimized.

When the user uses an accelerator that corresponds to a menu item, the window procedure receives the **WM_INITMENU** and **WM_INITMENUPOPUP** messages as though the user had selected the menu item. For information about how to process these messages, see Menus.

An accelerator that corresponds to a menu item should be included in the text of the menu item.

UI State

Windows enables applications to hide or show various features in its UI. These settings are known as the UI state. The UI state includes the following settings:

- focus indicators (such as focus rectangles on buttons)
- keyboard accelerators (indicated by underlines in control labels)

A window can send messages to request a change in the UI state, can query the UI state, or enforce a certain state for its child windows. These messages are as follows.

Message	Description
WM_CHANGEUISTATE	Indicates that the UI state should change.
WM_QUERYUISTATE	Retrieves the UI state for a window.
WM_UPDATEUISTATE	Changes the UI state.

By default, all child windows of a top-level window are created with the same UI state as their parent.

The system handles the UI state for controls in dialog boxes. At dialog box creation, the system initializes the UI state accordingly. All child controls inherit this state. After the dialog box is created, the system monitors the user's keystrokes. If the UI state settings are hidden and the user navigates using the keyboard, the system updates the UI state. For example, if the user presses the Tab key to move the focus to the next control, the system calls **WM_CHANGEUISTATE** to make the focus indicators visible. If the user presses the Alt key, the system calls **WM_CHANGEUISTATE** to make the keyboard accelerators visible.

If a control supports navigation between the UI elements it contains, it can update its own UI state. The control can call **WM_QUERYUISTATE** to retrieve and cache the initial UI state. Whenever the control receives an **WM_UPDATEUISTATE** message, it can update

its UI state and send a **WM_CHANGEUISTATE** message to its parent. Each window will continue to send the message to its parent until it reaches the top-level window. The top-level window sends the **WM_UPDATEUISTATE** message to the windows in the window tree. If a window does not pass on the **WM_CHANGEUISTATE** message, it will not reach the top-level window and the UI state will not be updated.

Feedback

Using Keyboard Accelerators

Article • 02/15/2022

This section covers tasks that are associated with keyboard accelerators.

Using an Accelerator Table Resource

The most common way to add accelerator support to an application is to include an accelerator-table resource with the application's executable file and then load the resource at run time.

This section covers the following topics.

Creating the Accelerator Table Resource

You create an accelerator-table resource by using the ACCELERATORS statement in your application's resource-definition file. You must assign a name or resource identifier to the accelerator table, preferably unlike that of any other resource. The system uses this identifier to load the resource at run time.

Each accelerator you define requires a separate entry in the accelerator table. In each entry, you define the keystroke (either an ASCII character code or virtual-key code) that generates the accelerator and the accelerator's identifier. You must also specify whether the keystroke must be used in some combination with the ALT, SHIFT, or CTRL keys. For more information about virtual keys, see Keyboard Input.

An ASCII keystroke is specified either by enclosing the ASCII character in double quotation marks or by using the integer value of the character in combination with the ASCII flag. The following examples show how to define ASCII accelerators.

syntax

```
"A", ID_ACCEL1         ; SHIFT+A
65,  ID_ACCEL2, ASCII  ; SHIFT+A
```

A virtual-key code keystroke is specified differently depending on whether the keystroke is an alphanumeric key or a non-alphanumeric key. For an alphanumeric key, the key's letter or number, enclosed in double quotation marks, is combined with the **VIRTKEY** flag. For a non-alphanumeric key, the virtual-key code for the specific key is combined with the **VIRTKEY** flag. The following examples show how to define virtual-key code accelerators.

syntax

```
"a",       ID_ACCEL3, VIRTKEY   ; A (caps-lock on) or a
VK_INSERT, ID_ACCEL4, VIRTKEY   ; INSERT key
```

The following example shows an accelerator-table resource that defines accelerators for file operations. The name of the resource is *FileAccel*.

syntax

```
FileAccel ACCELERATORS
BEGIN
    VK_F12, IDM_OPEN, CONTROL, VIRTKEY  ; CTRL+F12
    VK_F4,  IDM_CLOSE, ALT, VIRTKEY     ; ALT+F4
    VK_F12, IDM_SAVE, SHIFT, VIRTKEY    ; SHIFT+F12
    VK_F12, IDM_SAVEAS, VIRTKEY         ; F12
END
```

If you want the user to press the ALT, SHIFT, or CTRL keys in some combination with the accelerator keystroke, specify the ALT, SHIFT, and CONTROL flags in the accelerator's definition. Following are some examples.

syntax

```
"B",   ID_ACCEL5, ALT                   ; ALT_SHIFT+B
"I",   ID_ACCEL6, CONTROL, VIRTKEY      ; CTRL+I
VK_F5, ID_ACCEL7, CONTROL, ALT, VIRTKEY ; CTRL+ALT+F5
```

By default, when an accelerator key corresponds to a menu item, the system highlights the menu item. You can use the **NOINVERT** flag to prevent highlighting for an individual accelerator. The following example shows how to use the **NOINVERT** flag:

syntax

```
VK_DELETE, ID_ACCEL8, VIRTKEY, SHIFT, NOINVERT  ; SHIFT+DELETE
```

To define accelerators that correspond to menu items in your application, include the accelerators in the text of the menu items. The following example shows how to include accelerators in menu-item text in a resource-definition file.

syntax

```
FilePopup MENU
BEGIN
    POPUP   "&File"
    BEGIN
        MENUITEM    "&New..",           IDM_NEW
        MENUITEM    "&Open\tCtrl+F12",  IDM_OPEN
        MENUITEM    "&Close\tAlt+F4"    IDM_CLOSE
        MENUITEM    "&Save\tShift+F12", IDM_SAVE
        MENUITEM    "Save &As...\tF12", IDM_SAVEAS
    END
END
```

Loading the Accelerator Table Resource

An application loads an accelerator-table resource by calling the LoadAccelerators function and specifying the instance handle to the application whose executable file contains the resource and the name or identifier of the resource. **LoadAccelerators** loads the specified accelerator table into memory and returns the handle to the accelerator table.

An application can load an accelerator-table resource at any time. Usually, a single-threaded application loads its accelerator table before entering its main message loop. An application that uses multiple threads typically loads the accelerator-table resource for a thread before entering the message loop for the thread. An application or thread

might also use multiple accelerator tables, each associated with a particular window in the application. Such an application would load the accelerator table for the window each time the user activated the window. For more information about threads, see Processes and Threads.

Calling the Translate Accelerator Function

To process accelerators, an application's (or thread's) message loop must contain a call to the **TranslateAccelerator** function. **TranslateAccelerator** compares keystrokes to an accelerator table and, if it finds a match, translates the keystrokes into a **WM_COMMAND** (or **WM_SYSCOMMAND**) message. The function then sends the message to a window procedure. The parameters of the **TranslateAccelerator** function include the handle to the window that is to receive the **WM_COMMAND** messages, the handle to the accelerator table used to translate accelerators, and a pointer to an **MSG** structure containing a message from the queue. The following example shows how to call **TranslateAccelerator** from within a message loop.

C++

```
MSG msg;
BOOL bRet;

while ( (bRet = GetMessage(&msg, (HWND) NULL, 0, 0)) != 0)
{
    if (bRet == -1)
    {
        // handle the error and possibly exit
    }
    else
    {
        // Check for accelerator keystrokes.

        if (!TranslateAccelerator(
                hwndMain,      // handle to receiving window
                haccel,        // handle to active accelerator table
                &msg))         // message data
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
}
```

Processing WM_COMMAND Messages

When an accelerator is used, the window specified in the TranslateAccelerator function receives a WM_COMMAND or WM_SYSCOMMAND message. The low-order word of the *wParam* parameter contains the identifier of the accelerator. The window procedure examines the identifier to determine the source of the **WM_COMMAND** message and process the message accordingly.

Typically, if an accelerator corresponds to a menu item in the application, the accelerator and menu item are assigned the same identifier. If you need to know whether a WM_COMMAND message was generated by an accelerator or by a menu item, you can examine the high-order word of the *wParam* parameter. If an accelerator generated the message, the high-order word is 1; if a menu item generated the message, the high-order word is 0.

Destroying the Accelerator Table Resource

The system automatically destroys accelerator-table resources loaded by the LoadAccelerators function, removing the resource from memory after the application closes.

Creating Accelerators for Font Attributes

The example in this section shows how to perform the following tasks:

- Create an accelerator-table resource.
- Load the accelerator table at run time.
- Translate accelerators in a message loop.
- Process WM_COMMAND messages generated by the accelerators.

These tasks are demonstrated in the context of an application that includes a **Character** menu and corresponding accelerators that allow the user to select attributes of the current font.

The following portion of a resource-definition file defines the **Character** menu and the associated accelerator table. Note that the menu items show the accelerator keystrokes and that each accelerator has the same identifier as its associated menu item.

```
#include <windows.h>
#include "acc.h"

MainMenu MENU
{
    POPUP   "&Character"
```

```
    {
        MENUITEM    "&Regular\tF5",         IDM_REGULAR
        MENUITEM    "&Bold\tCtrl+B",        IDM_BOLD
        MENUITEM    "&Italic\tCtrl+I",      IDM_ITALIC
        MENUITEM    "&Underline\tCtrl+U",   IDM_ULINE
    }
}

FontAccel ACCELERATORS
{
    VK_F5,  IDM_REGULAR,    VIRTKEY
    "B",    IDM_BOLD,       CONTROL, VIRTKEY
    "I",    IDM_ITALIC,     CONTROL, VIRTKEY
    "U",    IDM_ULINE,      CONTROL, VIRTKEY
}
```

The following sections from the application's source file show how to implement the accelerators.

C++

```
HWND hwndMain;      // handle to main window
HANDLE hinstAcc;    // handle to application instance
int WINAPI WinMain(HINSTANCE hinst, HINSTANCE hinstPrev, LPSTR lpCmdLine,
int nCmdShow)
{
    MSG msg;            // application messages
    BOOL bRet;          // for return value of GetMessage
    HACCEL haccel;      // handle to accelerator table

    // Perform the initialization procedure.

    // Create a main window for this application instance.

    hwndMain = CreateWindowEx(0L, "MainWindowClass",
        "Sample Application", WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL,
        hinst, NULL );

    // If a window cannot be created, return "failure."

    if (!hwndMain)
        return FALSE;

    // Make the window visible and update its client area.

    ShowWindow(hwndMain, nCmdShow);
    UpdateWindow(hwndMain);

    // Load the accelerator table.

    haccel = LoadAccelerators(hinstAcc, "FontAccel");
```

```
    if (haccel == NULL)
        HandleAccelErr(ERR_LOADING);     // application defined

    // Get and dispatch messages until a WM_QUIT message is
    // received.

    while ((bRet = GetMessage(&msg, NULL, 0, 0)) != 0)
    {
        if (bRet == -1)
        {
            // handle the error and possibly exit
        }
        else
        {
            // Check for accelerator keystrokes.

            if (!TranslateAccelerator(
                    hwndMain,  // handle to receiving window
                    haccel,    // handle to active accelerator table
                    &msg))         // message data
            {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
            }
        }
    }
    return msg.wParam;
}

LRESULT APIENTRY MainWndProc(HWND hwndMain, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{
    BYTE fbFontAttrib;        // array of font-attribute flags
    static HMENU hmenu;       // handle to main menu

    switch (uMsg)
    {
        case WM_CREATE:

            // Add a check mark to the Regular menu item to
            // indicate that it is the default.

            hmenu = GetMenu(hwndMain);
            CheckMenuItem(hmenu, IDM_REGULAR, MF_BYCOMMAND |
                MF_CHECKED);
            return 0;

        case WM_COMMAND:
            switch (LOWORD(wParam))
            {
                // Process the accelerator and menu commands.

                case IDM_REGULAR:
                case IDM_BOLD:
                case IDM_ITALIC:
```

```
                case IDM_ULINE:

                    // GetFontAttributes is an application-defined
                    // function that sets the menu-item check marks
                    // and returns the user-selected font attributes.

                    fbFontAttrib = GetFontAttributes(
                        (BYTE) LOWORD(wParam), hmenu);

                    // SetFontAttributes is an application-defined
                    // function that creates a font with the
                    // user-specified attributes the font with
                    // the main window's device context.

                    SetFontAttributes(fbFontAttrib);
                    break;

                default:
                    break;
            }
            break;

            // Process other messages.

        default:
            return DefWindowProc(hwndMain, uMsg, wParam, lParam);
    }
    return NULL;
}
```

Using an Accelerator Table Created at Run Time

This topic discusses how to use accelerator tables created at run time.

- Creating a Run-Time Accelerator Table
- Processing Accelerators
- Destroying a Run-Time Accelerator Table
- Creating User Editable Accelerators

Creating a Run-Time Accelerator Table

The first step in creating an accelerator table at run time is filling an array of **ACCEL** structures. Each structure in the array defines an accelerator in the table. An accelerator's definition includes its flags, its key, and its identifier. The **ACCEL** structure has the following form.

syntax

```
typedef struct tagACCEL { // accl
    BYTE   fVirt;
    WORD   key;
    WORD   cmd;
} ACCEL;
```

You define an accelerator's keystroke by specifying an ASCII character code or a virtual-key code in the **key** member of the **ACCEL** structure. If you specify a virtual-key code, you must first include the **FVIRTKEY** flag in the **fVirt** member; otherwise, the system interprets the code as an ASCII character code. You can include the **FCONTROL**, **FALT**, or **FSHIFT** flag, or all three, to combine the CTRL, ALT, or SHIFT key with the keystroke.

To create the accelerator table, pass a pointer to the array of **ACCEL** structures to the **CreateAcceleratorTable** function. **CreateAcceleratorTable** creates the accelerator table and returns the handle to the table.

Processing Accelerators

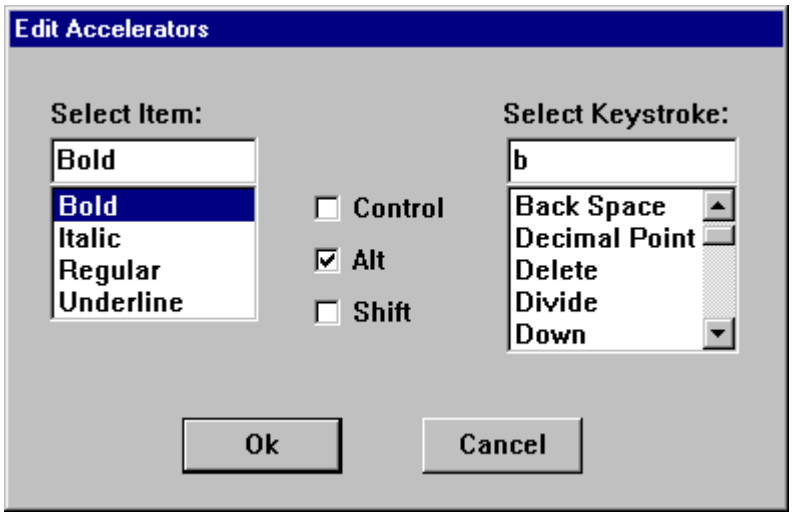
The process of loading and calling accelerators provided by an accelerator table created at run time is the same as processing those provided by an accelerator-table resource. For more information, see Loading the Accelerator Table Resource through Processing WM_COMMAND Messages.

Destroying a Run-Time Accelerator Table

The system automatically destroys accelerator tables created at run time, removing the resources from memory after the application closes. You can destroy an accelerator table and remove it from memory earlier by passing the table's handle to the **DestroyAcceleratorTable** function.

Creating User Editable Accelerators

This example shows how to construct a dialog box that allows the user to change the accelerator associated with a menu item. The dialog box consists of a combo box containing menu items, a combo box containing the names of keys, and check boxes for selecting the CTRL, ALT, and SHIFT keys. The following illustration shows the dialog box.



The following example shows how the dialog box is defined in the resource-definition file.

syntax

```
EdAccelBox DIALOG 5, 17, 193, 114
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "Edit Accelerators"
BEGIN
    COMBOBOX        IDD_MENUITEMS, 10, 22, 52, 53,
                        CBS_SIMPLE | CBS_SORT | WS_VSCROLL |
                        WS_TABSTOP
    CONTROL         "Control", IDD_CNTRL, "Button",
                        BS_AUTOCHECKBOX | WS_TABSTOP,
                        76, 35, 40, 10
    CONTROL         "Alt", IDD_ALT, "Button",
                        BS_AUTOCHECKBOX | WS_TABSTOP,
                        76, 48, 40, 10
    CONTROL         "Shift", IDD_SHIFT, "Button",
                        BS_AUTOCHECKBOX | WS_TABSTOP,
                        76, 61, 40, 10
    COMBOBOX        IDD_KEYSTROKES, 124, 22, 58, 58,
                        CBS_SIMPLE | CBS_SORT | WS_VSCROLL |
                        WS_TABSTOP
    PUSHBUTTON      "Ok", IDOK, 43, 92, 40, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 103, 92, 40, 14
    LTEXT           "Select Item:", 101, 10, 12, 43, 8
    LTEXT           "Select Keystroke:", 102, 123, 12, 60, 8
END
```

The application's menu bar contains a **Character** submenu whose items have accelerators associated with them.

syntax

```
MainMenu MENU
{
```

```
    POPUP "&Character"
    {
        MENUITEM    "&Regular\tF5",          IDM_REGULAR
        MENUITEM    "&Bold\tCtrl+B",         IDM_BOLD
        MENUITEM    "&Italic\tCtrl+I",       IDM_ITALIC
        MENUITEM    "&Underline\tCtrl+U",    IDM_ULINE
    }
}

FontAccel ACCELERATORS
{
    VK_F5,  IDM_REGULAR,    VIRTKEY
    "B",    IDM_BOLD,       CONTROL, VIRTKEY
    "I",    IDM_ITALIC,     CONTROL, VIRTKEY
    "U",    IDM_ULINE,      CONTROL, VIRTKEY
}
```

The menu item values for the menu template are constants defined as follows in the application's header file.

syntax

```
#define IDM_REGULAR    1100
#define IDM_BOLD       1200
#define IDM_ITALIC     1300
#define IDM_ULINE      1400
```

The dialog box uses an array of application-defined VKEY structures, each containing a keystroke-text string and an accelerator-text string. When the dialog box is created, it parses the array and adds each keystroke-text string to the **Select Keystroke** combo box. When the user clicks the **OK** button, the dialog box looks up the selected keystroke-text string and retrieves the corresponding accelerator-text string. The dialog box appends the accelerator-text string to the text of the menu item that the user selected. The following example shows the array of VKEY structures:

C++

```
// VKey Lookup Support

#define MAXKEYS 25

typedef struct _VKEYS {
    char *pKeyName;
    char *pKeyString;
} VKEYS;

VKEYS vkeys[MAXKEYS] = {
    "BkSp",     "Back Space",
```

```
    "PgUp",     "Page Up",
    "PgDn",     "Page Down",
    "End",      "End",
    "Home",     "Home",
    "Lft",      "Left",
    "Up",       "Up",
    "Rgt",      "Right",
    "Dn",       "Down",
    "Ins",      "Insert",
    "Del",      "Delete",
    "Mult",     "Multiply",
    "Add",      "Add",
    "Sub",      "Subtract",
    "DecPt",    "Decimal Point",
    "Div",      "Divide",
    "F2",       "F2",
    "F3",       "F3",
    "F5",       "F5",
    "F6",       "F6",
    "F7",       "F7",
    "F8",       "F8",
    "F9",       "F9",
    "F11",      "F11",
    "F12",      "F12"
};
```

The dialog box's initialization procedure fills the **Select Item** and **Select Keystroke** combo boxes. After the user selects a menu item and associated accelerator, the dialog box examines the controls in the dialog box to get the user's selection, updates the text of the menu item, and then creates a new accelerator table that contains the user-defined new accelerator. The following example shows the dialog-box procedure. Note that you must initialize in your window procedure.

C++

```
// Global variables

HWND hwndMain;      // handle to main window
HACCEL haccel;      // handle to accelerator table

// Dialog-box procedure

BOOL CALLBACK EdAccelProc(HWND hwndDlg, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{
    int nCurSel;            // index of list box item
    UINT idItem;            // menu-item identifier
    UINT uItemPos;          // menu-item position
    UINT i, j = 0;          // loop counters
    static UINT cItems;     // count of items in menu
    char szTemp[32];        // temporary buffer
```

```
    char szAccelText[32];   // buffer for accelerator text
    char szKeyStroke[16];   // buffer for keystroke text
    static char szItem[32]; // buffer for menu-item text
    HWND hwndCtl;           // handle to control window
    static HMENU hmenu;     // handle to "Character" menu
    PCHAR pch, pch2;        // pointers for string copying
    WORD wVKCode;           // accelerator virtual-key code
    BYTE fAccelFlags;       // fVirt flags for ACCEL structure
    LPACCEL lpaccelNew;     // pointer to new accelerator table
    HACCEL haccelOld;       // handle to old accelerator table
    int cAccelerators;      // number of accelerators in table
    static BOOL fItemSelected = FALSE; // item selection flag
    static BOOL fKeySelected = FALSE;  // key selection flag
    HRESULT hr;
    INT numTCHAR;           // TCHARs in listbox text

    switch (uMsg)
    {
        case WM_INITDIALOG:

            // Get the handle to the menu-item combo box.

            hwndCtl = GetDlgItem(hwndDlg, IDD_MENUITEMS);

            // Get the handle to the Character submenu and
            // count the number of items it has. In this example,
            // the menu has position 0. You must alter this value
            // if you add additional menus.
            hmenu = GetSubMenu(GetMenu(hwndMain), 0);
            cItems = GetMenuItemCount(hmenu);

            // Get the text of each item, strip out the '&' and
            // the accelerator text, and add the text to the
            // menu-item combo box.

            for (i = 0; i < cItems; i++)
            {
                if (!(GetMenuString(hmenu, i, szTemp,
                        sizeof(szTemp)/sizeof(TCHAR), MF_BYPOSITION)))
                    continue;
                for (pch = szTemp, pch2 = szItem; *pch != '\0'; )
                {
                    if (*pch != '&')
                    {
                        if (*pch == '\t')
                        {
                            *pch = '\0';
                            *pch2 = '\0';
                        }
                        else *pch2++ = *pch++;
                    }
                    else pch++;
                }
                SendMessage(hwndCtl, CB_ADDSTRING, 0,
                    (LONG) (LPSTR) szItem);
```

```
            }

            // Now fill the keystroke combo box with the list of
            // keystrokes that will be allowed for accelerators.
            // The list of keystrokes is in the application-defined
            // structure called "vkeys".

            hwndCtl = GetDlgItem(hwndDlg, IDD_KEYSTROKES);
            for (i = 0; i < MAXKEYS; i++)
            {
                SendMessage(hwndCtl, CB_ADDSTRING, 0,
                    (LONG) (LPSTR) vkeys[i].pKeyString);
            }

            return TRUE;

        case WM_COMMAND:
            switch (LOWORD(wParam))
            {
                case IDD_MENUITEMS:

                    // The user must select an item from the combo
                    // box. This flag is checked during IDOK
                    // processing to be sure a selection was made.

                    fItemSelected = TRUE;
                    return 0;

                case IDD_KEYSTROKES:

                    // The user must select an item from the combo
                    // box. This flag is checked during IDOK
                    // processing to be sure a selection was made.

                    fKeySelected = TRUE;

                    return 0;

                case IDOK:

                    // If the user has not selected a menu item
                    // and a keystroke, display a reminder in a
                    // message box.

                    if (!fItemSelected || !fKeySelected)
                    {
                        MessageBox(hwndDlg,
                            "Item or key not selected.", NULL,
                            MB_OK);
                        return 0;
                    }

                    // Determine whether the CTRL, ALT, and SHIFT
                    // keys are selected. Concatenate the
                    // appropriate strings to the accelerator-
```

```
                    // text buffer, and set the appropriate
                    // accelerator flags.

                    szAccelText[0] = '\0';
                    hwndCtl = GetDlgItem(hwndDlg, IDD_CNTRL);
                    if (SendMessage(hwndCtl, BM_GETCHECK, 0, 0) == 1)
                    {
                        hr = StringCchCat(szAccelText, 32, "Ctl+");
                        if (FAILED(hr))
                        {
                        // TODO: write error handler
                        }
                        fAccelFlags |= FCONTROL;
                    }
                    hwndCtl = GetDlgItem(hwndDlg, IDD_ALT);
                    if (SendMessage(hwndCtl, BM_GETCHECK, 0, 0) == 1)
                    {
                        hr = StringCchCat(szAccelText, 32, "Alt+");
                        if (FAILED(hr))
                        {
                        // TODO: write error handler
                        }
                        fAccelFlags |= FALT;
                    }
                    hwndCtl = GetDlgItem(hwndDlg, IDD_SHIFT);
                    if (SendMessage(hwndCtl, BM_GETCHECK, 0, 0) == 1)
                    {
                        hr = StringCchCat(szAccelText, 32, "Shft+");
                        if (FAILED(hr))
                        {
                        // TODO: write error handler
                        }
                        fAccelFlags |= FSHIFT;
                    }

                    // Get the selected keystroke, and look up the
                    // accelerator text and the virtual-key code
                    // for the keystroke in the vkeys structure.

                    hwndCtl = GetDlgItem(hwndDlg, IDD_KEYSTROKES);
                    nCurSel = (int) SendMessage(hwndCtl,
                        CB_GETCURSEL, 0, 0);
                    numTCHAR = SendMessage(hwndCtl, CB_GETLBTEXTLEN,
                        nCursel, 0);
                    if (numTCHAR <= 15)
                        {
                        SendMessage(hwndCtl, CB_GETLBTEXT,
                            nCurSel, (LONG) (LPSTR) szKeyStroke);
                        }
                    else
                        {
                        // TODO: writer error handler
                        }

                    for (i = 0; i < MAXKEYS; i++)
```

```
                    {
                    //
                    // lstrcmp requires that both parameters are
                    // null-terminated.
                    //
                        if(lstrcmp(vkeys[i].pKeyString, szKeyStroke)
                            == 0)
                        {
                            hr = StringCchCopy(szKeyStroke, 16,
vkeys[i].pKeyName);
                            if (FAILED(hr))
                            {
                            // TODO: write error handler
                            }
                            break;
                        }
                    }

                    // Concatenate the keystroke text to the
                    // "Ctl+","Alt+", or "Shft+" string.

                        hr = StringCchCat(szAccelText, 32, szKeyStroke);
                        if (FAILED(hr))
                        {
                        // TODO: write error handler
                        }

                    // Determine the position in the menu of the
                    // selected menu item. Menu items in the
                    // "Character" menu have positions 0,2,3, and 4.
                    // Note: the lstrcmp parameters must be
                    // null-terminated.

                    if (lstrcmp(szItem, "Regular") == 0)
                        uItemPos = 0;
                    else if (lstrcmp(szItem, "Bold") == 0)
                        uItemPos = 2;
                    else if (lstrcmp(szItem, "Italic") == 0)
                        uItemPos = 3;
                    else if (lstrcmp(szItem, "Underline") == 0)
                        uItemPos = 4;

                    // Get the string that corresponds to the
                    // selected item.

                    GetMenuString(hmenu, uItemPos, szItem,
                        sizeof(szItem)/sizeof(TCHAR), MF_BYPOSITION);

                    // Append the new accelerator text to the
                    // menu-item text.

                    for (pch = szItem; *pch != '\t'; pch++);
                        ++pch;

                    for (pch2 = szAccelText; *pch2 != '\0'; pch2++)
```

```
            *pch++ = *pch2;
        *pch = '\0';

        // Modify the menu item to reflect the new
        // accelerator text.

        idItem = GetMenuItemID(hmenu, uItemPos);
        ModifyMenu(hmenu, idItem, MF_BYCOMMAND |
            MF_STRING, idItem, szItem);

        // Reset the selection flags.

        fItemSelected = FALSE;
        fKeySelected = FALSE;

        // Save the current accelerator table.

        haccelOld = haccel;

        // Count the number of entries in the current
        // table, allocate a buffer for the table, and
        // then copy the table into the buffer.

        cAccelerators = CopyAcceleratorTable(
            haccelOld, NULL, 0);
        lpaccelNew = (LPACCEL) LocalAlloc(LPTR,
            cAccelerators * sizeof(ACCEL));

        if (lpaccelNew != NULL)
        {
            CopyAcceleratorTable(haccel, lpaccelNew,
                cAccelerators);
        }

        // Find the accelerator that the user modified
        // and change its flags and virtual-key code
        // as appropriate.

        for (i = 0; i < (UINT) cAccelerators; i++)
        {
               if (lpaccelNew[i].cmd == (WORD) idItem)
            {
                lpaccelNew[i].fVirt = fAccelFlags;
                lpaccelNew[i].key = wVKCode;
            }
        }

        // Create the new accelerator table, and
        // destroy the old one.

        DestroyAcceleratorTable(haccelOld);
        haccel = CreateAcceleratorTable(lpaccelNew,
            cAccelerators);

        // Destroy the dialog box.
```

```
                    EndDialog(hwndDlg, TRUE);
                    return 0;

                case IDCANCEL:
                    EndDialog(hwndDlg, TRUE);
                    return TRUE;

                default:
                    break;
            }
        default:
            break;
    }
    return FALSE;
}
```

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Keyboard Accelerator Reference

Article • 04/27/2021

In This Section

- Keyboard Accelerator Functions
- Keyboard Accelerator Messages
- Keyboard Accelerator Notifications
- Keyboard Accelerator Structures

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Keyboard Accelerator Functions

Article • 04/27/2021

In This Section

- CopyAcceleratorTable
- CreateAcceleratorTable
- DestroyAcceleratorTable
- LoadAccelerators
- TranslateAccelerator

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

CopyAcceleratorTableA function (winuser.h)

Article02/22/2024

Copies the specified accelerator table. This function is used to obtain the accelerator-table data that corresponds to an accelerator-table handle, or to determine the size of the accelerator-table data.

Syntax

C++

```
int CopyAcceleratorTableA(
  [in]            HACCEL  hAccelSrc,
  [out, optional] LPACCEL lpAccelDst,
  [in]            int     cAccelEntries
);
```

Parameters

`[in] hAccelSrc`

Type: **HACCEL**

A handle to the accelerator table to copy.

`[out, optional] lpAccelDst`

Type: **LPACCEL**

An array of ACCEL structures that receives the accelerator-table information.

`[in] cAccelEntries`

Type: **int**

The number of ACCEL structures to copy to the buffer pointed to by the *lpAccelDst* parameter.

Return value

Type: **int**

If *lpAccelDst* is **NULL**, the return value specifies the number of accelerator-table entries in the original table. Otherwise, it specifies the number of accelerator-table entries that were copied.

Remarks

> **Note**
>
> The winuser.h header defines CopyAcceleratorTable as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

ACCEL

Conceptual

CreateAcceleratorTable

DestroyAcceleratorTable

Keyboard Accelerators

LoadAccelerators

Reference

TranslateAccelerator

Feedback

CreateAcceleratorTableA function (winuser.h)

Article02/22/2024

Creates an accelerator table.

Syntax

```
C++
```

```
HACCEL CreateAcceleratorTableA(
  [in] LPACCEL paccel,
  [in] int     cAccel
);
```

Parameters

`[in] paccel`

Type: **LPACCEL**

An array of ACCEL structures that describes the accelerator table.

`[in] cAccel`

Type: **int**

The number of ACCEL structures in the array. This must be within the range 1 to 32767 or the function will fail.

Return value

Type: **HACCEL**

If the function succeeds, the return value is the handle to the created accelerator table; otherwise, it is **NULL**. To get extended error information, call GetLastError.

Remarks

Before an application closes, it can use the DestroyAcceleratorTable function to destroy any accelerator tables that it created by using the **CreateAcceleratorTable** function.

Examples

For an example, see Creating User Editable Accelerators.

> **Note**
>
> The winuser.h header defines CreateAcceleratorTable as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

ACCEL

Conceptual

CopyAcceleratorTable

DestroyAcceleratorTable

Keyboard Accelerators

LoadAccelerators

Reference

TranslateAccelerator

Feedback

DestroyAcceleratorTable function (winuser.h)

Article10/13/2021

Destroys an accelerator table.

Syntax

```
C++

BOOL DestroyAcceleratorTable(
  [in] HACCEL hAccel
);
```

Parameters

`[in] hAccel`

Type: **HACCEL**

A handle to the accelerator table to be destroyed. This handle must have been created by a call to the CreateAcceleratorTable or LoadAccelerators function.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero. However, if the table has been loaded more than one call to LoadAccelerators, the function will return a nonzero value only when **DestroyAcceleratorTable** has been called an equal number of times.

If the function fails, the return value is zero.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

CopyAcceleratorTable

CreateAcceleratorTable

Keyboard Accelerators

LoadAccelerators

Reference

TranslateAccelerator

Feedback

LoadAcceleratorsA function (winuser.h)

Article02/09/2023

Loads the specified accelerator table.

Syntax

```
C++

HACCEL LoadAcceleratorsA(
  [in, optional] HINSTANCE hInstance,
  [in]           LPCSTR    lpTableName
);
```

Parameters

`[in, optional] hInstance`

Type: **HINSTANCE**

A handle to the module whose executable file contains the accelerator table to be loaded.

`[in] lpTableName`

Type: **LPCTSTR**

The name of the accelerator table to be loaded. Alternatively, this parameter can specify the resource identifier of an accelerator-table resource in the low-order word and zero in the high-order word. To create this value, use the MAKEINTRESOURCE macro.

Return value

Type: **HACCEL**

If the function succeeds, the return value is a handle to the loaded accelerator table.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

If the accelerator table has not yet been loaded, the function loads it from the specified executable file.

Accelerator tables loaded from resources are freed automatically when the application terminates.

Examples

For an example, see Creating Accelerators for Font Attributes.

> **Note**
>
> The winuser.h header defines LoadAccelerators as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

CopyAcceleratorTable

CreateAcceleratorTable

DestroyAcceleratorTable

Keyboard Accelerators

MAKEINTRESOURCE

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

TranslateAcceleratorA function (winuser.h)

Article02/09/2023

Processes accelerator keys for menu commands. The function translates a WM_KEYDOWN or WM_SYSKEYDOWN message to a WM_COMMAND or WM_SYSCOMMAND message (if there is an entry for the key in the specified accelerator table) and then sends the **WM_COMMAND** or **WM_SYSCOMMAND** message directly to the specified window procedure. **TranslateAccelerator** does not return until the window procedure has processed the message.

Syntax

C++

```
int TranslateAcceleratorA(
  [in] HWND   hWnd,
  [in] HACCEL hAccTable,
  [in] LPMSG  lpMsg
);
```

Parameters

`[in] hWnd`

Type: **HWND**

A handle to the window whose messages are to be translated.

`[in] hAccTable`

Type: **HACCEL**

A handle to the accelerator table. The accelerator table must have been loaded by a call to the LoadAccelerators function or created by a call to the CreateAcceleratorTable function.

`[in] lpMsg`

Type: **LPMSG**

A pointer to an MSG structure that contains message information retrieved from the calling thread's message queue using the GetMessage or PeekMessage function.

Return value

Type: **int**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

To differentiate the message that this function sends from messages sent by menus or controls, the high-order word of the *wParam* parameter of the WM_COMMAND or WM_SYSCOMMAND message contains the value 1.

Accelerator key combinations used to select items from the **window** menu are translated into WM_SYSCOMMAND messages; all other accelerator key combinations are translated into WM_COMMAND messages.

When **TranslateAccelerator** returns a nonzero value and the message is translated, the application should not use the TranslateMessage function to process the message again.

An accelerator need not correspond to a menu command.

If the accelerator command corresponds to a menu item, the application is sent WM_INITMENU and WM_INITMENUPOPUP messages, as if the user were trying to display the menu. However, these messages are not sent if any of the following conditions exist:

- The window is disabled.
- The accelerator key combination does not correspond to an item on the **window** menu and the window is minimized.
- A mouse capture is in effect. For information about mouse capture, see the SetCapture function.

If the specified window is the active window and no window has the keyboard focus (which is generally the case if the window is minimized), **TranslateAccelerator** translates WM_SYSKEYUP and WM_SYSKEYDOWN messages instead of WM_KEYUP and WM_KEYDOWN messages.

If an accelerator keystroke occurs that corresponds to a menu item when the window that owns the menu is minimized, **TranslateAccelerator** does not send a WM_COMMAND message. However, if an accelerator keystroke occurs that does not match any of the items in the window's menu or in the **window** menu, the function sends a **WM_COMMAND** message, even if the window is minimized.

Examples

For an example, see Creating Accelerators for Font Attributes.

> **Note**
>
> The winuser.h header defines TranslateAccelerator as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

CreateAcceleratorTable

GetMessage

Keyboard Accelerators

LoadAccelerators

MSG

PeekMessage

Reference

SetCapture

TranslateMessage

WM_COMMAND

WM_INITMENU

WM_INITMENUPOPUP

WM_KEYDOWN

WM_SYSCOMMAND

WM_SYSKEYDOWN

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Keyboard Accelerator Messages

Article • 04/27/2021

In This Section

- WM_CHANGEUISTATE
- WM_INITMENU
- WM_QUERYUISTATE
- WM_UPDATEUISTATE

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

WM_CHANGEUISTATE message

Article • 12/11/2020

An application sends the **WM_CHANGEUISTATE** message to indicate that the UI state should be changed.

```
C++

#define WM_CHANGEUISTATE                0x0127
```

Parameters

wParam

The low-order word specifies the action to be taken. This member can be one of the following values.

Expand table

Value	Meaning
UIS_CLEAR 2	The UI state flags specified by the high-order word should be cleared.
UIS_INITIALIZE 3	The UI state flags specified by the high-order word should be changed based on the last input event. For more information, see Remarks.
UIS_SET 1	The UI state flags specified by the high-order word should be set.

The high-order word specifies which UI state elements are affected or the style of the control. This member can be one or more of the following values.

Expand table

Value	Meaning
UISF_ACTIVE 0x4	A control should be drawn in the style used for active controls.
UISF_HIDEACCEL 0x2	Keyboard accelerators are hidden.
UISF_HIDEFOCUS	Focus indicators are hidden.

Value	Meaning
0x1	

lParam

This parameter is not used and must be 0.

Remarks

A window should send this message to itself or its parent when it must change the UI state elements of all windows in the same hierarchy. The window procedure must let DefWindowProc process this message so that the entire window tree has a consistent UI state. When the top-level window receives the **WM_CHANGEUISTATE** message, it sends a WM_UPDATEUISTATE message with the same parameters to all child windows. When the system processes the **WM_UPDATEUISTATE** message, it makes the change in the UI state.

If the low-order word of *wParam* is UIS_INITIALIZE, the system will send the WM_UPDATEUISTATE message with a UI state based on the last input event. For example, if the last input came from the mouse, the system will hide the keyboard cues. And, if the last input came from the keyboard, the system will show the keyboard cues. If the state that results from processing **WM_CHANGEUISTATE** is the same as the old state, DefWindowProc does not send this message.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Reference

HIWORD

LOWORD

WM_QUERYUISTATE

Conceptual

Keyboard Accelerators

Feedback

Was this page helpful? Yes No

WM_INITMENU message

Article • 12/11/2020

Sent when a menu is about to become active. It occurs when the user clicks an item on the menu bar or presses a menu key. This allows the application to modify the menu before it is displayed.

A window receives this message through its **WindowProc** function.

```
C++

#define WM_INITMENU                     0x0116
```

Parameters

wParam

A handle to the menu to be initialized.

lParam

This parameter is not used.

Return value

If an application processes this message, it should return zero.

Remarks

A **WM_INITMENU** message is sent only when a menu is first accessed; only one **WM_INITMENU** message is generated for each access. For example, moving the mouse across several menu items while holding down the button does not generate new messages. **WM_INITMENU** does not provide information about menu items.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Reference

WM_INITMENUPOPUP

Conceptual

Keyboard Accelerators

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

WM_QUERYUISTATE message

Article • 12/11/2020

An application sends the **WM_QUERYUISTATE** message to retrieve the UI state for a window.

```
C++

#define WM_QUERYUISTATE                 0x0129
```

Parameters

wParam

This parameter is not used and must be 0.

lParam

This parameter is not used and must be 0.

Return value

The return value is **NULL** if the focus indicators and the keyboard accelerators are visible. Otherwise, the return value can be one or more of the following values.

Return code/value	Description
UISF_ACTIVE 0x4	A control should be drawn in the style used for active controls.
UISF_HIDEACCEL 0x2	Keyboard accelerators are hidden.
UISF_HIDEFOCUS 0x1	Focus indicators are hidden.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Reference

WM_CHANGEUISTATE

WM_UPDATEUISTATE

Conceptual

Keyboard Accelerators

Feedback

WM_UPDATEUISTATE message

Article • 12/11/2020

An application sends the **WM_UPDATEUISTATE** message to change the UI state for the specified window and all its child windows.

```
C++

#define WM_UPDATEUISTATE                0x0128
```

Parameters

wParam

The low-order word specifies the action to be performed. This parameter can be one of the following values.

Value	Meaning
UIS_CLEAR 2	The UI state element specified by the high-order word should be hidden.
UIS_INITIALIZE 3	The UI state element specified by the high-order word should be changed based on the last input event. For more information, see Remarks.
UIS_SET 1	The UI state element specified by the high-order word should be visible.

The high-order word specifies which UI state elements are affected or the style of the control. This parameter can be one or more of the following values.

Value	Meaning
UISF_ACTIVE 0x4	A control should be drawn in the style used for active controls.
UISF_HIDEACCEL 0x2	Keyboard accelerators.
UISF_HIDEFOCUS 0x1	Focus indicators.

lParam

This parameter is not used.

Remarks

A window should send this message to change the UI state of all its child windows. In contrast to the WM_CHANGEUISTATE message, which is a notification, when DefWindowProc processes the **WM_UPDATEUISTATE** message it changes the UI state and propagates the changes to all child windows.

The DefWindowProc function updates the UI state according to the *wParam* value. If the UI state is modified, the function sends the message to all the immediate child windows. **DefWindowProc** also sends this message when it receives a WM_CHANGEUISTATE message notifying the system that a child window intends to modify the UI state.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Reference

DefWindowProc

WM_CHANGEUISTATE

WM_QUERYUISTATE

Conceptual

Keyboard Accelerators

Feedback

Keyboard Accelerator Notifications

Article • 04/27/2021

In This Section

- WM_INITMENUPOPUP
- WM_MENUCHAR
- WM_MENUSELECT
- WM_SYSCHAR
- WM_SYSCOMMAND

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

WM_INITMENUPOPUP message

Article • 04/27/2021

Sent when a drop-down menu or submenu is about to become active. This allows an application to modify the menu before it is displayed, without changing the entire menu.

```
C++

#define WM_INITMENUPOPUP                0x0117
```

Parameters

wParam

A handle to the drop-down menu or submenu.

lParam

The low-order word specifies the zero-based relative position of the menu item that opens the drop-down menu or submenu.

The high-order word indicates whether the drop-down menu is the window menu. If the menu is the window menu, this parameter is **TRUE**; otherwise, it is **FALSE**.

Return value

If an application processes this message, it should return zero.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Reference

HIWORD

LOWORD

WM_INITMENU

Conceptual

Keyboard Accelerators

Feedback

Was this page helpful? Yes No

WM_MENUCHAR message

Article • 12/11/2020

Sent when a menu is active and the user presses a key that does not correspond to any mnemonic or accelerator key. This message is sent to the window that owns the menu.

```
C++

#define WM_MENUCHAR                     0x0120
```

Parameters

wParam

The low-order word specifies the character code that corresponds to the key the user pressed.

The high-order word specifies the active menu type. This parameter can be one of the following values.

Expand table

Value	Meaning
MF_POPUP 0x00000010L	A drop-down menu, submenu, or shortcut menu.
MF_SYSMENU 0x00002000L	The window menu.

lParam

A handle to the active menu.

Return value

An application that processes this message should return one of the following values in the high-order word of the return value.

Expand table

Return code/value	Description
MNC_CLOSE 1	Informs the system that it should close the active menu.
MNC_EXECUTE 2	Informs the system that it should choose the item specified in the low-order word of the return value. The owner window receives a WM_COMMAND message.
MNC_IGNORE 0	Informs the system that it should discard the character the user pressed and create a short beep on the system speaker.
MNC_SELECT 3	Informs the system that it should select the item specified in the low-order word of the return value.

Remarks

The low-order word is ignored if the high-order word contains 0 or 1.

An application should process this message when an accelerator is used to select a menu item that displays a bitmap.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Reference

HIWORD

LOWORD

Conceptual

Feedback

Was this page helpful? Yes No

WM_MENUSELECT message

Article • 12/11/2020

Sent to a menu's owner window when the user selects a menu item.

```
C++

#define WM_MENUSELECT                   0x011F
```

Parameters

wParam

The low-order word specifies the menu item or submenu index. If the selected item is a command item, this parameter contains the identifier of the menu item. If the selected item opens a drop-down menu or submenu, this parameter contains the index of the drop-down menu or submenu in the main menu, and the *lParam* parameter contains the handle to the main (clicked) menu; use the **GetSubMenu** function to get the menu handle to the drop-down menu or submenu.

The high-order word specifies one or more menu flags. This parameter can be one or more of the following values.

Expand table

Value	Meaning
MF_BITMAP 0x00000004L	Item displays a bitmap.
MF_CHECKED 0x00000008L	Item is checked.
MF_DISABLED 0x00000002L	Item is disabled.
MF_GRAYED 0x00000001L	Item is grayed.
MF_HILITE 0x00000080L	Item is highlighted.
MF_MOUSESELECT 0x00008000L	Item is selected with the mouse.

Value	Meaning
MF_OWNERDRAW 0x00000100L	Item is an owner-drawn item.
MF_POPUP 0x00000010L	Item opens a drop-down menu or submenu.
MF_SYSMENU 0x00002000L	Item is contained in the window menu. The *lParam* parameter contains a handle to the menu associated with the message.

lParam

A handle to the menu that was clicked.

Return value

If an application processes this message, it should return zero.

Remarks

If the high-order word of *wParam* contains 0xFFFF and the *lParam* parameter contains **NULL**, the system has closed the menu.

Do not use the value 1 for the high-order word of *wParam*, because this value is specified as (**UINT**) HIWORD(*wParam*). If the value is 0xFFFF, it would be interpreted as 0x0000FFFF, not 1, because of the cast to a **UINT**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Reference

GetSubMenu

HIWORD

LOWORD

Conceptual

Keyboard Accelerators

Feedback

Was this page helpful? Yes No

WM_SYSCHAR message

Article • 08/04/2022

Posted to the window with the keyboard focus when a **WM_SYSKEYDOWN** message is translated by the **TranslateMessage** function. It specifies the character code of a system character key that is, a character key that is pressed while the ALT key is down.

```
C++

#define WM_SYSCHAR                      0x0106
```

Parameters

wParam

The character code of the window menu key.

lParam

The repeat count, scan code, extended-key flag, context code, previous key-state flag, and transition-state flag, as shown in the following table.

Bits	Meaning
0 15	The repeat count for the current message. The value is the number of times the keystroke was auto-repeated as a result of the user holding down the key. If the keystroke is held long enough, multiple messages are sent. However, the repeat count is not cumulative.
16 23	The scan code. The value depends on the original equipment manufacturer (OEM).
24	Indicates whether the key is an extended key, such as the right-hand ALT and CTRL keys that appear on an enhanced 101- or 102-key keyboard. The value is 1 if it is an extended key; otherwise, it is 0.
25 28	Reserved; do not use.
29	The context code. The value is 1 if the ALT key is held down while the key is pressed; otherwise, the value is 0.
30	The previous key state. The value is 1 if the key is down before the message is sent, or it is 0 if the key is up.

Bits	Meaning
31	The transition state. The value is 1 if the key is being released, or it is 0 if the key is being pressed.

For more detail, see Keystroke Message Flags.

Return value

An application should return zero if it processes this message.

Remarks

When the context code is zero, the message can be passed to the **TranslateAccelerator** function, which will handle it as though it were a standard key message instead of a system character-key message. This allows accelerator keys to be used with the active window even if the active window does not have the keyboard focus.

For enhanced 101- and 102-key keyboards, extended keys are the right ALT and CTRL keys on the main section of the keyboard; the INS, DEL, HOME, END, PAGE UP, PAGE DOWN and arrow keys in the clusters to the left of the numeric keypad; the PRINT SCRN key; the BREAK key; the NUMLOCK key; and the divide (/) and ENTER keys in the numeric keypad. Other keyboards may support the extended-key bit in the parameter.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

- **TranslateAccelerator**
- **TranslateMessage**
- **WM_SYSKEYDOWN**
- Keyboard Accelerators
- Keyboard Input (Keyboard and Mouse Input)

- About Keyboard Input

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

WM_SYSCOMMAND message

Article • 08/19/2021

A window receives this message when the user chooses a command from the **Window** menu (formerly known as the system or control menu) or when the user chooses the maximize button, minimize button, restore button, or close button.

```
C++

#define WM_SYSCOMMAND                   0x0112
```

Example

```
C++

 case WM_SYSCOMMAND:
        if ((wParam & 0xFFF0) == SC_CLOSE)
        {
            EndDialog (hDlg, TRUE);
            return(TRUE);
        }
        break;
```

Example from Windows Classic Samples on GitHub.

Parameters

wParam

The type of system command requested. This parameter can be one of the following values.

Expand table

Value	Meaning
SC_CLOSE 0xF060	Closes the window.
SC_CONTEXTHELP 0xF180	Changes the cursor to a question mark with a pointer. If the user then clicks a control in the dialog box, the control receives a WM_HELP message.

Value	Meaning
SC_DEFAULT 0xF160	Selects the default item; the user double-clicked the window menu.
SC_HOTKEY 0xF150	Activates the window associated with the application-specified hot key. The *lParam* parameter identifies the window to activate.
SC_HSCROLL 0xF080	Scrolls horizontally.
SCF_ISSECURE 0x00000001	Indicates whether the screen saver is secure.
SC_KEYMENU 0xF100	Retrieves the window menu as a result of a keystroke. For more information, see the Remarks section.
SC_MAXIMIZE 0xF030	Maximizes the window.
SC_MINIMIZE 0xF020	Minimizes the window.
SC_MONITORPOWER 0xF170	Sets the state of the display. This command supports devices that have power-saving features, such as a battery-powered personal computer. The *lParam* parameter can have the following values: • -1 (the display is powering on) • 1 (the display is going to low power) • 2 (the display is being shut off)
SC_MOUSEMENU 0xF090	Retrieves the window menu as a result of a mouse click.
SC_MOVE 0xF010	Moves the window.
SC_NEXTWINDOW 0xF040	Moves to the next window.
SC_PREVWINDOW 0xF050	Moves to the previous window.
SC_RESTORE 0xF120	Restores the window to its normal position and size.
SC_SCREENSAVE 0xF140	Executes the screen saver application specified in the [boot] section of the System.ini file.
SC_SIZE 0xF000	Sizes the window.

Value	Meaning
SC_TASKLIST 0xF130	Activates the **Start** menu.
SC_VSCROLL 0xF070	Scrolls vertically.

lParam

The low-order word specifies the horizontal position of the cursor, in screen coordinates, if a window menu command is chosen with the mouse. Otherwise, this parameter is not used.

The high-order word specifies the vertical position of the cursor, in screen coordinates, if a window menu command is chosen with the mouse. This parameter is 1 if the command is chosen using a system accelerator, or zero if using a mnemonic.

Return value

An application should return zero if it processes this message.

Remarks

To obtain the position coordinates in screen coordinates, use the following code:

```
xPos = GET_X_LPARAM(lParam);    // horizontal position
yPos = GET_Y_LPARAM(lParam);    // vertical position
```

The DefWindowProc function carries out the window menu request for the predefined actions specified in the previous table.

In **WM_SYSCOMMAND** messages, the four low-order bits of the *wParam* parameter are used internally by the system. To obtain the correct result when testing the value of *wParam*, an application must combine the value 0xFFF0 with the *wParam* value by using the bitwise AND operator.

The menu items in a window menu can be modified by using the GetSystemMenu, AppendMenu, InsertMenu, ModifyMenu, InsertMenuItem, and SetMenuItemInfo functions. Applications that modify the window menu must process **WM_SYSCOMMAND** messages.

An application can carry out any system command at any time by passing a **WM_SYSCOMMAND** message to DefWindowProc. Any **WM_SYSCOMMAND** messages not handled by the application must be passed to **DefWindowProc**. Any command values added by an application must be processed by the application and cannot be passed to **DefWindowProc**.

If password protection is enabled by policy, the screen saver is started regardless of what an application does with the **SC_SCREENSAVE** notification even if fails to pass it to DefWindowProc.

Accelerator keys that are defined to choose items from the window menu are translated into **WM_SYSCOMMAND** messages; all other accelerator keystrokes are translated into WM_COMMAND messages.

If the *wParam* is **SC_KEYMENU**, *lParam* contains the character code of the key that is used with the ALT key to display the popup menu. For example, pressing ALT+F to display the File popup will cause a **WM_SYSCOMMAND** with *wParam* equal to **SC_KEYMENU** and *lParam* equal to 'f'.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Reference

AppendMenu

DefWindowProc

GET_X_LPARAM

GET_Y_LPARAM

GetSystemMenu

InsertMenu

ModifyMenu

WM_COMMAND

Conceptual

Keyboard Accelerators

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Keyboard Accelerator Structures

Article • 04/27/2021

In This Section

- ACCEL

Feedback

Was this page helpful? Yes No

ACCEL structure (winuser.h)

Article04/02/2021

Defines an accelerator key used in an accelerator table.

Syntax

C++

```
typedef struct tagACCEL {
#if ...
  BYTE  fVirt;
#if ...
  WORD  key;
#if ...
  WORD  cmd;
#else
  WORD  fVirt;
#endif
#else
  WORD  key;
#endif
#else
  DWORD cmd;
#endif
} ACCEL, *LPACCEL;
```

Members

`fVirt`

Type: **BYTE**

The accelerator behavior. This member can be one or more of the following values.

Expand table

Value	Meaning
FALT 0x10	The ALT key must be held down when the accelerator key is pressed.
FCONTROL 0x08	The CTRL key must be held down when the accelerator key is pressed.

FNOINVERT 0x02	No top-level menu item is highlighted when the accelerator is used. If this flag is not specified, a top-level menu item will be highlighted, if possible, when the accelerator is used. This attribute is obsolete and retained only for backward compatibility with resource files designed for 16-bit Windows.
FSHIFT 0x04	The SHIFT key must be held down when the accelerator key is pressed.
FVIRTKEY TRUE	The **key** member specifies a virtual-key code. If this flag is not specified, **key** is assumed to specify a character code.

`key`

Type: **WORD**

The accelerator key. This member can be either a virtual-key code or a character code.

`cmd`

Type: **WORD**

The accelerator identifier. This value is placed in the low-order word of the *wParam* parameter of the WM_COMMAND or WM_SYSCOMMAND message when the accelerator is pressed.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

Keyboard Accelerators

Reference

WM_COMMAND

WM_SYSCOMMAND

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Menus (Menus and Other Resources)

Article • 08/19/2021

This section describes menus and explains how to use them.

In This Section

Name	Description
About Menus	Discusses menus.
Using Menus	Provides code examples of tasks related to menus.
Menu Reference	Contains the API reference.

Menu Functions

Name	Description
AppendMenu	Appends a new item to the end of the specified menu bar, drop-down menu, submenu, or shortcut menu. You can use this function to specify the content, appearance, and behavior of the menu item.
CheckMenuItem	Sets the state of the specified menu item's check-mark attribute to either selected or clear.
CheckMenuRadioItem	Checks a specified menu item and makes it a radio item. At the same time, the function clears all other menu items in the associated group and clears the radio-item type flag for those items.
CreateMenu	Creates a menu. The menu is initially empty, but it can be filled with menu items by using the InsertMenuItem, AppendMenu, and InsertMenu functions.
CreatePopupMenu	Creates a drop-down menu, submenu, or shortcut menu. The menu is initially empty. You can insert or append menu items by using the InsertMenuItem function. You can also use the InsertMenu function to insert menu items and the AppendMenu function to append menu items.
DeleteMenu	Deletes an item from the specified menu. If the menu item opens a menu or submenu, this function destroys the handle to the menu or submenu and frees the memory used by the menu or submenu.

Name	Description
DestroyMenu	Destroys the specified menu and frees any memory that the menu occupies.
DrawMenuBar	Redraws the menu bar of the specified window. If the menu bar changes after the system has created the window, this function must be called to draw the changed menu bar.
EnableMenuItem	Enables, disables, or grays the specified menu item.
EndMenu	Ends the calling thread's active menu.
GetMenu	Retrieves a handle to the menu assigned to the specified window.
GetMenuBarInfo	Retrieves information about the specified menu bar.
GetMenuCheckMarkDimensions	Retrieves the dimensions of the default check-mark bitmap. The system displays this bitmap next to selected menu items. Before calling the SetMenuItemBitmaps function to replace the default check-mark bitmap for a menu item, an application must determine the correct bitmap size by calling GetMenuCheckMarkDimensions.
GetMenuDefaultItem	Determines the default menu item on the specified menu.
GetMenuInfo	Retrieves information about a specified menu.
GetMenuItemCount	Retrieves the number of items in the specified menu.
GetMenuItemID	Retrieves the menu item identifier of a menu item located at the specified position in a menu.
GetMenuItemInfo	Retrieves information about a menu item.
GetMenuItemRect	Retrieves the bounding rectangle for the specified menu item.
GetMenuState	Retrieves the menu flags associated with the specified menu item. If the menu item opens a submenu, this function also returns the number of items in the submenu.
GetMenuString	Copies the text string of the specified menu item into the specified buffer.
GetSubMenu	Retrieves a handle to the drop-down menu or submenu activated by the specified menu item.
GetSystemMenu	Enables the application to access the window menu (also known as the system menu or the control menu) for copying and modifying.

Name	Description
HiliteMenuItem	Highlights or removes the highlighting from an item in a menu bar.
InsertMenuItem	Inserts a new menu item at the specified position in a menu.
IsMenu	Determines whether a handle is a menu handle.
LoadMenu	Loads the specified menu resource from the executable (.exe) file associated with an application instance.
LoadMenuIndirect	Loads the specified menu template in memory.
MenuItemFromPoint	Determines which menu item, if any, is at the specified location.
ModifyMenu	Changes an existing menu item. This function is used to specify the content, appearance, and behavior of the menu item.
RemoveMenu	Deletes a menu item or detaches a submenu from the specified menu. If the menu item opens a drop-down menu or submenu, RemoveMenu does not destroy the menu or its handle, allowing the menu to be reused. Before this function is called, the GetSubMenu function should retrieve a handle to the drop-down menu or submenu.
SetMenu	Assigns a new menu to the specified window.
SetMenuDefaultItem	Sets the default menu item for the specified menu.
SetMenuInfo	Sets information for a specified menu.
SetMenuItemBitmaps	Associates the specified bitmap with a menu item. Whether the menu item is selected or clear, the system displays the appropriate bitmap next to the menu item.
SetMenuItemInfo	Changes information about a menu item.
TrackPopupMenu	Displays a shortcut menu at the specified location and tracks the selection of items on the menu. The shortcut menu can appear anywhere on the screen.
TrackPopupMenuEx	Displays a shortcut menu at the specified location and tracks the selection of items on the shortcut menu. The shortcut menu can appear anywhere on the screen.

The following function is obsolete.

Name	Description
InsertMenu	Inserts a new menu item into a menu, moving other items down the menu. **Note:** The **InsertMenu** function has been superseded by the **InsertMenuItem** function. You can still use **InsertMenu**, however, if you do not need any of the extended features of **InsertMenuItem**.

Menu Notifications

Name	Description
WM_COMMAND	Sent when the user selects a command item from a menu, when a control sends a notification message to its parent window, or when an accelerator keystroke is translated.
WM_CONTEXTMENU	Informs a window that the user clicked the right mouse button (*right-clicked*) in the window.
WM_ENTERMENULOOP	Informs an application's main window procedure that a menu modal loop has been entered.
WM_EXITMENULOOP	Informs an application's main window procedure that a menu modal loop has been exited.
WM_GETTITLEBARINFOEX	Sent to request extended title bar information. A window receives this message through its **WindowProc** function.
WM_MENUCOMMAND	Sent when the user makes a selection from a menu.
WM_MENUDRAG	Sent to the owner of a drag-and-drop menu when the user drags a menu item.
WM_MENUGETOBJECT	Sent to the owner of a drag-and-drop menu when the mouse cursor enters a menu item or moves from the center of the item to the top or bottom of the item.
WM_MENURBUTTONUP	Sent when the user releases the right mouse button while the cursor is on a menu item.
WM_NEXTMENU	Sent to an application when the right or left arrow key is used to switch between the menu bar and the system menu.
WM_UNINITMENUPOPUP	Sent when a drop-down menu or submenu has been destroyed.

Menu Structures

Name	Description
MDINEXTMENU	Contains information about the menu to be activated.
MENUBARINFO	Contains menu bar information.
MENUEX_TEMPLATE_HEADER	Defines the header for an extended menu template. This structure definition is for explanation only; it is not present in any standard header file.
MENUEX_TEMPLATE_ITEM	Defines a menu item in an extended menu template. This structure definition is for explanation only; it is not present in any standard header file.
MENUGETOBJECTINFO	Contains information about the menu that the mouse cursor is on.
MENUINFO	Contains information about a menu.
MENUITEMINFO	Contains information about a menu item.
MENUITEMTEMPLATE	Defines a menu item in a menu template.
MENUITEMTEMPLATEHEADER	Defines the header for a menu template. A complete menu template consists of a header and one or more menu item lists.
TPMPARAMS	Contains extended parameters for the TrackPopupMenuEx function.

About Menus

Article • 08/19/2020

A *menu* is a list of items that specify options or groups of options (a submenu) for an application. Clicking a menu item opens a submenu or causes the application to carry out a command. This section provides information on the following topics:

Menu Bars and Menus

A menu is arranged in a hierarchy. At the top level of the hierarchy is the *menu bar*; which contains a list of *menus*, which in turn can contain *submenus*. A menu bar is

sometimes called a *top-level menu*, and the menus and submenus are also known as *pop-up menus*.

A menu item can either carry out a command or open a submenu. An item that carries out a command is called a *command item* or a *command*.

An item on the menu bar almost always opens a menu. Menu bars rarely contain command items. A menu opened from the menu bar drops down from the menu bar and is sometimes called a *drop-down menu*. When a drop-down menu is displayed, it is attached to the menu bar. A menu item on the menu bar that opens a drop-down menu is also called a *menu name*.

The menu names on a menu bar represent the main categories of commands that an application provides. Selecting a menu name from the menu bar typically opens a menu whose menu items correspond to the commands in a category. For example, a menu bar might contain a **File** menu name that, when clicked by the user, activates a menu with menu items such as **New**, **Open**, and **Save**. To get information about a menu bar, call **GetMenuBarInfo**.

Only an overlapped or pop-up window can contain a menu bar; a child window cannot contain one. If the window has a title bar, the system positions the menu bar just below it. A menu bar is always visible. A submenu is not visible, however, until the user selects a menu item that activates it. For more information about overlapped and pop-up windows, see Window Types.

Each menu must have an owner window. The system sends messages to a menu's owner window when the user selects the menu or chooses an item from the menu.

This section discusses the following topics.

- Shortcut Menus
- The Window Menu
- Help Identifier

Shortcut Menus

The system also provides *shortcut menus*. A shortcut menu is not attached to the menu bar; it can appear anywhere on the screen. An application typically associates a shortcut menu with a portion of a window, such as the client area, or with a specific object, such as an icon. For this reason, these menus are also called *context menus*.

A shortcut menu remains hidden until the user activates it, typically by right-clicking a selection, a toolbar, or a taskbar button. The menu is usually displayed at the position of

the caret or mouse cursor.

The Window Menu

The **Window** menu (also known as the **System** menu or **Control** menu) is a pop-up menu defined and managed almost exclusively by the operating system. The user can open the window menu by clicking the application icon on the title bar or by right-clicking anywhere on the title bar.

The **Window** menu provides a standard set of menu items that the user can choose to change a window's size or position, or close the application. Items on the window menu can be added, deleted, and modified, but most applications just use the standard set of menu items. An overlapped, pop-up, or child window can have a window menu. It is uncommon for an overlapped or pop-up window not to include a window menu.

When the user chooses a command from the **Window** menu, the system sends a **WM_SYSCOMMAND** message to the menu's owner window. In most applications, the window procedure does not process messages from the window menu. Instead, it simply passes the messages to the **DefWindowProc** function for system-default processing of the message. If an application adds a command to the window menu, the window procedure must process the command.

An application can use the **GetSystemMenu** function to create a copy of the default window menu to modify. Any window that does not use the **GetSystemMenu** function to make its own copy of the window menu receives the standard window menu.

Help Identifier

Associated with each menu bar, menu, submenu, and shortcut menu is a help identifier. If the user presses the F1 key while the menu is active, this value is sent to the owner window as part of a **WM_HELP** message.

Keyboard Access to Menus

The system provides a standard keyboard interface for menus. You can enhance this interface by providing mnemonic access keys and shortcut (accelerator) keys for your menu items.

The following topics describe the standard keyboard interface, access keys, and shortcut keys:

- Standard Keyboard Interface

- Menu Access Keys
- Menu Shortcut Keys

Standard Keyboard Interface

The system is designed to work with or without a mouse or other pointing device. Because the system provides a standard keyboard interface, the user can use the keyboard to select menu items. This keyboard interface does not need special code. An application receives a command message whether the user selects a menu item through the keyboard or by using a mouse. The standard keyboard interface processes the following keystrokes.

Keystroke	Action
Alphabetical character	Selects the first menu item with the specified character as its access key. If the selected item invokes a menu, the menu is displayed and the first item is highlighted. Otherwise, the menu item is chosen.
ALT	Toggles in and out of menu bar mode.
ALT+SPACEBAR	Displays the window menu.
ENTER	Activates a menu and selects the first menu item if an item has a menu associated with it. Otherwise, this keystroke chooses the item as if the user released the mouse button while the item was selected.
ESC	Exits menu mode.
LEFT ARROW	Cycles to the previous top-level menu item. Top-level menu items include menu names and the window menu. If the selected item is in a menu, the previous column in the menu is selected or the previous top-level menu item is selected.
RIGHT ARROW	Works like the LEFT ARROW key, except in the opposite direction. In menus, this keystroke moves forward one column; when the currently selected item is in the far-right column, the next menu is selected.
UP or DOWN ARROWS	Activates a menu when pressed in a menu name. When pressed in a menu, the UP ARROW keystroke selects the previous item; the DOWN ARROW keystroke selects the next item.

Menu Access Keys

The standard keyboard interface for menus can be enhanced by adding access keys (mnemonics) to menu items. An access key is an underlined letter in the text of a menu

item. When a menu is active, the user can select a menu item by pressing the key that corresponds to the item's underlined letter. The user makes the menu bar active by pressing the ALT key to highlight the first item on the menu bar. A menu is active when it is displayed.

To create an access key for a menu item, precede any character in the item's text string with an ampersand. For example, the text string "&Move" causes the system to underline the letter "M".

Menu Shortcut Keys

In addition to having an access key, a menu item can have a shortcut key associated with it. A shortcut key is different from an access key, because the menu does not have to be active for the shortcut key to work. Also, an access key is *always* associated with a menu item, while an shortcut key is *usually* (but does not have to be) associated with a menu item.

Text that identifies the shortcut key is added to the menu-item text string. The shortcut text appears to the right of the menu item name, after a backslash and tab character (\t). For example, "&Close\tAlt+F4" represents a Close command with the ALT+F4 key combination as its shortcut key and with the letter "C" as its access key. For more information, see Keyboard Accelerators.

Menu Creation

You can create a menu using either a menu template or menu creation functions. Menu templates are typically defined as resources. Menu-template resources can be loaded explicitly or assigned as the default menu for a window class. You can also create menu-template resources dynamically in memory.

The following topics describe menu creation in detail:

- Menu Template Resources
- Menu Template in Memory
- Menu Handles
- Menu Creation Functions
- Menu Display
- Window Class Menus

Menu Template Resources

Most applications create menus using menu-template resources. A *menu template* defines a menu, including the items in the menu bar and all menus. For information about creating a menu-template resource, see the documentation included with your development tools.

After you create a menu-template resource and add it to your application's executable (.exe) file, you can use the **LoadMenu** function to load the resource into memory. This function returns a handle to the menu, which you can then assign to a window by using the **SetMenu** function. You can assign a menu to any window that is not a child window.

Implementing menus as resources makes an application easier to localize for use in multiple countries/regions. Only the resource-definition file needs to be localized for each language, not the application's source code.

Menu Template in Memory

A menu can be created from a menu template that is built in memory at run time. For example, an application that allows a user to customize its menu might create a menu template in memory based on the user's preferences. The application could then save the template in a file or in the registry for future use. To create a menu from a template in memory, use the **LoadMenuIndirect** function. For descriptions of menu-template formats, see Menu Template Resources.

A standard menu template consists of a **MENUITEMTEMPLATEHEADER** structure followed by one or more **MENUITEMTEMPLATE** structures.

An extended menu template consists of a **MENUEX_TEMPLATE_HEADER** structure followed by one or more **MENUEX_TEMPLATE_ITEM** structures.

Menu Handles

The system generates a unique handle to each menu. A *menu handle* is a value of the **HMENU** type. An application must specify a menu handle in many of the menu functions. You receive a handle to a menu bar when you create the menu or load a menu resource.

To retrieve a handle to the menu bar for a menu that has been created or loaded, use the **GetMenu** function. To retrieve a handle to the submenu associated with a menu item, use the **GetSubMenu** or **GetMenuItemInfo** function. To retrieve a handle to a window menu, use the **GetSystemMenu** function.

Menu Creation Functions

Using menu creation functions, you can create menus at run time or add menu items to existing menus. You can use the **CreateMenu** function to create an empty menu bar and the **CreatePopupMenu** function to create an empty menu. You can save certain settings information for a menu by using the **MENUINFO** structure. To get or retrieve the settings of a menu, use **GetMenuInfo** or **SetMenuInfo**. To add items to a menu, use the **InsertMenuItem** function. The older **AppendMenu** and **InsertMenu** functions are still supported, but **InsertMenuItem** should be used for new applications.

Menu Display

After a menu has been loaded or created, it must be assigned to a window before the system can display it. You can assign a menu by defining a class menu. For more information, see Window Class Menus. You can also assign a menu to a window by specifying a handle to the menu as the *hMenu* parameter of the **CreateWindow** or **CreateWindowEx** function, or by calling the **SetMenu** function.

To display a shortcut menu use the **TrackPopupMenuEx** function. Shortcut menus, also called floating pop-up menus or context menus, are typically displayed when the **WM_CONTEXTMENU** message is processed.

You can assign a menu to any window that is not a child window.

The older **TrackPopupMenu** function is still supported, but new applications should use the **TrackPopupMenuEx** function.

Window Class Menus

You can specify a default menu, called a *class menu,* when you register a window class. To do so, you assign the name of the menu-template resource to the **lpszMenuName** member of the **WNDCLASS** structure used to register the class.

By default, every window is assigned the class menu for its window class so you do not need to explicitly load the menu and assign it to each window. You can override the class menu by specifying a different menu handle in a call to the **CreateWindowEx** function. You can also change a window's menu after it is created by using the **SetMenu** function. For more information, see Window Classes.

Menu Items

The following topics discuss what the system does when the user chooses a menu item, and the ways an application can control an item's appearance and functionality:

Command Items and Items that Open Submenus

When the user chooses a command item, the system sends a command message to the window that owns the menu. If the command item is on the window menu, the system sends the **WM_SYSCOMMAND** message. Otherwise, it sends the **WM_COMMAND** message.

Associated with each menu item that opens a submenu is a handle to the corresponding submenu. When the user points to such an item, the system opens the submenu. No command message is sent to the owner window. However, the system sends a **WM_INITMENUPOPUP** message to the owner window before displaying the submenu. You can get a handle to the submenu associated with an item by using the **GetSubMenu** or **GetMenuItemInfo** function.

A menu bar typically contains menu names, but it can also contain command items. A submenu typically contains command items, but it can also contain items that open nested submenus. By adding such items to submenus, you can nest menus to any depth. To provide a visual cue for the user, the system automatically displays a small arrow to the right of the text of a menu item that opens a submenu.

Menu-Item Identifier

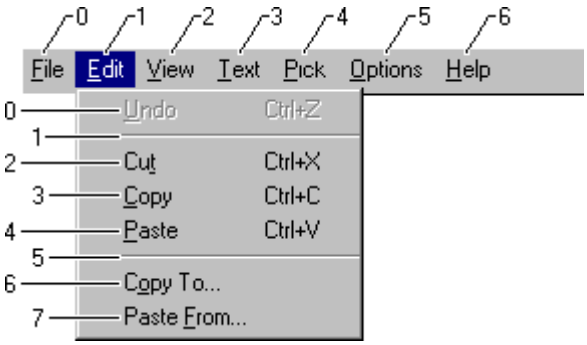
Associated with each menu item is a unique, application-defined integer, called a *menu-item identifier*. When the user chooses a command item from a menu, the system sends the item's identifier to the owner window as part of a **WM_COMMAND** message. The window procedure examines the identifier to determine the source of the message, and processes the message accordingly. In addition, you can specify a menu item using its identifier when you call menu functions; for example, to enable or disable a menu item.

Menu items that open submenus have identifiers just as command items do. However, the system does not send a command message when such an item is selected from a menu. Instead, the system opens the submenu associated with the menu item.

To retrieve the identifier of the menu item at a specified position, use the **GetMenuItemID** or **GetMenuItemInfo** function.

Menu-Item Position

In addition to having a unique identifier, each menu item in a menu bar or menu has a unique position value. The leftmost item in a menu bar, or the top item in a menu, has position zero. The position value is incremented for subsequent menu items. The system assigns a position value to all items in a menu, including separators. The following illustration shows the position values of items in a menu bar and in a menu.



When calling a menu function that modifies or retrieves information about a specific menu item, you can specify the item using either its identifier or its position. For more information, see the next section.

Accessing Menu Items Programmatically

Most menu functions allow you to specify a menu item either by position or by command. Some functions use the **MF_BYPOSITION** and **MF_BYCOMMAND** flags to indicate the search algorithm; others have an explicit *fByPosition* parameter. If you specify the menu item by position, the item number is a zero-based index into the menu. If you specify the menu item by command, the menu and its submenus are searched for an item whose menu identifier equals the item number provided. If more than one item in the menu hierarchy matches the item number, it is unspecified which one is used. If your menus contain duplicate menu identifiers, you should use position-based menu operations to avoid this ambiguity.

Default Menu Items

A submenu can contain one default menu item. When the user opens a submenu by double-clicking, the system sends a command message to the menu's owner window and closes the menu as if the default command item had been chosen. If there is no default command item, the submenu remains open. To retrieve and set the default item for a submenu, use the GetMenuDefaultItem and SetMenuDefaultItem functions.

Selected and Clear Menu Items

A menu item can be either selected or clear. The system displays a bitmap next to selected menu items to indicate their selected state. The system does not display a bitmap next to clear items, unless an application-defined "clear" bitmap is specified. Only menu items in a menu can be selected; items in a menu bar cannot be selected.

Applications typically check or clear a menu item to indicate whether an option is in effect. For example, suppose an application has a toolbar that the user can show or hide by using a **Toolbar** command on a menu. When the toolbar is hidden, the **Toolbar** menu item is clear. When the user chooses the command, the application checks the menu item and shows the toolbar.

A check mark attribute controls whether a menu item is selected. You can set a menu item's check mark attribute by using the CheckMenuItem function. You can use the GetMenuState function to determine whether a menu item is currently selected or cleared.

Instead of CheckMenuItem and GetMenuState, you can use the GetMenuItemInfo and SetMenuItemInfo functions to retrieve and set the check state of a menu item.

Sometimes, a group of menu items corresponds to a set of mutually exclusive options. In this case, you can indicate the selected option by using a selected radio menu item (analogous to a radio button control). Selected radio items are displayed with a bullet bitmap instead of a check mark bitmap. To check a menu item and make it a radio item, use the CheckMenuRadioItem function.

By default, the system displays a check mark or bullet bitmap next to selected menu items and no bitmap next to cleared menu items. However, you can use the SetMenuItemBitmaps function to associate application-defined selected and cleared bitmaps with a menu item. The system then uses the specified bitmaps to indicate the menu item's selected or cleared state.

Application-defined bitmaps associated with a menu item must be the same size as the default check mark bitmap, the dimensions of which may vary depending on screen resolution. To retrieve the correct dimensions, use the GetSystemMetrics function. You can create multiple bitmap resources for different screen resolutions; create one bitmap

resource and scale it, if necessary; or create a bitmap at run time and draw an image in it. The bitmaps may be either monochrome or color. However, because menu items are inverted when highlighted, the appearance of certain inverted color bitmaps may be undesirable. For more information, see Bitmaps.

Enabled, Grayed, and Disabled Menu Items

A menu item can be enabled, grayed, or disabled. By default, a menu item is enabled. When the user chooses an enabled menu item, the system sends a command message to the owner window or displays the corresponding submenu, depending on what kind of menu item it is.

When menu items are not available to the user, they should be grayed or disabled. Grayed and disabled menu items cannot be chosen. A disabled item looks just like an enabled item. When the user clicks on a disabled item, the item is not selected, and nothing happens. Disabled items can be useful in, for example, a tutorial that presents a menu that looks active but isn't.

An application grays an unavailable menu item to provide a visual cue to the user that a command is not available. You can use a grayed item when an action is not appropriate (for example, you can gray the **Print** command in the **File** menu when the system does not have a printer installed).

The **EnableMenuItem** function enables, grays, or disables a menu item. To determine whether a menu item is enabled, grayed, or disabled, use the **GetMenuItemInfo** function.

Instead of **GetMenuItemInfo**, you can also use the **GetMenuState** function to determine whether a menu item is enabled, grayed, or disabled.

Highlighted Menu Items

The system automatically highlights menu items on menus as the user selects them. However, highlighting can be explicitly added or removed from a menu name on the menu bar by using the **HiliteMenuItem** function. This function has no effect on menu items on menus. When **HiliteMenuItem** is used to highlight a menu name, though, the name only appears to be selected. If the user presses the ENTER key, the highlighted item is not chosen. This feature might be useful in, for example, a training application that demonstrates the use of menus.

Owner-Drawn Menu Items

An application can completely control the appearance of a menu item by using an *owner-drawn item*. Owner-drawn items require an application to take total responsibility for drawing selected (highlighted), selected, and cleared states. For example, if an application provided a font menu, it could draw each menu item by using the corresponding font; the item for Roman would be drawn with roman, the item for Italic would be drawn in italic, and so on. For more information, see Creating Owner-Drawn Menu Items.

Menu Item Separators and Line Breaks

The system provides a special type of menu item, called a *separator*, that appears as a horizontal line. You can use a separator to divide a menu into groups of related items. A separator cannot be used in a menu bar, and the user cannot select a separator.

When a menu bar contains more menu names than will fit on one line, the system wraps the menu bar by automatically breaking it into two or more lines. You can cause a line break to occur at a specific item on a menu bar by assigning the **MFT_MENUBREAK** type flag to the item. The system places that item and all subsequent items on a new line.

When a menu contains more items than will fit in one column, the menu will be truncated. You can cause a column break to occur at a specific item in a menu by assigning the **MFT_MENUBREAK** type flag to the item or using the MENUBREAK option in the MENUITEM statement. The system places that item and all subsequent items in a new column. The **MFT_MENUBARBREAK** type flag has the same effect, except that a vertical line appears between the new column and the old.

If you use the **AppendMenu**, **InsertMenu**, or **ModifyMenu** functions to assign line breaks, you should assign the type flags **MF_MENUBREAK** or **MF_MENUBARBREAK**.

Messages Used with Menus

The system reports menu-related activity by sending messages to the window procedure of the window that owns the menu. The system sends a series of messages when the user selects items on the menu bar or clicks the right mouse button to display a shortcut menu.

When the user activates an item on the menu bar, the owner window first receives a **WM_SYSCOMMAND** message. This message includes a flag that indicates whether the user activated the menu by using the keyboard (SC_KEYMENU) or the mouse (SC_MOUSEMENU). For more information, see Keyboard Access to Menus.

Next, before displaying any menus, the system sends the WM_INITMENU message to the window procedure so that an application can modify the menus before the user sees them. The system sends the **WM_INITMENU** message only once per menu activation.

When the user points to a menu item that opens a submenu, the system sends the owner window the WM_INITMENUPOPUP message before displaying the submenu. This message gives the application an opportunity to modify the submenu before it is displayed.

Each time the user moves the highlighting from one item to another, the system sends a WM_MENUSELECT message to the window procedure of the menu's owner window. This message identifies the currently selected menu item. Many applications provide an information area at the bottom of their main windows and use this message to display additional information about the selected menu item.

When the user chooses a command item from a menu, the system sends a WM_COMMAND message to the window procedure. The low-order word of the **WM_COMMAND** message's *wParam* parameter contains the identifier of the chosen item. The window procedure should examine the identifier and process the message accordingly.

You can save information for a menu using the MENUINFO structure. If the menu is defined with a **MENUINFO.dwStyle** value of MNS_NOTIFYBYPOS, the system sends WM_MENUCOMMAND instead of the WM_COMMAND when an item is selected. This allows you to access the information in the **MENUINFO** structure and also provides the index of the selected item directly.

Not all menus are accessible through a window's menu bar. Many applications display shortcut menus when the user clicks the right mouse button at a specific location. Such applications should process the WM_CONTEXTMENU message and display a shortcut menu, if appropriate. If an application does not display a shortcut menu, it should pass the **WM_CONTEXTMENU** message to the DefWindowProc function for default processing.

The WM_MENURBUTTONUP message is sent when the user releases the right mouse button while the cursor is on a menu item. This message is provided so that applications can display a context-sensitive or shortcut menu for a menu item.

There are a few messages that only involve drag-and-drop menus. The WM_MENUGETOBJECT is sent to the owner of a drag-and-drop menu when the mouse cursor enters a menu item or moves from the center of an item to the top or the bottom of an item. The WM_MENUDRAG message is sent when the user actually drags a menu item.

When a drop-down menu or a submenu has been destroyed, the system sends a **WM_UNINITMENUPOPUP** message.

Menu Destruction

If a menu is assigned to a window and that window is destroyed, the system automatically destroys the menu and its submenus, freeing the menu's handle and the memory occupied by the menu. The system does not automatically destroy a menu that is not assigned to a window. An application must destroy the unassigned menu by calling the **DestroyMenu** function. Otherwise, the menu continues to exist in memory even after the application closes. To end the calling thread's active menu, use **EndMenu**. If a platform does not support **EndMenu**, send the owner of the active menu a **WM_CANCELMODE** message.

Using Menus

Article • 09/08/2021

This section describes the following tasks:

Using a Menu-Template Resource

You typically include a menu in an application by creating a menu-template resource and then loading the menu at run time. This section describes the format of a menu template, and explains how to load a menu-template resource and use it in your application. For information about creating a menu-template resource, see the documentation included with your development tools.

Extended Menu-Template Format

The extended menu-template format supports additional menu functionality. Like standard menu-template resources, extended menu-template resources have the **RT_MENU** resource type. The system distinguishes the two resource formats by the version number, which is the first member of the resource header.

An extended menu template consists of a **MENUEX_TEMPLATE_HEADER** structure followed by one more **MENUEX_TEMPLATE_ITEM** item definition structures.

Old Menu-Template Format

An old menu template (Microsoft Windows NT 3.51 and earlier) defines a menu, but does not support the new menu functionality. An old menu-template resource has the **RT_MENU** resource type.

An old menu template consists of a **MENUITEMTEMPLATEHEADER** structure followed by one or more **MENUITEMTEMPLATE** structures.

Loading a Menu-Template Resource

To load a menu-template resource, use the **LoadMenu** function, specifying a handle to the module that contains the resource and the menu template's identifier. The **LoadMenu** function returns a menu handle that you can use to assign the menu to a window. This window becomes the menu's owner window, receiving all the messages generated by the menu.

To create a menu from a menu template that is already in memory, use the **LoadMenuIndirect** function. This is useful if your application generates menu templates dynamically.

To assign a menu to a window, use the **SetMenu** function or specify the menu's handle in the *hMenu* parameter of the **CreateWindowEx** function when creating a window. Another way you can assign a menu to a window is to specify a menu template when you register a window class; the template identifies the specified menu as the class menu for that window class.

To have the system automatically assign a specific menu to a window, specify the menu's template when you register the window's class. The template identifies the specified menu as the class menu for that window class. Then, when you create a window of the specified class, the system automatically assigns the specified menu to the window.

You cannot assign a menu to a window that is a child window.

To create a class menu, include the identifier of the menu-template resource as the **lpszMenuName** member of a WNDCLASS structure and then pass a pointer to the structure to the RegisterClass function.

Creating a Class Menu

The following example shows how to create a class menu for an application, create a window that uses the class menu, and process menu commands in the window procedure.

Following is the relevant portion of the application's header file:

C++

```
// Menu-template resource identifier

#define IDM_MYMENURESOURCE   3
```

Following are the relevant portions of the application itself:

C++

```
HINSTANCE hinst;

int APIENTRY WinMain(HINSTANCE hinstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow)
{
    MSG msg = { };  // message
    WNDCLASS wc;    // windowclass data
    HWND hwnd;      // handle to the main window

    // Create the window class for the main window. Specify
    // the identifier of the menu-template resource as the
    // lpszMenuName member of the WNDCLASS structure to create
    // the class menu.

    wc.style = 0;
    wc.lpfnWndProc = (WNDPROC) MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
```

```
    wc.hInstance = hinstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = GetStockObject(WHITE_BRUSH);
    wc.lpszMenuName =  MAKEINTRESOURCE(IDM_MYMENURESOURCE);
    wc.lpszClassName = "MainWClass";

    if (!RegisterClass(&wc))
        return FALSE;

    hinst = hinstance;

    // Create the main window. Set the hmenu parameter to NULL so
    // that the system uses the class menu for the window.

    hwnd = CreateWindow("MainWClass", "Sample Application",
        WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, hinstance,
        NULL);

    if (hwnd == NULL)
        return FALSE;

    // Make the window visible and send a WM_PAINT message to the
    // window procedure.

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    // Start the main message loop.

    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return msg.wParam;
        UNREFERENCED_PARAMETER(hPrevInstance);
}


LRESULT APIENTRY MainWndProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{

    switch (uMsg)
    {
        // Process other window messages.

        case WM_COMMAND:

            // Test for the identifier of a command item.

            switch(LOWORD(wParam))
            {
```

```
                case IDM_FI_OPEN:
                    DoFileOpen();   // application-defined
                    break;

                case IDM_FI_CLOSE:
                    DoFileClose();  // application-defined
                    break;
                // Process other menu commands.

                default:
                    break;

            }
            return 0;

        // Process other window messages.

        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return NULL;
}
```

Creating a Shortcut Menu

To use a shortcut menu in an application, pass its handle to the TrackPopupMenuEx function. An application typically calls **TrackPopupMenuEx** in a window procedure in response to a user-generated message, such as WM_LBUTTONDOWN or WM_KEYDOWN.

In addition to the pop-up menu handle, TrackPopupMenuEx requires that you specify a handle to the owner window, the position of the shortcut menu (in screen coordinates), and the mouse button that the user can use to choose an item.

The older TrackPopupMenu function is still supported, but new applications should use the TrackPopupMenuEx function. The **TrackPopupMenuEx** function requires the same parameters as **TrackPopupMenu**, but also lets you specify a portion of the screen that the menu should not obscure. An application typically calls these functions in a window procedure when processing the WM_CONTEXTMENU message.

You can specify the position of a shortcut menu by providing x- and y-coordinates along with the **TPM_CENTERALIGN**, **TPM_LEFTALIGN**, or **TPM_RIGHTALIGN** flag. The flag specifies the position of the shortcut menu relative to the x- and y-coordinates.

You should permit the user to choose an item from a shortcut menu by using the same mouse button used to display the menu. To do this, specify either **TPM_LEFTBUTTON** or

TPM_RIGHTBUTTON flag to indicate which mouse button the user can use to choose a menu item.

- Processing the WM_CONTEXTMENU Message
- Creating a Shortcut Font-Attributes Menu
- Displaying a Shortcut Menu

Processing the WM_CONTEXTMENU Message

The **WM_CONTEXTMENU** message is generated when an application's window procedure passes the **WM_RBUTTONUP** or **WM_NCRBUTTONUP** message to the **DefWindowProc** function. The application can process this message to display a shortcut menu appropriate to a specific portion of its screen. If the application does not display a shortcut menu, it should pass the message to **DefWindowProc** for default handling.

Following is an example of **WM_CONTEXTMENU** message processing as it might appear in an application's window procedure. The low-order and high-order words of the *lParam* parameter specify the screen coordinates of the mouse when the right mouse button is released (note that these coordinates can take negative values on systems with multiple monitors). The application-defined **OnContextMenu** function returns **TRUE** if it displays a context menu, or **FALSE** if it does not.

C++

```
case WM_CONTEXTMENU:
    if (!OnContextMenu(hwnd, GET_X_LPARAM(lParam),
              GET_Y_LPARAM(lParam)))
        return DefWindowProc(hwnd, uMsg, wParam, lParam);
    break;
```

The following application-defined OnContextMenu function displays a shortcut menu if the specified mouse position is within the window's client area. A more sophisticated function might display one of several different menus, depending on which portion of the client area is specified. To actually display the shortcut menu, this example calls an application-defined function called DisplayContextMenu. For a description of this function, see Displaying a Shortcut Menu.

C++

```
BOOL WINAPI OnContextMenu(HWND hwnd, int x, int y)
{
    RECT rc;                    // client area of window
    POINT pt = { x, y };        // location of mouse click
```

```
    // Get the bounding rectangle of the client area.

    GetClientRect(hwnd, &rc);

    // Convert the mouse position to client coordinates.

    ScreenToClient(hwnd, &pt);

    // If the position is in the client area, display a
    // shortcut menu.

    if (PtInRect(&rc, pt))
    {
        ClientToScreen(hwnd, &pt);
        DisplayContextMenu(hwnd, pt);
        return TRUE;
    }

    // Return FALSE if no menu is displayed.

    return FALSE;
}
```

Creating a Shortcut Font-Attributes Menu

The example in this section contains portions of code from an application that creates and displays a shortcut menu that enables the user to set fonts and font attributes. The application displays the menu in the client area of its main window whenever the user clicks the left mouse button.

Here is the menu template for the shortcut menu that is provided in the application's resource-definition file.

C++

```
PopupMenu MENU
BEGIN
  POPUP "Dummy Popup"
    BEGIN
      POPUP "Fonts"
        BEGIN
          MENUITEM "Courier",     IDM_FONT_COURIER
          MENUITEM "Times Roman", IDM_FONT_TMSRMN
          MENUITEM "Swiss",       IDM_FONT_SWISS
          MENUITEM "Helvetica",   IDM_FONT_HELV
          MENUITEM "Old English", IDM_FONT_OLDENG
        END
      POPUP "Sizes"
        BEGIN
```

```
            MENUITEM "7",  IDM_SIZE_7
            MENUITEM "8",  IDM_SIZE_8
            MENUITEM "9",  IDM_SIZE_9
            MENUITEM "10", IDM_SIZE_10
            MENUITEM "11", IDM_SIZE_11
            MENUITEM "12", IDM_SIZE_12
            MENUITEM "14", IDM_SIZE_14
          END
        POPUP "Styles"
          BEGIN
            MENUITEM "Bold",         IDM_STYLE_BOLD
            MENUITEM "Italic",       IDM_STYLE_ITALIC
            MENUITEM "Strike Out",   IDM_STYLE_SO
            MENUITEM "Superscript", IDM_STYLE_SUPER
            MENUITEM "Subscript",    IDM_STYLE_SUB
          END
    END

END
```

The following example gives the window procedure and supporting functions used to create and display the shortcut menu.

C++

```
LRESULT APIENTRY MenuWndProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{
    RECT rc;    // client area
    POINT pt;   // location of mouse click

    switch (uMsg)
    {
        case WM_LBUTTONDOWN:

            // Get the bounding rectangle of the client area.

            GetClientRect(hwnd, (LPRECT) &rc);

            // Get the client coordinates for the mouse click.

            pt.x = GET_X_LPARAM(lParam);
            pt.y = GET_Y_LPARAM(lParam);

            // If the mouse click took place inside the client
            // area, execute the application-defined function
            // that displays the shortcut menu.

            if (PtInRect((LPRECT) &rc, pt))
                HandlePopupMenu(hwnd, pt);
            break;
        // Process other window messages.
```

```
        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return NULL;
}


VOID APIENTRY HandlePopupMenu(HWND hwnd, POINT pt)
{
    HMENU hmenu;            // menu template
    HMENU hmenuTrackPopup;  // shortcut menu

    //  Load the menu template containing the shortcut menu from the
    //  application's resources.

    hmenu = LoadMenu(hinst, "PopupMenu");
    if (hmenu == NULL)
        return;

    // Get the first shortcut menu in the menu template. This is the
    // menu that TrackPopupMenu displays.

    hmenuTrackPopup = GetSubMenu(hmenu, 0);

    // TrackPopup uses screen coordinates, so convert the
    // coordinates of the mouse click to screen coordinates.

    ClientToScreen(hwnd, (LPPOINT) &pt);

    // Draw and track the shortcut menu.

    TrackPopupMenu(hmenuTrackPopup, TPM_LEFTALIGN | TPM_LEFTBUTTON,
        pt.x, pt.y, 0, hwnd, NULL);

    // Destroy the menu.

    DestroyMenu(hmenu);
}
```

Displaying a Shortcut Menu

The function shown in the following example displays a shortcut menu.

The application includes a menu resource identified by the string "ShortcutExample." The menu bar simply contains a menu name. The application uses the TrackPopupMenu function to display the menu associated with this menu item. (The menu bar itself is not displayed because **TrackPopupMenu** requires a handle to a menu, submenu, or shortcut menu.)

C++

```
VOID APIENTRY DisplayContextMenu(HWND hwnd, POINT pt)
{
    HMENU hmenu;            // top-level menu
    HMENU hmenuTrackPopup;  // shortcut menu

    // Load the menu resource.

    if ((hmenu = LoadMenu(hinst, "ShortcutExample")) == NULL)
        return;

    // TrackPopupMenu cannot display the menu bar so get
    // a handle to the first shortcut menu.

    hmenuTrackPopup = GetSubMenu(hmenu, 0);

    // Display the shortcut menu. Track the right mouse
    // button.

    TrackPopupMenu(hmenuTrackPopup,
            TPM_LEFTALIGN | TPM_RIGHTBUTTON,
            pt.x, pt.y, 0, hwnd, NULL);

    // Destroy the menu.

    DestroyMenu(hmenu);
}
```

Using Menu-Item Bitmaps

The system can use a bitmap instead of a text string to display a menu item. To use a bitmap, you must set the **MIIM_BITMAP** flag for the menu item and specify a handle to the bitmap that the system should display for the menu item in the **hbmpItem** member of the **MENUITEMINFO** structure. This section describes how to use bitmaps for menu items.

- Setting the Bitmap Type Flag
- Creating the Bitmap
- Adding Lines and Graphs to a Menu
- Example of Menu-Item Bitmaps

Setting the Bitmap Type Flag

The **MIIM_BITMAP** or **MF_BITMAP** flag tells the system to use a bitmap rather than a text string to display a menu item. A menu item's **MIIM_BITMAP** or **MF_BITMAP** flag must be set at run time; you cannot set it in the resource-definition file.

For new applications, you can use the SetMenuItemInfo or InsertMenuItem function to set the **MIIM_BITMAP** type flag. To change a menu item from a text item to a bitmap item, use **SetMenuItemInfo**. To add a new bitmap item to a menu, use the **InsertMenuItem** function.

Applications written for earlier versions of the system can continue to use the ModifyMenu, InsertMenu, or AppendMenu functions to set the **MF_BITMAP** flag. To change a menu item from a text string item to a bitmap item, use **ModifyMenu**. To add a new bitmap item to a menu, use the **MF_BITMAP** flag with the **InsertMenu** or **AppendMenu** function.

Creating the Bitmap

When you set the **MIIM_BITMAP** or **MF_BITMAP** type flag for a menu item, you must also specify a handle to the bitmap that the system should display for the menu item. You can provide the bitmap as a bitmap resource or create the bitmap at run time. If you use a bitmap resource, you can use the LoadBitmap function to load the bitmap and obtain its handle.

To create the bitmap at run time, use Windows Graphics Device Interface (GDI) functions. GDI provides several ways to create a bitmap at run time, but developers typically use the following method:

1. Use the CreateCompatibleDC function to create a device context compatible with the device context used by the application's main window.
2. Use the CreateCompatibleBitmap function to create a bitmap compatible with the application's main window or use the CreateBitmap function to create a monochrome bitmap.
3. Use the SelectObject function to select the bitmap into the compatible device context.
4. Use GDI drawing functions, such as Ellipse and LineTo, to draw an image into the bitmap.

For more information, see Bitmaps.

Adding Lines and Graphs to a Menu

The following code sample shows how to create a menu that contains menu-item bitmaps. It creates two menus. The first is a Chart menu that contains three menu-item bitmaps: a pie chart, a line chart, and a bar chart. The example demonstrates how to load these bitmaps from the application's resource file, and then use the CreatePopupMenu and AppendMenu functions to create the menu and menu items.

The second menu is a Lines menu. It contains bitmaps showing the line styles provided by the predefined pen in the system. The line-style bitmaps are created at run time by using GDI functions.

Here are the definitions of the bitmap resources in the application's resource-definition file.

C++

```
PIE BITMAP pie.bmp
LINE BITMAP line.bmp
BAR BITMAP bar.bmp
```

Here are the relevant portions of the application's header file.

C++

```
// Menu-item identifiers

#define IDM_SOLID       PS_SOLID
#define IDM_DASH        PS_DASH
#define IDM_DASHDOT     PS_DASHDOT
#define IDM_DASHDOTDOT  PS_DASHDOTDOT

#define IDM_PIE  1
#define IDM_LINE 2
#define IDM_BAR  3

// Line-type flags

#define SOLID       0
#define DOT         1
#define DASH        2
#define DASHDOT     3
#define DASHDOTDOT  4

// Count of pens

#define CPENS 5

// Chart-type flags

#define PIE  1
#define LINE 2
#define BAR  3

// Function prototypes

LRESULT APIENTRY MainWndProc(HWND, UINT, WPARAM, LPARAM);
VOID MakeChartMenu(HWND);
VOID MakeLineMenu(HWND, HPEN, HBITMAP);
```

The following example shows how menus and menu-item bitmaps are created in an application.

C++

```
LRESULT APIENTRY MainWndProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{

    static HPEN hpen[CPENS];
    static HBITMAP hbmp[CPENS];
    int i;

    switch (uMsg)
    {
        case WM_CREATE:

            // Create the Chart and Line menus.

            MakeChartMenu(hwnd);
            MakeLineMenu(hwnd, hpen, hbmp);
            return 0;

        // Process other window messages.

        case WM_DESTROY:

            for (i = 0; i < CPENS; i++)
            {
                DeleteObject(hbmp[i]);
                DeleteObject(hpen[i]);
            }

            PostQuitMessage(0);
            break;

        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return NULL;
}

VOID MakeChartMenu(HWND hwnd)
{
    HBITMAP hbmpPie;    // handle to pie chart bitmap
    HBITMAP hbmpLine;   // handle to line chart bitmap
    HBITMAP hbmpBar;    // handle to bar chart bitmap
    HMENU hmenuMain;    // handle to main menu
    HMENU hmenuChart;   // handle to Chart menu

    // Load the pie, line, and bar chart bitmaps from the
    // resource-definition file.
```

```
    hbmpPie = LoadBitmap(hinst, MAKEINTRESOURCE(PIE));
    hbmpLine = LoadBitmap(hinst, MAKEINTRESOURCE(LINE));
    hbmpBar = LoadBitmap(hinst, MAKEINTRESOURCE(BAR));

    // Create the Chart menu and add it to the menu bar.
    // Append the Pie, Line, and Bar menu items to the Chart
    // menu.

    hmenuMain = GetMenu(hwnd);
    hmenuChart = CreatePopupMenu();
    AppendMenu(hmenuMain, MF_STRING | MF_POPUP, (UINT) hmenuChart,
        "Chart");
    AppendMenu(hmenuChart, MF_BITMAP, IDM_PIE, (LPCTSTR) hbmpPie);
    AppendMenu(hmenuChart, MF_BITMAP, IDM_LINE,
        (LPCTSTR) hbmpLine);
    AppendMenu(hmenuChart, MF_BITMAP, IDM_BAR, (LPCTSTR) hbmpBar);

    return;
}

VOID MakeLineMenu(HWND hwnd, HPEN phpen, HBITMAP phbmp)
{
    HMENU hmenuLines;       // handle to Lines menu
    HMENU hmenu;            // handle to main menu
    COLORREF crMenuClr;     // menu-item background color
    HBRUSH hbrBackground;   // handle to background brush
    HBRUSH hbrOld;          // handle to previous brush
    WORD wLineX;            // width of line bitmaps
    WORD wLineY;            // height of line bitmaps
    HDC hdcMain;            // handle to main window's DC
    HDC hdcLines;           // handle to compatible DC
    HBITMAP hbmpOld;        // handle to previous bitmap
    int i;                  // loop counter

    // Create the Lines menu. Add it to the menu bar.

    hmenu = GetMenu(hwnd);
    hmenuLines = CreatePopupMenu();
    AppendMenu(hmenu, MF_STRING | MF_POPUP,
        (UINT) hmenuLines, "&Lines");

    // Create a brush for the menu-item background color.

    crMenuClr = GetSysColor(COLOR_MENU);
    hbrBackground = CreateSolidBrush(crMenuClr);

    // Create a compatible device context for the line bitmaps,
    // and then select the background brush into it.

    hdcMain = GetDC(hwnd);
    hdcLines = CreateCompatibleDC(hdcMain);
    hbrOld = SelectObject(hdcLines, hbrBackground);

    // Get the dimensions of the check-mark bitmap. The width of
    // the line bitmaps will be five times the width of the
```

```
    // check-mark bitmap.

    wLineX = GetSystemMetrics(SM_CXMENUCHECK) * (WORD) 5;
    wLineY = GetSystemMetrics(SM_CYMENUCHECK);

    // Create the bitmaps and select them, one at a time, into the
    // compatible device context. Initialize each bitmap by
    // filling it with the menu-item background color.

    for (i = 0; i < CPENS; i++)
    {
        phbmp[i] = CreateCompatibleBitmap(hdcMain, wLineX, wLineY);
        if (i == 0)
            hbmpOld = SelectObject(hdcLines, phbmp[i]);
        else
            SelectObject(hdcLines, phbmp[i]);
        ExtFloodFill(hdcLines, 0, 0, crMenuClr, FLOODFILLBORDER);
    }

    // Create the pens.

    phpen[0] = CreatePen(PS_SOLID, 1, RGB(0, 0, 0));
    phpen[1] = CreatePen(PS_DOT, 1, RGB(0, 0, 0));
    phpen[2] = CreatePen(PS_DASH, 1, RGB(0, 0, 0));
    phpen[3] = CreatePen(PS_DASHDOT, 1, RGB(0, 0, 0));
    phpen[4] = CreatePen(PS_DASHDOTDOT, 1, RGB(0, 0, 0));

    // Select a pen and a bitmap into the compatible device
    // context, draw a line into the bitmap, and then append
    // the bitmap as an item in the Lines menu.

    for (i = 0; i < CPENS; i++)
    {
        SelectObject(hdcLines, phbmp[i]);
        SelectObject(hdcLines, phpen[i]);
        MoveToEx(hdcLines, 0, wLineY / 2, NULL);
        LineTo(hdcLines, wLineX, wLineY / 2);
        AppendMenu(hmenuLines, MF_BITMAP, i + 1,
            (LPCTSTR) phbmp[i]);
    }

    // Release the main window's device context and destroy the
    // compatible device context. Also, destroy the background
    // brush.

    ReleaseDC(hwnd, hdcMain);
    SelectObject(hdcLines, hbrOld);
    DeleteObject(hbrBackground);
    SelectObject(hdcLines, hbmpOld);
    DeleteDC(hdcLines);

    return;
}
```

Example of Menu-Item Bitmaps

The example in this topic creates two menus, each containing several bitmap menu items. For each menu, the application adds a corresponding menu name to the main window's menu bar.

The first menu contains menu items showing each of three chart types: pie, line, and bar. The bitmaps for these menu items are defined as resources and are loaded by using the **LoadBitmap** function. Associated with this menu is a "Chart" menu name on the menu bar.

The second menu contains menu items showing each of the five line styles used with the **CreatePen** function: **PS_SOLID**, **PS_DASH**, **PS_DOT**, **PS_DASHDOT**, and **PS_DASHDOTDOT**. The application creates the bitmaps for these menu items at run time using GDI drawing functions. Associated with this menu is a **Lines** menu name on the menu bar.

Defined in the application's window procedure are two static arrays of bitmap handles. One array contains the handles of the three bitmaps used for the **Chart** menu. The other contains the handles of the five bitmaps used for the **Lines** menu. When processing the **WM_CREATE** message, the window procedure loads the chart bitmaps, creates the line bitmaps, and then adds the corresponding menu items. When processing the **WM_DESTROY** message, the window procedure deletes all of the bitmaps.

Following are the relevant portions of the application's header file.

C++

```
// Menu-item identifiers

#define IDM_PIE         1
#define IDM_LINE        2
#define IDM_BAR         3

#define IDM_SOLID       4
#define IDM_DASH        5
#define IDM_DASHDOT     6
#define IDM_DASHDOTDOT  7

// Number of items on the Chart and Lines menus

#define C_LINES         5
#define C_CHARTS        3

// Bitmap resource identifiers

#define IDB_PIE         1
#define IDB_LINE        2
```

```
#define IDB_BAR         3

// Dimensions of the line bitmaps

#define CX_LINEBMP      40
#define CY_LINEBMP      10
```

Following are the relevant portions of the window procedure. The window procedure performs most of its initialization by calling the application-defined LoadChartBitmaps, CreateLineBitmaps, and AddBitmapMenu functions, described later in this topic.

C++

```
LRESULT CALLBACK MainWindowProc(
        HWND hwnd,
        UINT uMsg,
        WPARAM wParam,
        LPARAM lParam
        )
{
    static HBITMAP aHbmLines[C_LINES];
    static HBITMAP aHbmChart[C_CHARTS];
    int i;

    switch (uMsg)
    {
        case WM_CREATE:

             // Call application-defined functions to load the
             // bitmaps for the Chart menu and create those for
             // the Lines menu.

            LoadChartBitmaps(aHbmChart);
            CreateLineBitmaps(aHbmLines);

             // Call an application-defined function to create
             // menus containing the bitmap menu items. The function
             // also adds a menu name to the window's menu bar.

            AddBitmapMenu(
                    hwnd,      // menu bar's owner window
                    "&Chart",  // text of menu name on menu bar
                    IDM_PIE,   // ID of first item on menu
                    aHbmChart, // array of bitmap handles
                    C_CHARTS   // number of items on menu
                    );
            AddBitmapMenu(hwnd, "&Lines", IDM_SOLID,
                    aHbmLines, C_LINES);
            break;

        case WM_DESTROY:
            for (i = 0; i < C_LINES; i++)
```

```
                DeleteObject(aHbmLines[i]);
            for (i = 0; i < C_CHARTS; i++)
                DeleteObject(aHbmChart[i]);
            PostQuitMessage(0);
            break;

        // Process additional messages here.

        default:
            return (DefWindowProc(hwnd, uMsg, wParam, lParam));
    }
    return 0;
}
```

The application-defined LoadChartBitmaps function loads the bitmap resources for the chart menu by calling the **LoadBitmap** function, as follows.

C++

```
VOID WINAPI LoadChartBitmaps(HBITMAP *paHbm)
{
    paHbm[0] = LoadBitmap(g_hinst, MAKEINTRESOURCE(IDB_PIE));
    paHbm[1] = LoadBitmap(g_hinst, MAKEINTRESOURCE(IDB_LINE));
    paHbm[2] = LoadBitmap(g_hinst, MAKEINTRESOURCE(IDB_BAR));
}
```

The application-defined CreateLineBitmaps function creates the bitmaps for the Lines menu by using GDI drawing functions. The function creates a memory device context (DC) with the same properties as the desktop window's DC. For each line style, the function creates a bitmap, selects it into the memory DC, and draws in it.

C++

```
VOID WINAPI CreateLineBitmaps(HBITMAP *paHbm)
{
    HWND hwndDesktop = GetDesktopWindow();
    HDC hdcDesktop = GetDC(hwndDesktop);
    HDC hdcMem = CreateCompatibleDC(hdcDesktop);
    COLORREF clrMenu = GetSysColor(COLOR_MENU);
    HBRUSH hbrOld;
    HPEN hpenOld;
    HBITMAP hbmOld;
    int fnDrawMode;
    int i;

     // Create a brush using the menu background color,
     // and select it into the memory DC.

    hbrOld = SelectObject(hdcMem, CreateSolidBrush(clrMenu));
```

```
     // Create the bitmaps. Select each one into the memory
     // DC that was created and draw in it.

    for (i = 0; i < C_LINES; i++)
    {
        // Create the bitmap and select it into the DC.

        paHbm[i] = CreateCompatibleBitmap(hdcDesktop,
                CX_LINEBMP, CY_LINEBMP);
        hbmOld = SelectObject(hdcMem, paHbm[i]);

        // Fill the background using the brush.

        PatBlt(hdcMem, 0, 0, CX_LINEBMP, CY_LINEBMP, PATCOPY);

        // Create the pen and select it into the DC.

        hpenOld = SelectObject(hdcMem,
                CreatePen(PS_SOLID + i, 1, RGB(0, 0, 0)));

         // Draw the line. To preserve the background color where
         // the pen is white, use the R2_MASKPEN drawing mode.

        fnDrawMode = SetROP2(hdcMem, R2_MASKPEN);
        MoveToEx(hdcMem, 0, CY_LINEBMP / 2, NULL);
        LineTo(hdcMem, CX_LINEBMP, CY_LINEBMP / 2);
        SetROP2(hdcMem, fnDrawMode);

        // Delete the pen, and select the old pen and bitmap.

        DeleteObject(SelectObject(hdcMem, hpenOld));
        SelectObject(hdcMem, hbmOld);
    }

    // Delete the brush and select the original brush.

    DeleteObject(SelectObject(hdcMem, hbrOld));

    // Delete the memory DC and release the desktop DC.

    DeleteDC(hdcMem);
    ReleaseDC(hwndDesktop, hdcDesktop);
}
```

The application-defined AddBitmapMenu function creates a menu and adds the specified number of bitmap menu items to it. Then it adds a corresponding menu name to the specified window's menu bar.

C++

```
VOID WINAPI AddBitmapMenu(
        HWND hwnd,          // window that owned the menu bar
```

```
        LPSTR lpszText,     // text of menu name on menu bar
        UINT uID,           // ID of first bitmap menu item
        HBITMAP *paHbm,     // bitmaps for the menu items
        int cItems)         // number bitmap menu items
{
    HMENU hmenuBar = GetMenu(hwnd);
    HMENU hmenuPopup = CreatePopupMenu();
    MENUITEMINFO mii;
    int i;

    // Add the bitmap menu items to the menu.

    for (i = 0; i < cItems; i++)
    {
        mii.fMask = MIIM_ID | MIIM_BITMAP | MIIM_DATA;
        mii.wID = uID + i;
        mii.hbmpItem = &paHbm[i];
        InsertMenuItem(hmenuPopup, i, TRUE, &mii);
    }

    // Add a menu name to the menu bar.

    mii.fMask = MIIM_STRING | MIIM_DATA | MIIM_SUBMENU;
    mii.fType = MFT_STRING;
    mii.hSubMenu = hmenuPopup;
    mii.dwTypeData = lpszText;
    InsertMenuItem(hmenuBar,
        GetMenuItemCount(hmenuBar), TRUE, &mii);
}
```

Creating Owner-Drawn Menu Items

If you need complete control over the appearance of a menu item, you can use an owner-drawn menu item in your application. This section describes the steps involved in creating and using an owner-drawn menu item.

- Setting the Owner-Drawn Flag
- Owner-Drawn Menus and the WM_MEASUREITEM Message
- Owner-Drawn Menus and the WM_DRAWITEM Message
- Owner-Drawn Menus and the WM_MENUCHAR Message
- Setting Fonts for Menu-Item Text Strings
- Example of Owner-Drawn Menu Items

Setting the Owner-Drawn Flag

You cannot define an owner-drawn menu item in your application's resource-definition file. Instead, you must create a new menu item or modify an existing one by using the

MFT_OWNERDRAW menu flag.

You can use the **InsertMenuItem** or **SetMenuItemInfo** function to specify an owner-drawn menu item. Use **InsertMenuItem** to insert a new menu item at the specified position in a menu bar or menu. Use **SetMenuItemInfo** to change the contents of a menu.

When calling these two functions, you must specify a pointer to a **MENUITEMINFO** structure, which specifies the properties of the new menu item or the properties you want to change for an existing menu item. To make an item an owner-drawn item, specify the **MIIM_FTYPE** value for the **fMask** member and the **MFT_OWNERDRAW** value for the **fType** member.

By setting the appropriate members of the **MENUITEMINFO** structure, you can associate an application-defined value, which is called **item data**, with each menu item. To do so, specify the **MIIM_DATA** value for the **fMask** member and the application-defined value for the **dwItemData** member.

You can use item data with any type of menu item, but it is particularly useful for owner-drawn items. For example, suppose a structure contains information used to draw a menu item. An application might use the item data for a menu item to store a pointer to the structure. The item data is sent to the menu's owner window with the **WM_MEASUREITEM** and **WM_DRAWITEM** messages. To retrieve the item data for a menu at any time, use the **GetMenuItemInfo** function.

Applications written for earlier versions of the system can continue to call **AppendMenu**, **InsertMenu**, or **ModifyMenu** to assign the **MF_OWNERDRAW** flag to an owner-drawn menu item.

When you call any of these three functions, you can pass a value as the *lpNewItem* parameter. This value can represent any information that is meaningful to your application, and that will be available to your application when the item is to be displayed. For example, the value could contain a pointer to a structure; the structure, in turn, might contain a text string and a handle to the logical font your application will use to draw the string.

Owner-Drawn Menus and the WM_MEASUREITEM Message

Before the system displays an owner-drawn menu item for the first time, it sends the **WM_MEASUREITEM** message to the window procedure of the window that owns the item's menu. This message contains a pointer to a **MEASUREITEMSTRUCT** structure that

identifies the item and contains the item data that an application may have assigned to it. The window procedure must fill the **itemWidth** and **itemHeight** members of the structure before returning from processing the message. The system uses the information in these members when creating the bounding rectangle in which an application draws the menu item. It also uses the information to detect when the user chooses the item.

Owner-Drawn Menus and the WM_DRAWITEM Message

Whenever the item must be drawn (for example, when it is first displayed or when the user selects it), the system sends the WM_DRAWITEM message to the window procedure of the menu's owner window. This message contains a pointer to a DRAWITEMSTRUCT structure, which contains information about the item, including the item data that an application may have assigned to it. In addition, **DRAWITEMSTRUCT** contains flags that indicate the state of the item (such as whether it is grayed or selected) as well as a bounding rectangle and a device context that the application uses to draw the item.

An application must do the following while processing the WM_DRAWITEM message:

- Determine the type of drawing that is necessary. To do so, check the **itemAction** member of the DRAWITEMSTRUCT structure.
- Draw the menu item appropriately, using the bounding rectangle and device context obtained from the DRAWITEMSTRUCT structure. The application must draw only within the bounding rectangle. For performance reasons, the system does not clip portions of the image that are drawn outside the rectangle.
- Restore all GDI objects selected for the menu item's device context.

If the user selects the menu item, the system sets the **itemAction** member of the DRAWITEMSTRUCT structure to the **ODA_SELECT** value and sets the **ODS_SELECTED** value in the **itemState** member. This is an application's cue to redraw the menu item to indicate that it is selected.

Owner-Drawn Menus and the WM_MENUCHAR Message

Menus other than owner-drawn menus can specify a menu mnemonic by inserting an underscore next to a character in the menu string. This allows the user to select the menu by pressing ALT and pressing the menu mnemonic character. In owner-drawn menus, however, you cannot specify a menu mnemonic in this manner. Instead, your application must process the WM_MENUCHAR message to provide owner-drawn menus with menu mnemonics.

The WM_MENUCHAR message is sent when the user types a menu mnemonic that does not match any of the predefined mnemonics of the current menu. The value contained in *wParam* specifies the ASCII character that corresponds to the key the user pressed with the ALT key. The low-order word of *wParam* specifies the type of the selected menu and can be one of the following values:

- **MF_POPUP** if the current menu is a submenu.
- **MF_SYSMENU** if the menu is the system menu.

The high-order word of *wParam* contains the menu handle to the current menu. The window with the owner-drawn menus can process WM_MENUCHAR as follows:

```
C++
```

```
   case WM_MENUCHAR:
      nIndex = Determine index of menu item to be selected from
               character that was typed and handle to the current
               menu.
      return MAKELRESULT(nIndex, 2);
```

The two in the high-order word of the return value informs the system that the low-order word of the return value contains the zero-based index of the menu item to be selected.

The following constants correspond to the possible return values from the WM_MENUCHAR message.

Constant	Value	Meaning
MNC_IGNORE	0	The system should discard the character the user pressed and create a short beep on the system speaker.
MNC_CLOSE	1	The system should close the active menu.
MNC_EXECUTE	2	The system should choose the item specified in the low-order word of the return value. The owner window receives a WM_COMMAND message.
MNC_SELECT	3	The system should select the item specified in the low-order word of the return value.

Setting Fonts for Menu-Item Text Strings

This topic contains an example from an application that uses owner-drawn menu items in a menu. The menu contains items that set the attributes of the current font, and the items are displayed using the appropriate font attribute.

Here is how the menu is defined in the resource-definition file. Note that the strings for the Regular, Bold, Italic, and Underline menu items are assigned at run time, so their strings are empty in the resource-definition file.

C++

```
MainMenu MENU
BEGIN
    POPUP   "&Character"
    BEGIN
        MENUITEM    "",    IDM_REGULAR
        MENUITEM SEPARATOR
        MENUITEM    "",    IDM_BOLD
        MENUITEM    "",    IDM_ITALIC
        MENUITEM    "",    IDM_ULINE
    END
END
```

The application's window procedure processes the messages involved in using owner-drawn menu items. The application uses the WM_CREATE message to do the following:

- Set the **MF_OWNERDRAW** flag for the menu items.
- Set the text strings for the menu items.
- Obtain handles of the fonts used to draw the items.
- Obtain the text and background color values for selected menu items.

The text strings and font handles are stored in an array of application-defined MYITEM structures. The application-defined GetAFont function creates a font that corresponds to the specified font attribute and returns a handle to the font. The handles are destroyed during the processing of the WM_DESTROY message.

During the processing of the WM_MEASUREITEM message, the example gets the width and height of a menu-item string and copies these values into the MEASUREITEMSTRUCT structure. The system uses the width and height values to calculate the size of the menu.

During the processing of the WM_DRAWITEM message, the menu item's string is drawn with room left next to the string for the check-mark bitmap. If the user selects the item, the selected text and background colors are used to draw the item.

C++

```
LRESULT APIENTRY MainWndProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{

    typedef struct _MYITEM
    {
        HFONT hfont;
        LPSTR psz;
    } MYITEM;             // structure for item font and string

    MYITEM *pmyitem;      // pointer to item's font and string
    static MYITEM myitem[CITEMS];   // array of MYITEMS
    static HMENU hmenu;             // handle to main menu
    static COLORREF crSelText;  // text color of selected item
    static COLORREF crSelBkgnd; // background color of selected item
    COLORREF crText;            // text color of unselected item
    COLORREF crBkgnd;           // background color unselected item
    LPMEASUREITEMSTRUCT lpmis;  // pointer to item of data
    LPDRAWITEMSTRUCT lpdis;     // pointer to item drawing data
    HDC hdc;                    // handle to screen DC
    SIZE size;                  // menu-item text extents
    WORD wCheckX;               // check-mark width
    int nTextX;                 // width of menu item
    int nTextY;                 // height of menu item
    int i;                      // loop counter
    HFONT hfontOld;             // handle to old font
    BOOL fSelected = FALSE;     // menu-item selection flag
    size_t * pcch;
    HRESULT hResult;

    switch (uMsg)
    {
        case WM_CREATE:

            // Modify the Regular, Bold, Italic, and Underline
            // menu items to make them owner-drawn items. Associate
            // a MYITEM structure with each item to contain the
            // string for and font handle to each item.

            hmenu = GetMenu(hwnd);
            ModifyMenu(hmenu, IDM_REGULAR, MF_BYCOMMAND |
                MF_CHECKED | MF_OWNERDRAW, IDM_REGULAR,
                (LPTSTR) &myitem[REGULAR]);
            ModifyMenu(hmenu, IDM_BOLD, MF_BYCOMMAND |
                MF_OWNERDRAW, IDM_BOLD, (LPTSTR) &myitem[BOLD]);
            ModifyMenu(hmenu, IDM_ITALIC, MF_BYCOMMAND |
                MF_OWNERDRAW, IDM_ITALIC,
                (LPTSTR) &myitem[ITALIC]);
            ModifyMenu(hmenu, IDM_ULINE, MF_BYCOMMAND |
                MF_OWNERDRAW, IDM_ULINE, (LPTSTR) &myitem[ULINE]);

            // Retrieve each item's font handle and copy it into
            // the hfont member of each item's MYITEM structure.
            // Also, copy each item's string into the structures.
```

```
            myitem[REGULAR].hfont = GetAFont(REGULAR);
            myitem[REGULAR].psz = "Regular";
            myitem[BOLD].hfont = GetAFont(BOLD);
            myitem[BOLD].psz = "Bold";
            myitem[ITALIC].hfont = GetAFont(ITALIC);
            myitem[ITALIC].psz = "Italic";
            myitem[ULINE].hfont = GetAFont(ULINE);
            myitem[ULINE].psz = "Underline";

            // Retrieve the text and background colors of the
            // selected menu text.

            crSelText = GetSysColor(COLOR_HIGHLIGHTTEXT);
            crSelBkgnd = GetSysColor(COLOR_HIGHLIGHT);

            return 0;

        case WM_MEASUREITEM:

            // Retrieve a device context for the main window.

            hdc = GetDC(hwnd);

            // Retrieve pointers to the menu item's
            // MEASUREITEMSTRUCT structure and MYITEM structure.

            lpmis = (LPMEASUREITEMSTRUCT) lParam;
            pmyitem = (MYITEM *) lpmis->itemData;

            // Select the font associated with the item into
            // the main window's device context.

            hfontOld = SelectObject(hdc, pmyitem->hfont);

            // Retrieve the width and height of the item's string,
            // and then copy the width and height into the
            // MEASUREITEMSTRUCT structure's itemWidth and
            // itemHeight members.

            hResult = StringCchLength(pmyitem->psz,STRSAFE_MAX_CCH, pcch);
            if (FAILED(hResult))
            {
            // Add code to fail as securely as possible.
                return;
            }

            GetTextExtentPoint32(hdc, pmyitem->psz,
                *pcch, &size);
            lpmis->itemWidth = size.cx;
            lpmis->itemHeight = size.cy;

            // Select the old font back into the device context,
            // and then release the device context.
```

```
            SelectObject(hdc, hfontOld);
            ReleaseDC(hwnd, hdc);

            return TRUE;

            break;

        case WM_DRAWITEM:

            // Get pointers to the menu item's DRAWITEMSTRUCT
            // structure and MYITEM structure.

            lpdis = (LPDRAWITEMSTRUCT) lParam;
            pmyitem = (MYITEM *) lpdis->itemData;

            // If the user has selected the item, use the selected
            // text and background colors to display the item.

            if (lpdis->itemState & ODS_SELECTED)
            {
                crText = SetTextColor(lpdis->hDC, crSelText);
                crBkgnd = SetBkColor(lpdis->hDC, crSelBkgnd);
                fSelected = TRUE;
            }

            // Remember to leave space in the menu item for the
            // check-mark bitmap. Retrieve the width of the bitmap
            // and add it to the width of the menu item.

            wCheckX = GetSystemMetrics(SM_CXMENUCHECK);
            nTextX = wCheckX + lpdis->rcItem.left;
            nTextY = lpdis->rcItem.top;

            // Select the font associated with the item into the
            // item's device context, and then draw the string.

            hfontOld = SelectObject(lpdis->hDC, pmyitem->hfont);

            hResult = StringCchLength(pmyitem->psz,STRSAFE_MAX_CCH, pcch);
            if (FAILED(hResult))
            {
            // Add code to fail as securely as possible.
                return;
            }

            ExtTextOut(lpdis->hDC, nTextX, nTextY, ETO_OPAQUE,
                &lpdis->rcItem, pmyitem->psz,
                *pcch, NULL);

            // Select the previous font back into the device
            // context.

            SelectObject(lpdis->hDC, hfontOld);

            // Return the text and background colors to their
```

```
            // normal state (not selected).

            if (fSelected)
            {
                SetTextColor(lpdis->hDC, crText);
                SetBkColor(lpdis->hDC, crBkgnd);
            }

            return TRUE;

        // Process other messages.

        case WM_DESTROY:

            // Destroy the menu items' font handles.

            for (i = 0; i < CITEMS; i++)
                DeleteObject(myitem[i].hfont);

            PostQuitMessage(0);
            break;

        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return NULL;
}

HFONT GetAFont(int fnFont)
{
    static LOGFONT lf;  // structure for font information

    // Get a handle to the ANSI fixed-pitch font, and copy
    // information about the font to a LOGFONT structure.

    GetObject(GetStockObject(ANSI_FIXED_FONT), sizeof(LOGFONT),
        &lf);

    // Set the font attributes, as appropriate.

    if (fnFont == BOLD)
        lf.lfWeight = FW_BOLD;
    else
        lf.lfWeight = FW_NORMAL;

    lf.lfItalic = (fnFont == ITALIC);
    lf.lfItalic = (fnFont == ULINE);

    // Create the font, and then return its handle.

    return CreateFont(lf.lfHeight, lf.lfWidth,
        lf.lfEscapement, lf.lfOrientation, lf.lfWeight,
        lf.lfItalic, lf.lfUnderline, lf.lfStrikeOut, lf.lfCharSet,
        lf.lfOutPrecision, lf.lfClipPrecision, lf.lfQuality,
```

```
        lf.lfPitchAndFamily, lf.lfFaceName);
}
```

Example of Owner-Drawn Menu Items

The example in this topic uses owner-drawn menu items in a menu. The menu items select specific font attributes, and the application displays each menu item using a font that has the corresponding attribute. For example, the **Italic** menu item is displayed in an italic font. The **Character** menu name on the menu bar opens the menu.

The menu bar and drop-down menu are defined initially by an extended menu-template resource. Because a menu template cannot specify owner-drawn items, the menu initially contains four text menu items with the following strings: "Regular," "Bold," "Italic," and "Underline." The application's window procedure changes these to owner-drawn items when it processes the WM_CREATE message. When it receives the **WM_CREATE** message, the window procedure calls the application-defined OnCreate function, which performs the following steps for each menu item:

- Allocates an application-defined MYITEM structure.
- Gets the text of the menu item and saves it in the application-defined MYITEM structure.
- Creates the font used to display the menu item and saves its handle in the application-defined MYITEM structure.
- Changes the menu item type to **MFT_OWNERDRAW** and saves a pointer to the application-defined MYITEM structure as item data.

Because a pointer to each application-defined MYITEM structure is saved as item data, it is passed to the window procedure in conjunction with the WM_MEASUREITEM and WM_DRAWITEM messages for the corresponding menu item. The pointer is contained in the **itemData** member of both the MEASUREITEMSTRUCT and DRAWITEMSTRUCT structures.

A WM_MEASUREITEM message is sent for each owner-drawn menu item the first time it is displayed. The application processes this message by selecting the font for the menu item into a device context and then determining the space required to display the menu item text in that font. The font and menu item text are both specified by the menu item's `MYITEM` structure (the structure defined by the application). The application determines the size of the text by using the GetTextExtentPoint32 function.

The window procedure processes the WM_DRAWITEM message by displaying the menu item text in the appropriate font. The font and menu item text are both specified by the

menu item's `MYITEM` structure. The application selects text and background colors appropriate to the menu item's state.

The window procedure processes the **WM_DESTROY** message to destroy fonts and free memory. The application deletes the font and frees the application-defined MYITEM structure for each menu item.

The following are the relevant portions of the application's header file.

C++

```
// Menu-item identifiers for the Character menu

#define IDM_CHARACTER 10
#define IDM_REGULAR   11
#define IDM_BOLD      12
#define IDM_ITALIC    13
#define IDM_UNDERLINE 14

// Structure associated with menu items

typedef struct tagMYITEM
{
    HFONT hfont;
    int   cchItemText;
    char  szItemText[1];
} MYITEM;

#define CCH_MAXITEMTEXT 256
```

The following are the relevant portions of the application's window procedure and its associated functions.

C++

```
LRESULT CALLBACK MainWindowProc(
        HWND hwnd,
        UINT uMsg,
        WPARAM wParam,
        LPARAM lParam
        )
{
    switch (uMsg)
    {
        case WM_CREATE:
            if (!OnCreate(hwnd))
                return -1;
            break;

        case WM_DESTROY:
```

```
            OnDestroy(hwnd);
            PostQuitMessage(0);
            break;

        case WM_MEASUREITEM:
            OnMeasureItem(hwnd, (LPMEASUREITEMSTRUCT) lParam);
            return TRUE;

        case WM_DRAWITEM:
            OnDrawItem(hwnd, (LPDRAWITEMSTRUCT) lParam);
            return TRUE;

        // Additional message processing goes here.

        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}


BOOL WINAPI OnCreate(HWND hwnd)
{
    HMENU hmenuBar = GetMenu(hwnd);
    HMENU hmenuPopup;
    MENUITEMINFO mii;
    UINT uID;
    MYITEM *pMyItem;

    // Get a handle to the pop-up menu.

    mii.fMask = MIIM_SUBMENU;     // information to get
    GetMenuItemInfo(hmenuBar, IDM_CHARACTER, FALSE, &mii);
    hmenuPopup = mii.hSubMenu;

    // Modify each menu item. Assume that the IDs IDM_REGULAR
    // through IDM_UNDERLINE are consecutive numbers.

    for (uID = IDM_REGULAR; uID <= IDM_UNDERLINE; uID++)
    {
         // Allocate an item structure, leaving space for a
         // string of up to CCH_MAXITEMTEXT characters.

        pMyItem = (MYITEM *) LocalAlloc(LMEM_FIXED,
                sizeof(MYITEM) + CCH_MAXITEMTEXT);

        // Save the item text in the item structure.

        mii.fMask = MIIM_STRING;
        mii.dwTypeData = pMyItem->szItemText;
        mii.cch = CCH_MAXITEMTEXT;
        GetMenuItemInfo(hmenuPopup, uID, FALSE, &mii);
        pMyItem->cchItemText = mii.cch;

        // Reallocate the structure to the minimum required size.
```

```
        pMyItem = (MYITEM *) LocalReAlloc(pMyItem,
                sizeof(MYITEM) + mii.cch, LMEM_MOVEABLE);

        // Create the font used to draw the item.

        pMyItem->hfont = CreateMenuItemFont(uID);

        // Change the item to an owner-drawn item, and save
        // the address of the item structure as item data.

        mii.fMask = MIIM_FTYPE | MIIM_DATA;
        mii.fType = MFT_OWNERDRAW;
        mii.dwItemData = (ULONG_PTR) pMyItem;
        SetMenuItemInfo(hmenuPopup, uID, FALSE, &mii);
    }
    return TRUE;
}

HFONT CreateMenuItemFont(UINT uID)
{
    LOGFONT lf;
    HRESULT hr;

    ZeroMemory(&lf, sizeof(lf));
    lf.lfHeight = 20;
    hr = StringCchCopy(lf.lfFaceName, 32, "Times New Roman");
    if (FAILED(hr))
    {
    // TODO: writer error handler
    }

    switch (uID)
    {
        case IDM_BOLD:
            lf.lfWeight = FW_HEAVY;
            break;

        case IDM_ITALIC:
            lf.lfItalic = TRUE;
            break;

        case IDM_UNDERLINE:
            lf.lfUnderline = TRUE;
            break;
    }
    return CreateFontIndirect(&lf);
}

VOID WINAPI OnDestroy(HWND hwnd)
{
    HMENU hmenuBar = GetMenu(hwnd);
    HMENU hmenuPopup;
    MENUITEMINFO mii;
    UINT uID;
```

```
    MYITEM *pMyItem;

    // Get a handle to the menu.

    mii.fMask = MIIM_SUBMENU;     // information to get
    GetMenuItemInfo(hmenuBar, IDM_CHARACTER, FALSE, &mii);
    hmenuPopup = mii.hSubMenu;

    // Free resources associated with each menu item.

    for (uID = IDM_REGULAR; uID <= IDM_UNDERLINE; uID++)
    {
        // Get the item data.

        mii.fMask = MIIM_DATA;
        GetMenuItemInfo(hmenuPopup, uID, FALSE, &mii);
        pMyItem = (MYITEM *) mii.dwItemData;

        // Destroy the font and free the item structure.

        DeleteObject(pMyItem->hfont);
        LocalFree(pMyItem);
    }
}

VOID WINAPI OnMeasureItem(HWND hwnd, LPMEASUREITEMSTRUCT lpmis)
{
    MYITEM *pMyItem = (MYITEM *) lpmis->itemData;
    HDC hdc = GetDC(hwnd);
    HFONT hfntOld = (HFONT)SelectObject(hdc, pMyItem->hfont);
    SIZE size;

    GetTextExtentPoint32(hdc, pMyItem->szItemText,
            pMyItem->cchItemText, &size);

    lpmis->itemWidth = size.cx;
    lpmis->itemHeight = size.cy;

    SelectObject(hdc, hfntOld);
    ReleaseDC(hwnd, hdc);
}

VOID WINAPI OnDrawItem(HWND hwnd, LPDRAWITEMSTRUCT lpdis)
{
    MYITEM *pMyItem = (MYITEM *) lpdis->itemData;
    COLORREF clrPrevText, clrPrevBkgnd;
    HFONT hfntPrev;
    int x, y;

    // Set the appropriate foreground and background colors.

    if (lpdis->itemState & ODS_SELECTED)
    {
        clrPrevText = SetTextColor(lpdis->hDC,
                GetSysColor(COLOR_HIGHLIGHTTEXT));
```

```
        clrPrevBkgnd = SetBkColor(lpdis->hDC,
                GetSysColor(COLOR_HIGHLIGHT));
    }
    else
    {
        clrPrevText = SetTextColor(lpdis->hDC,
                GetSysColor(COLOR_MENUTEXT));
        clrPrevBkgnd = SetBkColor(lpdis->hDC,
                GetSysColor(COLOR_MENU));
    }

    // Determine where to draw and leave space for a check mark.

    x = lpdis->rcItem.left;
    y = lpdis->rcItem.top;
    x += GetSystemMetrics(SM_CXMENUCHECK);

    // Select the font and draw the text.

    hfntPrev = (HFONT)SelectObject(lpdis->hDC, pMyItem->hfont);
    ExtTextOut(lpdis->hDC, x, y, ETO_OPAQUE,
            &lpdis->rcItem, pMyItem->szItemText,
            pMyItem->cchItemText, NULL);

    // Restore the original font and colors.

    SelectObject(lpdis->hDC, hfntPrev);
    SetTextColor(lpdis->hDC, clrPrevText);
    SetBkColor(lpdis->hDC, clrPrevBkgnd);
}
```

Using Custom Check Mark Bitmaps

The system provides a default check-mark bitmap for displaying next to a menu item that is selected. You can customize an individual menu item by providing a pair of bitmaps to replace the default check-mark bitmap. The system displays one bitmap when the item is selected and the other when it is clear. This section describes the steps involved in creating and using custom check-mark bitmaps.

- Creating Custom Check Mark Bitmaps
- Associating Bitmaps with a Menu Item
- Setting the Check-mark Attribute
- Simulating Check Boxes in a Menu
- Example of Using Custom Check-mark Bitmaps

Creating Custom Check Mark Bitmaps

A custom check-mark bitmap must be the same size as the default check-mark bitmap. You can retrieve the default check-mark size of the bitmap by calling the GetSystemMetrics function. The low-order word of this function's return value specifies the width; the high-order word specifies the height.

You can use bitmap resources to provide check-mark bitmaps. However, because the required bitmap size varies depending on the display type, you may need to resize the bitmap at run time by using the StretchBlt function. Depending on the bitmap, the distortion caused by sizing could produce unacceptable results.

Instead of using a bitmap resource, you can create a bitmap at run time by using GDI functions.

To create a bitmap at run time

1. Use the CreateCompatibleDC function to create a device context compatible with the one used by the application's main window.

   The function's *hdc* parameter can specify either **NULL** or the return value from the function. CreateCompatibleDC returns a handle to the compatible device context.

2. Use the CreateCompatibleBitmap function to create a bitmap compatible with the application's main window.

   This function's *nWidth* and *nHeight* parameters set the size of the bitmap; they should specify the width and height information returned by the GetSystemMetrics function.

   > ⓘ **Note**
   >
   > You can also use the **CreateBitmap** function to create a monochrome bitmap.

3. Use the SelectObject function to select the bitmap into the compatible device context.

4. Use GDI drawing functions, such as Ellipse and LineTo, to draw an image into the bitmap, or use functions such as BitBlt and StretchBlt to copy an image into the bitmap.

For more information, see Bitmaps.

Associating Bitmaps with a Menu Item

You associate a pair of check-mark bitmaps with a menu item by passing the handles of the bitmaps to the **SetMenuItemBitmaps** function. The *hBitmapUnchecked* parameter identifies the clear bitmap, and the *hBitmapChecked* parameter identifies the selected bitmap. If you want to remove one or both check marks from a menu item, you can set the *hBitmapUnchecked* or *hBitmapChecked* parameter, or both, to **NULL**.

Setting the Check-mark Attribute

The **CheckMenuItem** function sets a menu item's check-mark attribute to either selected or cleared. You can specify the **MF_CHECKED** value to set the check-mark attribute to selected and the **MF_UNCHECKED** value to set it to clear.

You can also set the check state of a menu item by using the **SetMenuItemInfo** function.

Sometimes a group of menu items represents a set of mutually exclusive options. By using the **CheckMenuRadioItem** function, you can check one menu item while simultaneously removing the check mark from all other menu items in the group.

Simulating Check Boxes in a Menu

This topic contains an example that shows how to simulate check boxes in a menu. The example contains a Character menu whose items allow the user to set the bold, italic, and underline attributes of the current font. When a font attribute is in effect, a check mark is displayed in the check box next to the corresponding menu item; otherwise, an empty check box is displayed next to the item.

The example replaces the default check-mark bitmap with two bitmaps: a bitmap with a selected check box and the bitmap with an empty box. The selected check box bitmap is displayed next to the Bold, Italic, or Underline menu item when the item's check-mark attribute is set to **MF_CHECKED**. The clear or empty check box bitmap is displayed when the check-mark attribute is set to **MF_UNCHECKED**.

The system provides a predefined bitmap that contains the images used for check boxes and radio buttons. The example isolates the selected and empty check boxes, copies them to two separate bitmaps, and then uses them as the selected and cleared bitmaps for items in the **Character** menu.

To retrieve a handle to the system-defined check box bitmap, the example calls the **LoadBitmap** function, specifying **NULL** as the *hInstance* parameter and **OBM_CHECKBOXES** as the *lpBitmapName* parameter. Because the images in the bitmap are all the same size, the example can isolate them by dividing the bitmap width and height by the number of images in its rows and columns.

The following portion of a resource-definition file shows how the menu items in the **Character** menu are defined. Note that no font attributes are in effect initially, so the check-mark attribute for the **Regular** item is set to selected and, by default, the check-mark attribute of the remaining items is set to clear.

C++

```
#include "men3.h"

MainMenu MENU
BEGIN
    POPUP   "&Character"
    BEGIN
        MENUITEM    "&Regular",     IDM_REGULAR, CHECKED
        MENUITEM SEPARATOR
        MENUITEM    "&Bold",        IDM_BOLD
        MENUITEM    "&Italic",      IDM_ITALIC
        MENUITEM    "&Underline",   IDM_ULINE
    END
END
```

Here are the relevant contents of the application's header file.

C++

```
// Menu-item identifiers

#define IDM_REGULAR 0x1
#define IDM_BOLD    0x2
#define IDM_ITALIC  0x4
#define IDM_ULINE   0x8

// Check-mark flags

#define CHECK   1
#define UNCHECK 2

// Font-attribute mask

#define ATTRIBMASK 0xe

// Function prototypes

LRESULT APIENTRY MainWndProc(HWND, UINT, WPARAM, LPARAM);
HBITMAP GetMyCheckBitmaps(UINT);
BYTE CheckOrUncheckMenuItem(BYTE, HMENU);
```

The following example shows the portions of the window procedure that create the check-mark bitmaps; set the check-mark attribute of the **Bold**, **Italic**, and **Underline** menu items; and destroy check-mark bitmaps.

C++

```
LRESULT APIENTRY MainWndProc(HWND hwndMain, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{

    static HBITMAP hbmpCheck;   // handle to checked bitmap
    static HBITMAP hbmpUncheck; // handle to unchecked bitmap
    static HMENU hmenu;         // handle to main menu
    BYTE fbFontAttrib;          // font-attribute flags

    switch (uMsg)
    {
        case WM_CREATE:

            // Call the application-defined GetMyCheckBitmaps
            // function to get the predefined checked and
            // unchecked check box bitmaps.

            hbmpCheck = GetMyCheckBitmaps(CHECK);
            hbmpUncheck = GetMyCheckBitmaps(UNCHECK);

            // Set the checked and unchecked bitmaps for the menu
            // items.

            hmenu = GetMenu(hwndMain);
            SetMenuItemBitmaps(hmenu, IDM_BOLD, MF_BYCOMMAND,
                hbmpUncheck, hbmpCheck);
            SetMenuItemBitmaps(hmenu, IDM_ITALIC, MF_BYCOMMAND,
                hbmpUncheck, hbmpCheck);
            SetMenuItemBitmaps(hmenu, IDM_ULINE, MF_BYCOMMAND,
                hbmpUncheck, hbmpCheck);

            return 0;

        case WM_COMMAND:
            switch (LOWORD(wParam))
            {
                // Process the menu commands.

                case IDM_REGULAR:
                case IDM_BOLD:
                case IDM_ITALIC:
                case IDM_ULINE:

                    // CheckOrUncheckMenuItem is an application-
                    // defined function that sets the menu item
                    // checkmarks and returns the user-selected
                    // font attributes.

                    fbFontAttrib = CheckOrUncheckMenuItem(
                        (BYTE) LOWORD(wParam), hmenu);

                    // Set the font attributes.
```

```
                    return 0;

                // Process other command messages.

                default:
                    break;
            }

            break;

        // Process other window messages.

        case WM_DESTROY:

            // Destroy the checked and unchecked bitmaps.

            DeleteObject(hbmpCheck);
            DeleteObject(hbmpUncheck);

            PostQuitMessage(0);
            break;

        default:
            return DefWindowProc(hwndMain, uMsg, wParam, lParam);
    }
    return NULL;
}

HBITMAP GetMyCheckBitmaps(UINT fuCheck)
{
    COLORREF crBackground;  // background color
    HBRUSH hbrBackground;   // background brush
    HBRUSH hbrTargetOld;    // original background brush
    HDC hdcSource;          // source device context
    HDC hdcTarget;          // target device context
    HBITMAP hbmpCheckboxes; // handle to check-box bitmap
    BITMAP bmCheckbox;      // structure for bitmap data
    HBITMAP hbmpSourceOld;  // handle to original source bitmap
    HBITMAP hbmpTargetOld;  // handle to original target bitmap
    HBITMAP hbmpCheck;      // handle to check-mark bitmap
    RECT rc;                // rectangle for check-box bitmap
    WORD wBitmapX;          // width of check-mark bitmap
    WORD wBitmapY;          // height of check-mark bitmap

    // Get the menu background color and create a solid brush
    // with that color.

    crBackground = GetSysColor(COLOR_MENU);
    hbrBackground = CreateSolidBrush(crBackground);

    // Create memory device contexts for the source and
    // destination bitmaps.

    hdcSource = CreateCompatibleDC((HDC) NULL);
```

```
    hdcTarget = CreateCompatibleDC(hdcSource);

    // Get the size of the system default check-mark bitmap and
    // create a compatible bitmap of the same size.

    wBitmapX = GetSystemMetrics(SM_CXMENUCHECK);
    wBitmapY = GetSystemMetrics(SM_CYMENUCHECK);

    hbmpCheck = CreateCompatibleBitmap(hdcSource, wBitmapX,
        wBitmapY);

    // Select the background brush and bitmap into the target DC.

    hbrTargetOld = SelectObject(hdcTarget, hbrBackground);
    hbmpTargetOld = SelectObject(hdcTarget, hbmpCheck);

    // Use the selected brush to initialize the background color
    // of the bitmap in the target device context.

    PatBlt(hdcTarget, 0, 0, wBitmapX, wBitmapY, PATCOPY);

    // Load the predefined check box bitmaps and select it
    // into the source DC.

    hbmpCheckboxes = LoadBitmap((HINSTANCE) NULL,
        (LPTSTR) OBM_CHECKBOXES);

    hbmpSourceOld = SelectObject(hdcSource, hbmpCheckboxes);

    // Fill a BITMAP structure with information about the
    // check box bitmaps, and then find the upper-left corner of
    // the unchecked check box or the checked check box.

    GetObject(hbmpCheckboxes, sizeof(BITMAP), &bmCheckbox);

    if (fuCheck == UNCHECK)
    {
        rc.left = 0;
        rc.right = (bmCheckbox.bmWidth / 4);
    }
    else
    {
        rc.left = (bmCheckbox.bmWidth / 4);
        rc.right = (bmCheckbox.bmWidth / 4) * 2;
    }

    rc.top = 0;
    rc.bottom = (bmCheckbox.bmHeight / 3);

    // Copy the appropriate bitmap into the target DC. If the
    // check-box bitmap is larger than the default check-mark
    // bitmap, use StretchBlt to make it fit; otherwise, just
    // copy it.

    if (((rc.right - rc.left) > (int) wBitmapX) ||
```

```
            ((rc.bottom - rc.top) > (int) wBitmapY))
    {
        StretchBlt(hdcTarget, 0, 0, wBitmapX, wBitmapY,
            hdcSource, rc.left, rc.top, rc.right - rc.left,
            rc.bottom - rc.top, SRCCOPY);
    }

    else
    {
        BitBlt(hdcTarget, 0, 0, rc.right - rc.left,
            rc.bottom - rc.top,
            hdcSource, rc.left, rc.top, SRCCOPY);
    }

    // Select the old source and destination bitmaps into the
    // source and destination DCs, and then delete the DCs and
    // the background brush.

    SelectObject(hdcSource, hbmpSourceOld);
    SelectObject(hdcTarget, hbrTargetOld);
    hbmpCheck = SelectObject(hdcTarget, hbmpTargetOld);

    DeleteObject(hbrBackground);
    DeleteObject(hdcSource);
    DeleteObject(hdcTarget);

    // Return a handle to the new check-mark bitmap.

    return hbmpCheck;
}


BYTE CheckOrUncheckMenuItem(BYTE bMenuItemID, HMENU hmenu)
{
    DWORD fdwMenu;
    static BYTE fbAttributes;

    switch (bMenuItemID)
    {
        case IDM_REGULAR:

            // Whenever the Regular menu item is selected, add a
            // check mark to it and then remove checkmarks from
            // any font-attribute menu items.

            CheckMenuItem(hmenu, IDM_REGULAR, MF_BYCOMMAND |
                MF_CHECKED);

            if (fbAttributes & ATTRIBMASK)
            {
                CheckMenuItem(hmenu, IDM_BOLD, MF_BYCOMMAND |
                    MF_UNCHECKED);
                CheckMenuItem(hmenu, IDM_ITALIC, MF_BYCOMMAND |
                    MF_UNCHECKED);
                CheckMenuItem(hmenu, IDM_ULINE, MF_BYCOMMAND |
```

```
                    MF_UNCHECKED);
            }
            fbAttributes = IDM_REGULAR;
            return fbAttributes;

        case IDM_BOLD:
        case IDM_ITALIC:
        case IDM_ULINE:

            // Toggle the check mark for the selected menu item and
            // set the font attribute flags appropriately.

            fdwMenu = GetMenuState(hmenu, (UINT) bMenuItemID,
                MF_BYCOMMAND);
            if (!(fdwMenu & MF_CHECKED))
            {
                CheckMenuItem(hmenu, (UINT) bMenuItemID,
                    MF_BYCOMMAND | MF_CHECKED);
                fbAttributes |= bMenuItemID;
            }
            else
            {
                CheckMenuItem(hmenu, (UINT) bMenuItemID,
                    MF_BYCOMMAND | MF_UNCHECKED);
                fbAttributes ^= bMenuItemID;
            }

            // If any font attributes are currently selected,
            // remove the check mark from the Regular menu item;
            // if no attributes are selected, add a check mark
            // to the Regular menu item.

            if (fbAttributes & ATTRIBMASK)
            {
                CheckMenuItem(hmenu, IDM_REGULAR,
                    MF_BYCOMMAND | MF_UNCHECKED);
                fbAttributes &= (BYTE) ~IDM_REGULAR;
            }
            else
            {
                CheckMenuItem(hmenu, IDM_REGULAR,
                    MF_BYCOMMAND | MF_CHECKED);
                fbAttributes = IDM_REGULAR;
            }

            return fbAttributes;
    }
}
```

Example of Using Custom Check-mark Bitmaps

The example in this topic assigns custom check-mark bitmaps to menu items in two menus. The menu items in the first menu specify character attributes: bold, italic, and underline. Each menu item can be either selected or cleared. For these menu items, the example uses check-mark bitmaps that resemble the selected and cleared states of a check box control.

The menu items in the second menu specify paragraph alignment settings: left, centered, and right. Only one of these menu items is selected at any time. For these menu items, the example uses check-mark bitmaps that resemble the selected and clear states of a radio button control.

The window procedure processes the **WM_CREATE** message by calling the application-defined OnCreate function. `OnCreate` creates the four check-mark bitmaps and then assigns them to their appropriate menu items by using the **SetMenuItemBitmaps** function.

To create each bitmap, OnCreate calls the application-defined CreateMenuBitmaps function, specifying a pointer to a bitmap-specific drawing function. CreateMenuBitmaps creates a monochrome bitmap of the required size, selects it into a memory device context, and erases the background. Then it calls the specified drawing function to fill in the foreground.

The four application-defined drawing functions are DrawCheck, DrawUncheck, **DrawRadioCheck**, and DrawRadioUncheck. They draw a rectangle with an X, an empty rectangle, an ellipse containing a smaller filled ellipse, and an empty ellipse, respectively.

The window procedure processes the **WM_DESTROY** message by deleting the check-mark bitmaps. It retrieves each bitmap handle by using the **GetMenuItemInfo** function and then passes a handle to the function.

When the user chooses a menu item, a **WM_COMMAND** message is sent to the owner window. For menu items on the **Character** menu, the window procedure calls the application-defined CheckCharacterItem function. For items on the **Paragraph** menu, the window procedure calls the application-defined CheckParagraphItem function.

Each item on the **Character** menu can be selected and cleared independently. Therefore, CheckCharacterItem simply switches the specified menu item's check state. First, the function calls the **GetMenuItemInfo** function to get the current menu item state. Then it switches the **MFS_CHECKED** state flag and sets the new state by calling the **SetMenuItemInfo** function.

Unlike character attributes, only one paragraph alignment can be selected at a time. Therefore, CheckParagraphItem checks the specified menu item and removes the check

mark from all other items on the menu. To do so, it calls the **CheckMenuRadioItem** function.

Following are the relevant portions of the application's header file.

C++

```
// Menu-item identifiers for the Character menu

#define IDM_CHARACTER 10
#define IDM_BOLD      11
#define IDM_ITALIC    12
#define IDM_UNDERLINE 13

// Menu-item identifiers for the Paragraph menu

#define IDM_PARAGRAPH 20
#define IDM_LEFT      21
#define IDM_CENTER    22
#define IDM_RIGHT     23

// Function-pointer type for drawing functions

typedef VOID (WINAPI * DRAWFUNC)(HDC hdc, SIZE size);
```

The following are the relevant portions of the application's window procedure and related functions.

C++

```
LRESULT CALLBACK MainWindowProc(
        HWND hwnd,
        UINT uMsg,
        WPARAM wParam,
        LPARAM lParam
        )
{
    switch (uMsg)
    {
        case WM_CREATE:
            if (!OnCreate(hwnd))
                return -1;
            break;

        case WM_DESTROY:
            OnDestroy(hwnd);
            PostQuitMessage(0);
            break;

        case WM_COMMAND:
            switch (LOWORD(wParam))
```

```
            {
                case IDM_BOLD:
                case IDM_ITALIC:
                case IDM_UNDERLINE:
                    CheckCharacterItem(hwnd, LOWORD(wParam));
                    break;

                case IDM_LEFT:
                case IDM_CENTER:
                case IDM_RIGHT:
                    CheckParagraphItem(hwnd, LOWORD(wParam));
                    break;

                // Process other commands here.

            }
            break;

        // Process other messages here.

        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

VOID WINAPI CheckCharacterItem(HWND hwnd, UINT uID)
{
    HMENU hmenuBar = GetMenu(hwnd);
    HMENU hmenuPopup;
    MENUITEMINFO mii;

    // Get a handle to the Character menu.

    mii.fMask = MIIM_SUBMENU;  // information to get
    GetMenuItemInfo(hmenuBar, IDM_CHARACTER, FALSE, &mii);
    hmenuPopup = mii.hSubMenu;

    // Get the state of the specified menu item.

    mii.fMask = MIIM_STATE;    // information to get
    GetMenuItemInfo(hmenuPopup, uID, FALSE, &mii);

    // Toggle the checked state.

    mii.fState ^= MFS_CHECKED;
    SetMenuItemInfo(hmenuPopup, uID, FALSE, &mii);
}

VOID WINAPI CheckParagraphItem(HWND hwnd, UINT uID)
{
    HMENU hmenuBar = GetMenu(hwnd);
    HMENU hmenuPopup;
    MENUITEMINFO mii;
```

```
    // Get a handle to the Paragraph menu.

    mii.fMask = MIIM_SUBMENU;  // information to get
    GetMenuItemInfo(hmenuBar, IDM_PARAGRAPH, FALSE, &mii);
    hmenuPopup = mii.hSubMenu;

    // Check the specified item and uncheck all the others.

    CheckMenuRadioItem(
            hmenuPopup,         // handle to menu
            IDM_LEFT,           // first item in range
            IDM_RIGHT,          // last item in range
            uID,                // item to check
            MF_BYCOMMAND        // IDs, not positions
            );
}

BOOL WINAPI OnCreate(HWND hwnd)
{
    HMENU hmenuBar = GetMenu(hwnd);
    HMENU hmenuPopup;
    MENUITEMINFO mii;
    UINT uID;
    HBITMAP hbmChecked;
    HBITMAP hbmUnchecked;

    // Get a handle to the Character menu.

    mii.fMask = MIIM_SUBMENU;     // information to get
    GetMenuItemInfo(hmenuBar, IDM_CHARACTER, FALSE, &mii);
    hmenuPopup = mii.hSubMenu;

    // Create the checked and unchecked bitmaps.

    hbmChecked = CreateMenuBitmap(DrawCheck);
    hbmUnchecked = CreateMenuBitmap(DrawUncheck);

    // Set the check-mark bitmaps for each menu item.

    for (uID = IDM_BOLD; uID <= IDM_UNDERLINE; uID++)
    {
        SetMenuItemBitmaps(hmenuPopup, uID, MF_BYCOMMAND,
                hbmUnchecked, hbmChecked);
    }

    // Get a handle to the Paragraph pop-up menu.

    mii.fMask = MIIM_SUBMENU;     // information to get
    GetMenuItemInfo(hmenuBar, IDM_PARAGRAPH, FALSE, &mii);
    hmenuPopup = mii.hSubMenu;

    // Create the checked and unchecked bitmaps.

    hbmChecked = CreateMenuBitmap(DrawRadioCheck);
    hbmUnchecked = CreateMenuBitmap(DrawRadioUncheck);
```

```
    // Set the check-mark bitmaps for each menu item.

    for (uID = IDM_LEFT; uID <= IDM_RIGHT; uID++)
    {
        SetMenuItemBitmaps(hmenuPopup, uID, MF_BYCOMMAND,
                hbmUnchecked, hbmChecked);
    }

    // Initially check the IDM_LEFT paragraph item.

    CheckMenuRadioItem(hmenuPopup, IDM_LEFT, IDM_RIGHT,
            IDM_LEFT, MF_BYCOMMAND);
    return TRUE;
}

HBITMAP WINAPI CreateMenuBitmap(DRAWFUNC lpfnDraw)
{
    // Create a DC compatible with the desktop window's DC.

    HWND hwndDesktop = GetDesktopWindow();
    HDC hdcDesktop = GetDC(hwndDesktop);
    HDC hdcMem = CreateCompatibleDC(hdcDesktop);

    // Determine the required bitmap size.

    SIZE size = { GetSystemMetrics(SM_CXMENUCHECK),
                  GetSystemMetrics(SM_CYMENUCHECK) };

    // Create a monochrome bitmap and select it.

    HBITMAP hbm = CreateBitmap(size.cx, size.cy, 1, 1, NULL);
    HBITMAP hbmOld = SelectObject(hdcMem, hbm);

    // Erase the background and call the drawing function.

    PatBlt(hdcMem, 0, 0, size.cx, size.cy, WHITENESS);
    (*lpfnDraw)(hdcMem, size);

    // Clean up.

    SelectObject(hdcMem, hbmOld);
    DeleteDC(hdcMem);
    ReleaseDC(hwndDesktop, hdcDesktop);
    return hbm;
}

VOID WINAPI DrawCheck(HDC hdc, SIZE size)
{
    HBRUSH hbrOld;
    hbrOld = SelectObject(hdc, GetStockObject(NULL_BRUSH));
    Rectangle(hdc, 0, 0, size.cx, size.cy);
    MoveToEx(hdc, 0, 0, NULL);
    LineTo(hdc, size.cx, size.cy);
    MoveToEx(hdc, 0, size.cy - 1, NULL);
```

```
    LineTo(hdc, size.cx - 1, 0);
    SelectObject(hdc, hbrOld);
}

VOID WINAPI DrawUncheck(HDC hdc, SIZE size)
{
    HBRUSH hbrOld;
    hbrOld = SelectObject(hdc, GetStockObject(NULL_BRUSH));
    Rectangle(hdc, 0, 0, size.cx, size.cy);
    SelectObject(hdc, hbrOld);
}

VOID WINAPI DrawRadioCheck(HDC hdc, SIZE size)
{
    HBRUSH hbrOld;
    hbrOld = SelectObject(hdc, GetStockObject(NULL_BRUSH));
    Ellipse(hdc, 0, 0, size.cx, size.cy);
    SelectObject(hdc, GetStockObject(BLACK_BRUSH));
    Ellipse(hdc, 2, 2, size.cx - 2, size.cy - 2);
    SelectObject(hdc, hbrOld);
}

VOID WINAPI DrawRadioUncheck(HDC hdc, SIZE size)
{
    HBRUSH hbrOld;
    hbrOld = SelectObject(hdc, GetStockObject(NULL_BRUSH));
    Ellipse(hdc, 0, 0, size.cx, size.cy);
    SelectObject(hdc, hbrOld);
}

VOID WINAPI OnDestroy(HWND hwnd)
{
    HMENU hmenuBar = GetMenu(hwnd);
    HMENU hmenuPopup;
    MENUITEMINFO mii;

    // Get a handle to the Character menu.

    mii.fMask = MIIM_SUBMENU;     // information to get
    GetMenuItemInfo(hmenuBar, IDM_CHARACTER, FALSE, &mii);
    hmenuPopup = mii.hSubMenu;

    // Get the check-mark bitmaps and delete them.

    mii.fMask = MIIM_CHECKMARKS;
    GetMenuItemInfo(hmenuPopup, IDM_BOLD, FALSE, &mii);
    DeleteObject(mii.hbmpChecked);
    DeleteObject(mii.hbmpUnchecked);

    // Get a handle to the Paragraph menu.

    mii.fMask = MIIM_SUBMENU;     // information to get
    GetMenuItemInfo(hmenuBar, IDM_PARAGRAPH, FALSE, &mii);
    hmenuPopup = mii.hSubMenu;
```

```
    // Get the check-mark bitmaps and delete them.

    mii.fMask = MIIM_CHECKMARKS;
    GetMenuItemInfo(hmenuPopup, IDM_LEFT, FALSE, &mii);
    DeleteObject(mii.hbmpChecked);
    DeleteObject(mii.hbmpUnchecked);
}
```

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Menu Reference

Article • 04/27/2021

In This Section

- Menu Functions
- Menu Notifications
- Menu Structures

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Menu Functions

Article • 04/27/2021

In This Section

- AppendMenu
- CheckMenuItem
- CheckMenuRadioItem
- CreateMenu
- CreatePopupMenu
- DeleteMenu
- DestroyMenu
- DrawMenuBar
- EnableMenuItem
- EndMenu
- GetMenu
- GetMenuBarInfo
- GetMenuCheckMarkDimensions
- GetMenuDefaultItem
- GetMenuInfo
- GetMenuItemCount
- GetMenuItemID
- GetMenuItemInfo
- GetMenuItemRect
- GetMenuState
- GetMenuString
- GetSubMenu
- GetSystemMenu
- HiliteMenuItem
- InsertMenu
- InsertMenuItem
- IsMenu
- LoadMenu
- LoadMenuIndirect
- MenuItemFromPoint
- ModifyMenu
- RemoveMenu
- SetMenu
- SetMenuDefaultItem

- SetMenuInfo
- SetMenuItemBitmaps
- SetMenuItemInfo
- TrackPopupMenu
- TrackPopupMenuEx

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

AppendMenuA function (winuser.h)

Article02/09/2023

Appends a new item to the end of the specified menu bar, drop-down menu, submenu, or shortcut menu. You can use this function to specify the content, appearance, and behavior of the menu item.

Syntax

C++

```
BOOL AppendMenuA(
  [in]           HMENU    hMenu,
  [in]           UINT     uFlags,
  [in]           UINT_PTR uIDNewItem,
  [in, optional] LPCSTR   lpNewItem
);
```

Parameters

`[in] hMenu`

Type: **HMENU**

A handle to the menu bar, drop-down menu, submenu, or shortcut menu to be changed.

`[in] uFlags`

Type: **UINT**

Controls the appearance and behavior of the new menu item. This parameter can be a combination of the following values.

Expand table

Value	Meaning
MF_BITMAP 0x00000004L	Uses a bitmap as the menu item. The *lpNewItem* parameter contains a handle to the bitmap.
MF_CHECKED 0x00000008L	Places a check mark next to the menu item. If the application provides check-mark bitmaps (see

	SetMenuItemBitmaps, this flag displays the check-mark bitmap next to the menu item.
MF_DISABLED 0x00000002L	Disables the menu item so that it cannot be selected, but the flag does not gray it.
MF_ENABLED 0x00000000L	Enables the menu item so that it can be selected, and restores it from its grayed state.
MF_GRAYED 0x00000001L	Disables the menu item and grays it so that it cannot be selected.
MF_MENUBARBREAK 0x00000020L	Functions the same as the **MF_MENUBREAK** flag for a menu bar. For a drop-down menu, submenu, or shortcut menu, the new column is separated from the old column by a vertical line.
MF_MENUBREAK 0x00000040L	Places the item on a new line (for a menu bar) or in a new column (for a drop-down menu, submenu, or shortcut menu) without separating columns.
MF_OWNERDRAW 0x00000100L	Specifies that the item is an owner-drawn item. Before the menu is displayed for the first time, the window that owns the menu receives a WM_MEASUREITEM message to retrieve the width and height of the menu item. The WM_DRAWITEM message is then sent to the window procedure of the owner window whenever the appearance of the menu item must be updated.
MF_POPUP 0x00000010L	Specifies that the menu item opens a drop-down menu or submenu. The *uIDNewItem* parameter specifies a handle to the drop-down menu or submenu. This flag is used to add a menu name to a menu bar, or a menu item that opens a submenu to a drop-down menu, submenu, or shortcut menu.
MF_SEPARATOR 0x00000800L	Draws a horizontal dividing line. This flag is used only in a drop-down menu, submenu, or shortcut menu. The line cannot be grayed, disabled, or highlighted. The *lpNewItem* and *uIDNewItem* parameters are ignored.
MF_STRING 0x00000000L	Specifies that the menu item is a text string; the *lpNewItem* parameter is a pointer to the string.
MF_UNCHECKED 0x00000000L	Does not place a check mark next to the item (default). If the application supplies check-mark bitmaps (see SetMenuItemBitmaps), this flag displays the clear bitmap next to the menu item.

`[in] uIDNewItem`

Type: **UINT_PTR**

The identifier of the new menu item or, if the *uFlags* parameter is set to **MF_POPUP**, a handle to the drop-down menu or submenu.

`[in, optional] lpNewItem`

Type: **LPCTSTR**

The content of the new menu item. The interpretation of *lpNewItem* depends on whether the *uFlags* parameter includes the following values.

Expand table

Value	Meaning
MF_BITMAP 0x00000004L	Contains a bitmap handle.
MF_OWNERDRAW 0x00000100L	Contains an application-supplied value that can be used to maintain additional data related to the menu item. The value is in the **itemData** member of the structure pointed to by the *lParam* parameter of the WM_MEASUREITEM or WM_DRAWITEM message sent when the menu is created or its appearance is updated.
MF_STRING 0x00000000L	Contains a pointer to a null-terminated string.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero. If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The application must call the DrawMenuBar function whenever a menu changes, whether the menu is in a displayed window.

To get keyboard accelerators to work with bitmap or owner-drawn menu items, the owner of the menu must process the WM_MENUCHAR message. For more information, see Owner-Drawn Menus and the WM_MENUCHAR Message.

The following groups of flags cannot be used together:

- **MF_BITMAP**, **MF_STRING**, and **MF_OWNERDRAW**
- **MF_CHECKED** and **MF_UNCHECKED**
- **MF_DISABLED**, **MF_ENABLED**, and **MF_GRAYED**
- **MF_MENUBARBREAK** and **MF_MENUBREAK**

Examples

For an example, see Adding Lines and Graphs to a Menu.

> **Note**
>
> The winuser.h header defines AppendMenu as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-0 (introduced in Windows 8)

See also

Conceptual

CreateMenu

DeleteMenu

DestroyMenu

DrawMenuBar

InsertMenu

InsertMenuItem

Menus

ModifyMenu

Reference

RemoveMenu

SetMenuItemBitmaps

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CheckMenuItem function (winuser.h)

Article02/22/2024

[**CheckMenuItem** is available for use in the operating systems specified in the Requirements section. It may be altered or unavailable in subsequent versions. Instead, use SetMenuItemInfo.]

Sets the state of the specified menu item's check-mark attribute to either selected or clear.

Syntax

```
C++

DWORD CheckMenuItem(
  [in] HMENU hMenu,
  [in] UINT  uIDCheckItem,
  [in] UINT  uCheck
);
```

Parameters

`[in] hMenu`

Type: **HMENU**

A handle to the menu of interest.

`[in] uIDCheckItem`

Type: **UINT**

The menu item whose check-mark attribute is to be set, as determined by the *uCheck* parameter.

`[in] uCheck`

Type: **UINT**

The flags that control the interpretation of the *uIDCheckItem* parameter and the state of the menu item's check-mark attribute. This parameter can be a combination of either **MF_BYCOMMAND**, or **MF_BYPOSITION** and **MF_CHECKED** or **MF_UNCHECKED**.

Expand table

Value	Meaning
MF_BYCOMMAND 0x00000000L	Indicates that the *uIDCheckItem* parameter gives the identifier of the menu item. The **MF_BYCOMMAND** flag is the default, if neither the **MF_BYCOMMAND** nor **MF_BYPOSITION** flag is specified.
MF_BYPOSITION 0x00000400L	Indicates that the *uIDCheckItem* parameter gives the zero-based relative position of the menu item.
MF_CHECKED 0x00000008L	Sets the check-mark attribute to the selected state.
MF_UNCHECKED 0x00000000L	Sets the check-mark attribute to the clear state.

Return value

Type: **DWORD**

The return value specifies the previous state of the menu item (either **MF_CHECKED** or **MF_UNCHECKED**). If the menu item does not exist, the return value is –1.

Remarks

An item in a menu bar cannot have a check mark.

The *uIDCheckItem* parameter identifies a item that opens a submenu or a command item. For a item that opens a submenu, the *uIDCheckItem* parameter must specify the position of the item. For a command item, the *uIDCheckItem* parameter can specify either the item's position or its identifier.

Examples

For an example, see Simulating Check Boxes in a Menu.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-0 (introduced in Windows 8)

See also

Conceptual

EnableMenuItem

GetMenuItemID

Menus

Reference

SetMenuItemBitmaps

SetMenuItemInfo

Feedback

CheckMenuRadioItem function (winuser.h)

Article02/22/2024

Checks a specified menu item and makes it a radio item. At the same time, the function clears all other menu items in the associated group and clears the radio-item type flag for those items.

Syntax

```
C++

BOOL CheckMenuRadioItem(
  [in] HMENU hmenu,
  [in] UINT  first,
  [in] UINT  last,
  [in] UINT  check,
  [in] UINT  flags
);
```

Parameters

`[in] hmenu`

Type: **HMENU**

A handle to the menu that contains the group of menu items.

`[in] first`

Type: **UINT**

The identifier or position of the first menu item in the group.

`[in] last`

Type: **UINT**

The identifier or position of the last menu item in the group.

`[in] check`

Type: **UINT**

The identifier or position of the menu item to check.

`[in] flags`

Type: **UINT**

Indicates the meaning of *idFirst*, *idLast*, and *idCheck*. If this parameter is **MF_BYCOMMAND**, the other parameters specify menu item identifiers. If it is **MF_BYPOSITION**, the other parameters specify the menu item positions.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, use the GetLastError function.

Remarks

The **CheckMenuRadioItem** function sets the **MFT_RADIOCHECK** type flag and the **MFS_CHECKED** state for the item specified by *idCheck* and, at the same time, clears both flags for all other items in the group. The selected item is displayed using a bullet bitmap instead of a check-mark bitmap.

For more information about menu item type and state flags, see the MENUITEMINFO structure.

Examples

For an example, see Example of Example of Using Custom Checkmark Bitmaps.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-0 (introduced in Windows 8)

See also

Conceptual

MENUITEMINFO

Menus

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CreateMenu function (winuser.h)

Article02/22/2024

Creates a menu. The menu is initially empty, but it can be filled with menu items by using the InsertMenuItem, AppendMenu, and InsertMenu functions.

Syntax

```
C++

HMENU CreateMenu();
```

Return value

Type: **HMENU**

If the function succeeds, the return value is a handle to the newly created menu.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

Resources associated with a menu that is assigned to a window are freed automatically. If the menu is not assigned to a window, an application must free system resources associated with the menu before closing. An application frees menu resources by calling the DestroyMenu function.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-2 (introduced in Windows 10, version 10.0.10240)

See also

AppendMenu

Conceptual

CreatePopupMenu

DestroyMenu

InsertMenu

InsertMenuItem

Menus

Reference

SetMenu

Feedback

CreatePopupMenu function (winuser.h)

Article02/22/2024

Creates a drop-down menu, submenu, or shortcut menu. The menu is initially empty. You can insert or append menu items by using the InsertMenuItem function. You can also use the InsertMenu function to insert menu items and the AppendMenu function to append menu items.

Syntax

```
C++

HMENU CreatePopupMenu();
```

Return value

Type: **HMENU**

If the function succeeds, the return value is a handle to the newly created menu.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

The application can add the new menu to an existing menu, or it can display a shortcut menu by calling the TrackPopupMenuEx or TrackPopupMenu functions.

Resources associated with a menu that is assigned to a window are freed automatically. If the menu is not assigned to a window, an application must free system resources associated with the menu before closing. An application frees menu resources by calling the DestroyMenu function.

Examples

For an example, see Adding Lines and Graphs to a Menu.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-0 (introduced in Windows 8)

See also

AppendMenu

Conceptual

CreateMenu

DestroyMenu

InsertMenu

InsertMenuItem

Menus

Reference

SetMenu

TrackPopupMenu

TrackPopupMenuEx

Feedback

Was this page helpful? Yes No

DeleteMenu function (winuser.h)

Article02/22/2024

Deletes an item from the specified menu. If the menu item opens a menu or submenu, this function destroys the handle to the menu or submenu and frees the memory used by the menu or submenu.

Syntax

C++

```
BOOL DeleteMenu(
  [in] HMENU hMenu,
  [in] UINT  uPosition,
  [in] UINT  uFlags
);
```

Parameters

`[in] hMenu`

Type: **HMENU**

A handle to the menu to be changed.

`[in] uPosition`

Type: **UINT**

The menu item to be deleted, as determined by the *uFlags* parameter.

`[in] uFlags`

Type: **UINT**

Indicates how the *uPosition* parameter is interpreted. This parameter must be one of the following values.

Expand table

Value	Meaning

MF_BYCOMMAND 0x00000000L	Indicates that *uPosition* gives the identifier of the menu item. The **MF_BYCOMMAND** flag is the default flag if neither the **MF_BYCOMMAND** nor **MF_BYPOSITION** flag is specified.
MF_BYPOSITION 0x00000400L	Indicates that *uPosition* gives the zero-based relative position of the menu item.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The application must call the DrawMenuBar function whenever a menu changes, whether the menu is in a displayed window.

Examples

For an example, see Example of a Clipboard Viewer.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

Requirement	Value
API set	ext-ms-win-ntuser-menu-l1-1-0 (introduced in Windows 8)

See also

Conceptual

DrawMenuBar

Menus

Reference

RemoveMenu

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

DestroyMenu function (winuser.h)

Article02/22/2024

Destroys the specified menu and frees any memory that the menu occupies.

Syntax

```
C++

BOOL DestroyMenu(
  [in] HMENU hMenu
);
```

Parameters

`[in] hMenu`

Type: **HMENU**

A handle to the menu to be destroyed.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

Before closing, an application must use the **DestroyMenu** function to destroy a menu not assigned to a window. A menu that is assigned to a window is automatically destroyed when the application closes.

DestroyMenu is recursive, that is, it will destroy the menu and all its submenus.

Examples

For an example, see Displaying a Shortcut Menu.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-0 (introduced in Windows 8)

See also

Conceptual

CreateMenu

DeleteMenu

Menus

Reference

RemoveMenu

SetMenuItemInfo

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

DrawMenuBar function (winuser.h)

Article02/22/2024

Redraws the menu bar of the specified window. If the menu bar changes after the system has created the window, this function must be called to draw the changed menu bar.

Syntax

```
C++

BOOL DrawMenuBar(
  [in] HWND hWnd
);
```

Parameters

`[in] hWnd`

Type: **HWND**

A handle to the window whose menu bar is to be redrawn.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

DeleteMenu

InsertMenuItem

Menus

Reference

RemoveMenu

SetMenuItemInfo

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

EnableMenuItem function (winuser.h)

Article10/13/2021

Enables, disables, or grays the specified menu item.

Syntax

```
C++

BOOL EnableMenuItem(
  [in] HMENU hMenu,
  [in] UINT  uIDEnableItem,
  [in] UINT  uEnable
);
```

Parameters

`[in] hMenu`

Type: **HMENU**

A handle to the menu.

`[in] uIDEnableItem`

Type: **UINT**

The menu item to be enabled, disabled, or grayed, as determined by the *uEnable* parameter. This parameter specifies an item in a menu bar, menu, or submenu.

`[in] uEnable`

Type: **UINT**

Controls the interpretation of the *uIDEnableItem* parameter and indicate whether the menu item is enabled, disabled, or grayed. This parameter must be a combination of the following values.

Expand table

Value	Meaning

MF_BYCOMMAND 0x00000000L	Indicates that *uIDEnableItem* gives the identifier of the menu item. If neither the **MF_BYCOMMAND** nor **MF_BYPOSITION** flag is specified, the **MF_BYCOMMAND** flag is the default flag.
MF_BYPOSITION 0x00000400L	Indicates that *uIDEnableItem* gives the zero-based relative position of the menu item.
MF_DISABLED 0x00000002L	Indicates that the menu item is disabled, but not grayed, so it cannot be selected.
MF_ENABLED 0x00000000L	Indicates that the menu item is enabled and restored from a grayed state so that it can be selected.
MF_GRAYED 0x00000001L	Indicates that the menu item is disabled and grayed so that it cannot be selected.

Return value

Type: **BOOL**

The return value specifies the previous state of the menu item (it is either **MF_DISABLED**, **MF_ENABLED**, or **MF_GRAYED**). If the menu item does not exist, the return value is -1.

Remarks

An application must use the **MF_BYPOSITION** flag to specify the correct menu handle. If the menu handle to the menu bar is specified, the top-level menu item (an item in the menu bar) is affected. To set the state of an item in a drop-down menu or submenu by position, an application must specify a handle to the drop-down menu or submenu.

When an application specifies the **MF_BYCOMMAND** flag, the system checks all items that open submenus in the menu identified by the specified menu handle. Therefore, unless duplicate menu items are present, specifying the menu handle to the menu bar is sufficient.

The InsertMenu, InsertMenuItem, LoadMenuIndirect, ModifyMenu, and SetMenuItemInfo functions can also set the state (enabled, disabled, or grayed) of a menu item.

When you change a window menu, the menu bar is not immediately updated. To force the update, call DrawMenuBar.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-0 (introduced in Windows 8)

See also

Conceptual

DrawMenuBar

GetMenuItemID

InsertMenu

InsertMenuItem

LoadMenuIndirect

Menus

ModifyMenu

Reference

SetMenuItemInfo

WM_SYSCOMMAND

Feedback

EndMenu function (winuser.h)

Article02/22/2024

Ends the calling thread's active menu.

Syntax

```
C++

BOOL EndMenu();
```

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

If a platform does not support **EndMenu**, send the owner of the active menu a WM_CANCELMODE message.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib

Requirement	Value
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-0 (introduced in Windows 8)

See also

Conceptual

Menus

Reference

WM_CANCELMODE

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetMenu function (winuser.h)

Article02/22/2024

Retrieves a handle to the menu assigned to the specified window.

Syntax

```
C++

HMENU GetMenu(
  [in] HWND hWnd
);
```

Parameters

`[in] hWnd`

Type: **HWND**

A handle to the window whose menu handle is to be retrieved.

Return value

Type: **HMENU**

The return value is a handle to the menu. If the specified window has no menu, the return value is **NULL**. If the window is a child window, the return value is undefined.

Remarks

GetMenu does not work on floating menu bars. Floating menu bars are custom controls that mimic standard menus; they are not menus. To get the handle on a floating menu bar, use the Active Accessibility APIs.

Examples

For an example, see Adding Lines and Graphs to a Menu.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetSubMenu

Menus

Reference

SetMenu

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetMenuBarInfo function (winuser.h)

Article10/13/2021

Retrieves information about the specified menu bar.

Syntax

```
C++

BOOL GetMenuBarInfo(
  [in]      HWND          hwnd,
  [in]      LONG          idObject,
  [in]      LONG          idItem,
  [in, out] PMENUBARINFO pmbi
);
```

Parameters

`[in] hwnd`

Type: **HWND**

A handle to the window (menu bar) whose information is to be retrieved.

`[in] idObject`

Type: **LONG**

The menu object. This parameter can be one of the following values.

Expand table

Value	Meaning
OBJID_CLIENT ((LONG)0xFFFFFFFC)	The popup menu associated with the window.
OBJID_MENU ((LONG)0xFFFFFFFD)	The menu bar associated with the window (see the GetMenu function).
OBJID_SYSMENU ((LONG)0xFFFFFFFF)	The system menu associated with the window (see the GetSystemMenu function).

`[in] idItem`

Type: **LONG**

The item for which to retrieve information. If this parameter is zero, the function retrieves information about the menu itself. If this parameter is 1, the function retrieves information about the first item on the menu, and so on.

`[in, out] pmbi`

Type: **PMENUBARINFO**

A pointer to a MENUBARINFO structure that receives the information. Note that you must set the **cbSize** member to `sizeof(MENUBARINFO)` before calling this function.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-misc-l1-5-0 (introduced in Windows 10, version 10.0.10240)

See also

Conceptual

GetMenu

GetSystemMenu

MENUBARINFO

Menus

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetMenuCheckMarkDimensions function (winuser.h)

Article02/22/2024

Retrieves the dimensions of the default check-mark bitmap. The system displays this bitmap next to selected menu items. Before calling the SetMenuItemBitmaps function to replace the default check-mark bitmap for a menu item, an application must determine the correct bitmap size by calling **GetMenuCheckMarkDimensions**.

> **Note** The **GetMenuCheckMarkDimensions** function is included only for compatibility with 16-bit versions of Windows. Applications should use the **GetSystemMetrics** function with the **CXMENUCHECK** and **CYMENUCHECK** values to retrieve the bitmap dimensions.

Syntax

```
C++

LONG GetMenuCheckMarkDimensions();
```

Return value

Type: **LONG**

The return value specifies the height and width, in pixels, of the default check-mark bitmap. The high-order word contains the height; the low-order word contains the width.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

Menus

Reference

SetMenuItemBitmaps

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetMenuDefaultItem function (winuser.h)

Article10/13/2021

Determines the default menu item on the specified menu.

Syntax

```
C++

UINT GetMenuDefaultItem(
  [in] HMENU hMenu,
  [in] UINT  fByPos,
  [in] UINT  gmdiFlags
);
```

Parameters

`[in] hMenu`

Type: **HMENU**

A handle to the menu for which to retrieve the default menu item.

`[in] fByPos`

Type: **UINT**

Indicates whether to retrieve the menu item's identifier or its position. If this parameter is **FALSE**, the identifier is returned. Otherwise, the position is returned.

`[in] gmdiFlags`

Type: **UINT**

Indicates how the function should search for menu items. This parameter can be zero or more of the following values.

Expand table

Value	Meaning

GMDI_GOINTOPOPUPS 0x0002L	If the default item is one that opens a submenu, the function is to search recursively in the corresponding submenu. If the submenu has no default item, the return value identifies the item that opens the submenu. By default, the function returns the first default item on the specified menu, regardless of whether it is an item that opens a submenu.
GMDI_USEDISABLED 0x0001L	The function is to return a default item, even if it is disabled. By default, the function skips disabled or grayed items.

Return value

Type: **UINT**

If the function succeeds, the return value is the identifier or position of the menu item.

If the function fails, the return value is -1. To get extended error information, call GetLastError.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-0 (introduced in Windows 8)

See also

Conceptual

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetMenuInfo function (winuser.h)

Article02/22/2024

Retrieves information about a specified menu.

Syntax

```
C++
```

```
BOOL GetMenuInfo(
  [in]      HMENU      unnamedParam1,
  [in, out] LPMENUINFO unnamedParam2
);
```

Parameters

`[in] unnamedParam1`

Type: **HMENU**

A handle on a menu.

`[in, out] unnamedParam2`

Type: **LPMENUINFO**

A pointer to a MENUINFO structure containing information for the menu. Note that you must set the **cbSize** member to `sizeof(MENUINFO)` before calling this function.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-3 (introduced in Windows 10, version 10.0.14393)

See also

Menus

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetMenuItemCount function (winuser.h)

Article02/22/2024

Determines the number of items in the specified menu.

Syntax

```
C++
```

```
int GetMenuItemCount(
  [in, optional] HMENU hMenu
);
```

Parameters

`[in, optional] hMenu`

Type: **HMENU**

A handle to the menu to be examined.

Return value

Type: **int**

If the function succeeds, the return value specifies the number of items in the menu.

If the function fails, the return value is -1. To get extended error information, call GetLastError.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-0 (introduced in Windows 8)

See also

Conceptual

GetMenuItemID

Menus

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetMenuItemID function (winuser.h)

Article02/22/2024

Retrieves the menu item identifier of a menu item located at the specified position in a menu.

Syntax

```
C++

UINT GetMenuItemID(
  [in] HMENU hMenu,
  [in] int   nPos
);
```

Parameters

`[in] hMenu`

Type: **HMENU**

A handle to the menu that contains the item whose identifier is to be retrieved.

`[in] nPos`

Type: **int**

The zero-based relative position of the menu item whose identifier is to be retrieved.

Return value

Type: **UINT**

The return value is the identifier of the specified menu item. If the menu item identifier is **NULL** or if the specified item opens a submenu, the return value is -1.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-0 (introduced in Windows 8)

See also

Conceptual

GetMenuItemCount

Menus

Reference

Feedback

GetMenuItemInfoA function (winuser.h)

Article02/22/2024

Retrieves information about a menu item.

Syntax

```
C++
```

```
BOOL GetMenuItemInfoA(
  [in]      HMENU           hmenu,
  [in]      UINT            item,
  [in]      BOOL            fByPosition,
  [in, out] LPMENUITEMINFOA lpmii
);
```

Parameters

`[in] hmenu`

Type: **HMENU**

A handle to the menu that contains the menu item.

`[in] item`

Type: **UINT**

The identifier or position of the menu item to get information about. The meaning of this parameter depends on the value of *fByPosition*.

`[in] fByPosition`

Type: **BOOL**

The meaning of *uItem*. If this parameter is **FALSE**, *uItem* is a menu item identifier. Otherwise, it is a menu item position. See Accessing Menu Items Programmatically for more information.

`[in, out] lpmii`

Type: **LPMENUITEMINFO**

A pointer to a MENUITEMINFO structure that specifies the information to retrieve and receives information about the menu item. Note that you must set the **cbSize** member to `sizeof(MENUITEMINFO)` before calling this function.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, use the GetLastError function.

Remarks

To retrieve a menu item of type **MFT_STRING**, first find the size of the string by setting the **dwTypeData** member of MENUITEMINFO to **NULL** and then calling **GetMenuItemInfo**. The value of **cch**+1 is the size needed. Then allocate a buffer of this size, place the pointer to the buffer in **dwTypeData**, increment **cch** by one, and then call **GetMenuItemInfo** once again to fill the buffer with the string.

If the retrieved menu item is of some other type, then **GetMenuItemInfo** sets the **dwTypeData** member to a value whose type is specified by the **fTypefType** member and sets **cch** to 0.

Examples

For an example, see Example of Owner-Drawn Menu Items.

> ⓘ **Note**
>
> The winuser.h header defines GetMenuItemInfo as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-3 (introduced in Windows 10, version 10.0.14393)

See also

Conceptual

Menus

Reference

SetMenuItemInfo

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetMenuItemRect function (winuser.h)

Article02/22/2024

Retrieves the bounding rectangle for the specified menu item.

Syntax

C++

```
BOOL GetMenuItemRect(
  [in, optional] HWND   hWnd,
  [in]           HMENU  hMenu,
  [in]           UINT   uItem,
  [out]          LPRECT lprcItem
);
```

Parameters

`[in, optional] hWnd`

Type: **HWND**

A handle to the window containing the menu.

If this value is **NULL** and the *hMenu* parameter represents a popup menu, the function will find the menu window.

`[in] hMenu`

Type: **HMENU**

A handle to a menu.

`[in] uItem`

Type: **UINT**

The zero-based position of the menu item.

`[out] lprcItem`

Type: **LPRECT**

A pointer to a RECT structure that receives the bounding rectangle of the specified menu item expressed in screen coordinates.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, use the GetLastError function.

Remarks

In order for the returned rectangle to be meaningful, the menu must be popped up if a popup menu or attached to a window if a menu bar. Menu item positions are not determined until the menu is displayed.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Menus

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetMenuState function (winuser.h)

Article10/13/2021

Retrieves the menu flags associated with the specified menu item. If the menu item opens a submenu, this function also returns the number of items in the submenu.

> **Note** The **GetMenuState** function has been superseded by the **GetMenuItemInfo**. You can still use **GetMenuState**, however, if you do not need any of the extended features of **GetMenuItemInfo**.

Syntax

```
C++

UINT GetMenuState(
  [in] HMENU hMenu,
  [in] UINT  uId,
  [in] UINT  uFlags
);
```

Parameters

`[in] hMenu`

Type: **HMENU**

A handle to the menu that contains the menu item whose flags are to be retrieved.

`[in] uId`

Type: **UINT**

The menu item for which the menu flags are to be retrieved, as determined by the *uFlags* parameter.

`[in] uFlags`

Type: **UINT**

Indicates how the *uId* parameter is interpreted. This parameter can be one of the following values.

Expand table

Value	Meaning
MF_BYCOMMAND 0x00000000L	Indicates that the *uId* parameter gives the identifier of the menu item. The **MF_BYCOMMAND** flag is the default if neither the **MF_BYCOMMAND** nor **MF_BYPOSITION** flag is specified.
MF_BYPOSITION 0x00000400L	Indicates that the *uId* parameter gives the zero-based relative position of the menu item.

Return value

Type: **UINT**

If the specified item does not exist, the return value is -1.

If the menu item opens a submenu, the low-order byte of the return value contains the menu flags associated with the item, and the high-order byte contains the number of items in the submenu opened by the item.

Otherwise, the return value is a mask (Bitwise OR) of the menu flags. Following are the menu flags associated with the menu item.

Expand table

Return code/value	Description
MF_CHECKED 0x00000008L	A check mark is placed next to the item (for drop-down menus, submenus, and shortcut menus only).
MF_DISABLED 0x00000002L	The item is disabled.
MF_GRAYED 0x00000001L	The item is disabled and grayed.
MF_HILITE 0x00000080L	The item is highlighted.
MF_MENUBARBREAK 0x00000020L	This is the same as the **MF_MENUBREAK** flag, except for drop-down menus, submenus, and shortcut menus,

	where the new column is separated from the old column by a vertical line.
MF_MENUBREAK 0x00000040L	The item is placed on a new line (for menu bars) or in a new column (for drop-down menus, submenus, and shortcut menus) without separating columns.
MF_OWNERDRAW 0x00000100L	The item is owner-drawn.
MF_POPUP 0x00000010L	Menu item is a submenu.
MF_SEPARATOR 0x00000800L	There is a horizontal dividing line (for drop-down menus, submenus, and shortcut menus only).

Remarks

It is possible to test an item for a flag value of **MF_ENABLED**, **MF_STRING**, **MF_UNCHECKED**, or **MF_UNHILITE**. However, since these values equate to zero you must use an expression to test for them.

Expand table

Flag	Expression to test for the flag
MF_ENABLED	`! (Flag&(MF_DISABLED | MF_GRAYED))`
MF_STRING	`! (Flag&(MF_BITMAP | MF_OWNERDRAW))`
MF_UNCHECKED	`! (Flag&MF_CHECKED)`
MF_UNHILITE	`! (Flag&HILITE)`

Examples

For an example, see Simulating Check Boxes in a Menu.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-2 (introduced in Windows 10, version 10.0.10240)

See also

Conceptual

GetMenu

GetMenuItemCount

GetMenuItemID

GetMenuItemInfo

Menus

Reference

Feedback

GetMenuStringA function (winuser.h)

Article02/09/2023

Copies the text string of the specified menu item into the specified buffer.

> **Note** The **GetMenuString** function has been superseded. Use the **GetMenuItemInfo** function to retrieve the menu item text.

Syntax

```
C++

int GetMenuStringA(
  [in]            HMENU hMenu,
  [in]            UINT  uIDItem,
  [out, optional] LPSTR lpString,
  [in]            int   cchMax,
  [in]            UINT  flags
);
```

Parameters

`[in] hMenu`

Type: **HMENU**

A handle to the menu.

`[in] uIDItem`

Type: **UINT**

The menu item to be changed, as determined by the *uFlag* parameter.

`[out, optional] lpString`

Type: **LPTSTR**

The buffer that receives the null-terminated string. If the string is as long or longer than *lpString*, the string is truncated and the terminating null character is added. If *lpString* is **NULL**, the function returns the length of the menu string.

`[in] cchMax`

Type: **int**

The maximum length, in characters, of the string to be copied. If the string is longer than the maximum specified in the *nMaxCount* parameter, the extra characters are truncated. If *nMaxCount* is 0, the function returns the length of the menu string.

`[in] flags`

Type: **UINT**

Indicates how the *uIDItem* parameter is interpreted. This parameter must be one of the following values.

Expand table

Value	Meaning
MF_BYCOMMAND 0x00000000L	Indicates that *uIDItem* gives the identifier of the menu item. If neither the **MF_BYCOMMAND** nor **MF_BYPOSITION** flag is specified, the **MF_BYCOMMAND** flag is the default flag.
MF_BYPOSITION 0x00000400L	Indicates that *uIDItem* gives the zero-based relative position of the menu item.

Return value

Type: **int**

If the function succeeds, the return value specifies the number of characters copied to the buffer, not including the terminating null character.

If the function fails, the return value is zero.

If the specified item is not of type **MIIM_STRING** or **MFT_STRING**, then the return value is zero.

Remarks

The *nMaxCount* parameter must be one larger than the number of characters in the text string to accommodate the terminating null character.

If *nMaxCount* is 0, the function returns the length of the menu string.

Security Warning

The *lpString* parameter is a **TCHAR** buffer, and *nMaxCount* is the length of the menu string in characters. Sizing these parameters incorrectly can cause truncation of the string, leading to possible loss of data.

Examples

For an example, see Creating User Editable Accelerators.

> **Note**
>
> The winuser.h header defines GetMenuString as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-3 (introduced in Windows 10, version 10.0.14393)

See also

Conceptual

GetMenuItemID

Menus

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetSubMenu function (winuser.h)

Article02/22/2024

Retrieves a handle to the drop-down menu or submenu activated by the specified menu item.

Syntax

```
C++

HMENU GetSubMenu(
  [in] HMENU hMenu,
  [in] int   nPos
);
```

Parameters

`[in] hMenu`

Type: **HMENU**

A handle to the menu.

`[in] nPos`

Type: **int**

The zero-based relative position in the specified menu of an item that activates a drop-down menu or submenu.

Return value

Type: **HMENU**

If the function succeeds, the return value is a handle to the drop-down menu or submenu activated by the menu item. If the menu item does not activate a drop-down menu or submenu, the return value is **NULL**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-0 (introduced in Windows 8)

See also

Conceptual

CreatePopupMenu

GetMenu

Menus

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetSystemMenu function (winuser.h)

Article02/22/2024

Enables the application to access the window menu (also known as the system menu or the control menu) for copying and modifying.

Syntax

```
C++

HMENU GetSystemMenu(
  [in] HWND hWnd,
  [in] BOOL bRevert
);
```

Parameters

`[in] hWnd`

Type: **HWND**

A handle to the window that will own a copy of the window menu.

`[in] bRevert`

Type: **BOOL**

The action to be taken. If this parameter is **FALSE**, **GetSystemMenu** returns a handle to the copy of the window menu currently in use. The copy is initially identical to the window menu, but it can be modified. If this parameter is **TRUE**, **GetSystemMenu** resets the window menu back to the default state. The previous window menu, if any, is destroyed.

Return value

Type: **HMENU**

If the *bRevert* parameter is **FALSE**, the return value is a handle to a copy of the window menu. If the *bRevert* parameter is **TRUE**, the return value is **NULL**.

Remarks

Any window that does not use the **GetSystemMenu** function to make its own copy of the window menu receives the standard window menu.

The window menu initially contains items with various identifier values, such as **SC_CLOSE**, **SC_MOVE**, and **SC_SIZE**.

Menu items on the window menu send WM_SYSCOMMAND messages.

All predefined window menu items have identifier numbers greater than 0xF000. If an application adds commands to the window menu, it should use identifier numbers less than 0xF000.

The system automatically grays items on the standard window menu, depending on the situation. The application can perform its own checking or graying by responding to the WM_INITMENU message that is sent before any menu is displayed.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-0 (introduced in Windows 8)

See also

Conceptual

GetMenu

InsertMenuItem

Menus

Reference

SetMenuItemInfo

WM_INITMENU

WM_SYSCOMMAND

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

HiliteMenuItem function (winuser.h)

Article02/22/2024

Adds or removes highlighting from an item in a menu bar.

Syntax

```
C++
```

```
BOOL HiliteMenuItem(
  [in] HWND  hWnd,
  [in] HMENU hMenu,
  [in] UINT  uIDHiliteItem,
  [in] UINT  uHilite
);
```

Parameters

`[in] hWnd`

Type: **HWND**

A handle to the window that contains the menu.

`[in] hMenu`

Type: **HMENU**

A handle to the menu bar that contains the item.

`[in] uIDHiliteItem`

Type: **UINT**

The menu item. This parameter is either the identifier of the menu item or the offset of the menu item in the menu bar, depending on the value of the *uHilite* parameter.

`[in] uHilite`

Type: **UINT**

Controls the interpretation of the *uItemHilite* parameter and indicates whether the menu item is highlighted. This parameter must be a combination of either **MF_BYCOMMAND** or **MF_BYPOSITION** and **MF_HILITE** or **MF_UNHILITE**.

Expand table

Value	Meaning
MF_BYCOMMAND 0x00000000L	Indicates that *uItemHilite* gives the identifier of the menu item.
MF_BYPOSITION 0x00000400L	Indicates that *uItemHilite* gives the zero-based relative position of the menu item.
MF_HILITE 0x00000080L	Highlights the menu item. If this flag is not specified, the highlighting is removed from the item.
MF_UNHILITE 0x00000000L	Removes highlighting from the menu item.

Return value

Type: **BOOL**

If the menu item is set to the specified highlight state, the return value is nonzero.

If the menu item is not set to the specified highlight state, the return value is zero.

Remarks

The **MF_HILITE** and **MF_UNHILITE** flags can be used only with the **HiliteMenuItem** function; they cannot be used with the ModifyMenu function.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

Menus

ModifyMenu

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

InsertMenuA function (winuser.h)

Article02/09/2023

Inserts a new menu item into a menu, moving other items down the menu.

> **Note** The **InsertMenu** function has been superseded by the **InsertMenuItem** function. You can still use **InsertMenu**, however, if you do not need any of the extended features of **InsertMenuItem**.

Syntax

```
C++

BOOL InsertMenuA(
  [in]           HMENU    hMenu,
  [in]           UINT     uPosition,
  [in]           UINT     uFlags,
  [in]           UINT_PTR uIDNewItem,
  [in, optional] LPCSTR   lpNewItem
);
```

Parameters

`[in] hMenu`

Type: **HMENU**

A handle to the menu to be changed.

`[in] uPosition`

Type: **UINT**

The menu item before which the new menu item is to be inserted, as determined by the *uFlags* parameter.

`[in] uFlags`

Type: **UINT**

Controls the interpretation of the *uPosition* parameter and the content, appearance, and behavior of the new menu item. This parameter must include one of the following required values.

Expand table

Value	Meaning
MF_BYCOMMAND 0x00000000L	Indicates that the *uPosition* parameter gives the identifier of the menu item. The **MF_BYCOMMAND** flag is the default if neither the **MF_BYCOMMAND** nor **MF_BYPOSITION** flag is specified.
MF_BYPOSITION 0x00000400L	Indicates that the *uPosition* parameter gives the zero-based relative position of the new menu item. If *uPosition* is -1, the new menu item is appended to the end of the menu.

The parameter must also include at least one of the following values.

Expand table

Value	Meaning
MF_BITMAP 0x00000004L	Uses a bitmap as the menu item. The *lpNewItem* parameter contains a handle to the bitmap.
MF_CHECKED 0x00000008L	Places a check mark next to the menu item. If the application provides check-mark bitmaps (see SetMenuItemBitmaps), this flag displays the check-mark bitmap next to the menu item.
MF_DISABLED 0x00000002L	Disables the menu item so that it cannot be selected, but does not gray it.
MF_ENABLED 0x00000000L	Enables the menu item so that it can be selected and restores it from its grayed state.
MF_GRAYED 0x00000001L	Disables the menu item and grays it so it cannot be selected.
MF_MENUBARBREAK 0x00000020L	Functions the same as the **MF_MENUBREAK** flag for a menu bar. For a drop-down menu, submenu, or shortcut menu, the new column is separated from the old column by a vertical line.
MF_MENUBREAK 0x00000040L	Places the item on a new line (for menu bars) or in a new column (for a drop-down menu, submenu, or shortcut

	menu) without separating columns.
MF_OWNERDRAW 0x00000100L	Specifies that the item is an owner-drawn item. Before the menu is displayed for the first time, the window that owns the menu receives a WM_MEASUREITEM message to retrieve the width and height of the menu item. The WM_DRAWITEM message is then sent to the window procedure of the owner window whenever the appearance of the menu item must be updated.
MF_POPUP 0x00000010L	Specifies that the menu item opens a drop-down menu or submenu. The *uIDNewItem* parameter specifies a handle to the drop-down menu or submenu. This flag is used to add a menu name to a menu bar or a menu item that opens a submenu to a drop-down menu, submenu, or shortcut menu.
MF_SEPARATOR 0x00000800L	Draws a horizontal dividing line. This flag is used only in a drop-down menu, submenu, or shortcut menu. The line cannot be grayed, disabled, or highlighted. The *lpNewItem* and *uIDNewItem* parameters are ignored.
MF_STRING 0x00000000L	Specifies that the menu item is a text string; the *lpNewItem* parameter is a pointer to the string.
MF_UNCHECKED 0x00000000L	Does not place a check mark next to the menu item (default). If the application supplies check-mark bitmaps (see the SetMenuItemBitmaps function), this flag displays the clear bitmap next to the menu item.

`[in] uIDNewItem`

Type: **UINT_PTR**

The identifier of the new menu item or, if the *uFlags* parameter has the **MF_POPUP** flag set, a handle to the drop-down menu or submenu.

`[in, optional] lpNewItem`

Type: **LPCTSTR**

The content of the new menu item. The interpretation of *lpNewItem* depends on whether the *uFlags* parameter includes the **MF_BITMAP**, **MF_OWNERDRAW**, or **MF_STRING** flag, as follows.

Expand table

Value	Meaning

MF_BITMAP 0x00000004L	Contains a bitmap handle.
MF_OWNERDRAW 0x00000100L	Contains an application-supplied value that can be used to maintain additional data related to the menu item. The value is in the **itemData** member of the structure pointed to by the *lParam* parameter of the WM_MEASUREITEM or WM_DRAWITEM message sent when the menu item is created or its appearance is updated.
MF_STRING 0x00000000L	Contains a pointer to a null-terminated string (the default).

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The application must call the DrawMenuBar function whenever a menu changes, whether the menu is in a displayed window.

The following groups of flags cannot be used together:

- **MF_BYCOMMAND** and **MF_BYPOSITION**
- **MF_DISABLED**, **MF_ENABLED**, and **MF_GRAYED**
- **MF_BITMAP**, **MF_STRING**, **MF_OWNERDRAW**, and **MF_SEPARATOR**
- **MF_MENUBARBREAK** and **MF_MENUBREAK**
- **MF_CHECKED** and **MF_UNCHECKED**

> ⓘ **Note**
>
> The winuser.h header defines InsertMenu as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-0 (introduced in Windows 8)

See also

AppendMenu

Conceptual

DeleteMenu

DrawMenuBar

InsertMenuItem

Menus

ModifyMenu

Other Resources

Reference

RemoveMenu

SetMenuItemBitmaps

WM_DRAWITEM

WM_MEASUREITEM

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

InsertMenuItemA function (winuser.h)

Article02/09/2023

Inserts a new menu item at the specified position in a menu.

Syntax

```
C++

BOOL InsertMenuItemA(
  [in] HMENU            hmenu,
  [in] UINT             item,
  [in] BOOL             fByPosition,
  [in] LPCMENUITEMINFOA lpmi
);
```

Parameters

`[in] hmenu`

Type: **HMENU**

A handle to the menu in which the new menu item is inserted.

`[in] item`

Type: **UINT**

The identifier or position of the menu item before which to insert the new item. The meaning of this parameter depends on the value of *fByPosition*.

`[in] fByPosition`

Type: **BOOL**

Controls the meaning of *item*. If this parameter is **FALSE**, *item* is a menu item identifier. Otherwise, it is a menu item position. See Accessing Menu Items Programmatically for more information.

`[in] lpmi`

Type: **LPCMENUITEMINFO**

A pointer to a MENUITEMINFO structure that contains information about the new menu item.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, use the GetLastError function.

Remarks

The application must call the DrawMenuBar function whenever a menu changes, whether the menu is in a displayed window.

In order for keyboard accelerators to work with bitmap or owner-drawn menu items, the owner of the menu must process the WM_MENUCHAR message. See Owner-Drawn Menus and the WM_MENUCHAR Message for more information.

Examples

For an example, see Example of Menu-Item Bitmaps.

> **Note**
>
> The winuser.h header defines InsertMenuItem as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-1 (introduced in Windows 8.1)

See also

Conceptual

DrawMenuBar

MENUITEMINFO

Menus

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

IsMenu function (winuser.h)

Article02/22/2024

Determines whether a handle is a menu handle.

Syntax

```
C++

BOOL IsMenu(
  [in] HMENU hMenu
);
```

Parameters

`[in] hMenu`

Type: **HMENU**

A handle to be tested.

Return value

Type: **BOOL**

If the handle is a menu handle, the return value is nonzero.

If the handle is not a menu handle, the return value is zero.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-1 (introduced in Windows 8.1)

See also

Menus

Feedback

LoadMenuA function (winuser.h)

Article02/09/2023

Loads the specified menu resource from the executable (.exe) file associated with an application instance.

Syntax

```
C++

HMENU LoadMenuA(
  [in, optional] HINSTANCE hInstance,
  [in]           LPCSTR    lpMenuName
);
```

Parameters

`[in, optional] hInstance`

Type: **HINSTANCE**

A handle to the module containing the menu resource to be loaded.

`[in] lpMenuName`

Type: **LPCTSTR**

The name of the menu resource. Alternatively, this parameter can consist of the resource identifier in the low-order word and zero in the high-order word. To create this value, use the MAKEINTRESOURCE macro.

Return value

Type: **HMENU**

If the function succeeds, the return value is a handle to the menu resource.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

The DestroyMenu function is used, before an application closes, to destroy the menu and free memory that the loaded menu occupied.

Examples

For an example, see Displaying a Shortcut Menu

> **Note**
>
> The winuser.h header defines LoadMenu as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-3 (introduced in Windows 10, version 10.0.14393)

See also

Conceptual

LoadMenuIndirect

MAKEINTRESOURCE

Menus

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

LoadMenuIndirectA function (winuser.h)

Article02/22/2024

Loads the specified menu template in memory.

Syntax

```
C++
```

```
HMENU LoadMenuIndirectA(
  [in] const MENUTEMPLATEA *lpMenuTemplate
);
```

Parameters

`[in] lpMenuTemplate`

Type: **const MENUTEMPLATE***

A pointer to a menu template or an extended menu template. A menu template consists of a MENUITEMTEMPLATEHEADER structure followed by one or more contiguous MENUITEMTEMPLATE structures. An extended menu template consists of a MENUEX_TEMPLATE_HEADER structure followed by one or more contiguous MENUEX_TEMPLATE_ITEM structures.

Return value

Type: **HMENU**

If the function succeeds, the return value is a handle to the menu.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

For both the ANSI and the Unicode version of this function, the strings in the MENUITEMTEMPLATE structure must be Unicode strings.

> **Note**
>
> The winuser.h header defines LoadMenuIndirect as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

LoadMenu

MENUEX_TEMPLATE_HEADER

MENUEX_TEMPLATE_ITEM

MENUITEMTEMPLATE

MENUITEMTEMPLATEHEADER

Menus

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MenuItemFromPoint function (winuser.h)

Article11/19/2022

Determines which menu item, if any, is at the specified location.

Syntax

```
C++
```

```
int MenuItemFromPoint(
  [in, optional] HWND  hWnd,
  [in]           HMENU hMenu,
  [in]           POINT ptScreen
);
```

Parameters

`[in, optional] hWnd`

Type: **HWND**

A handle to the window containing the menu. If this value is **NULL** and the *hMenu* parameter represents a popup menu, the function will find the menu window.

`[in] hMenu`

Type: **HMENU**

A handle to the menu containing the menu items to hit test.

`[in] ptScreen`

Type: **POINT**

A structure that specifies the location to test. If *hMenu* specifies a menu bar, this parameter is in window coordinates. Otherwise, it is in client coordinates.

Return value

Type: **int**

Returns the zero-based position of the menu item at the specified location or -1 if no menu item is at the specified location.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Menus

Feedback

ModifyMenuA function (winuser.h)

Article02/09/2023

Changes an existing menu item. This function is used to specify the content, appearance, and behavior of the menu item.

> **Note** The **ModifyMenu** function has been superseded by the **SetMenuItemInfo** function. You can still use **ModifyMenu**, however, if you do not need any of the extended features of **SetMenuItemInfo**.

Syntax

```
C++

BOOL ModifyMenuA(
  [in]           HMENU    hMnu,
  [in]           UINT     uPosition,
  [in]           UINT     uFlags,
  [in]           UINT_PTR uIDNewItem,
  [in, optional] LPCSTR   lpNewItem
);
```

Parameters

`[in] hMnu`

Type: **HMENU**

A handle to the menu to be changed.

`[in] uPosition`

Type: **UINT**

The menu item to be changed, as determined by the *uFlags* parameter.

`[in] uFlags`

Type: **UINT**

Controls the interpretation of the *uPosition* parameter and the content, appearance, and behavior of the menu item. This parameter must include one of the following required values.

Expand table

Value	Meaning
MF_BYCOMMAND 0x00000000L	Indicates that the *uPosition* parameter gives the identifier of the menu item. The **MF_BYCOMMAND** flag is the default if neither the **MF_BYCOMMAND** nor **MF_BYPOSITION** flag is specified.
MF_BYPOSITION 0x00000400L	Indicates that the *uPosition* parameter gives the zero-based relative position of the menu item.

The parameter must also include at least one of the following values.

Expand table

Value	Meaning
MF_BITMAP 0x00000004L	Uses a bitmap as the menu item. The *lpNewItem* parameter contains a handle to the bitmap.
MF_CHECKED 0x00000008L	Places a check mark next to the item. If your application provides check-mark bitmaps (see the SetMenuItemBitmaps function), this flag displays a selected bitmap next to the menu item.
MF_DISABLED 0x00000002L	Disables the menu item so that it cannot be selected, but this flag does not gray it.
MF_ENABLED 0x00000000L	Enables the menu item so that it can be selected and restores it from its grayed state.
MF_GRAYED 0x00000001L	Disables the menu item and grays it so that it cannot be selected.
MF_MENUBARBREAK 0x00000020L	Functions the same as the **MF_MENUBREAK** flag for a menu bar. For a drop-down menu, submenu, or shortcut menu, the new column is separated from the old column by a vertical line.
MF_MENUBREAK 0x00000040L	Places the item on a new line (for menu bars) or in a new column (for a drop-down menu, submenu, or shortcut menu) without separating columns.

MF_OWNERDRAW 0x00000100L	Specifies that the item is an owner-drawn item. Before the menu is displayed for the first time, the window that owns the menu receives a WM_MEASUREITEM message to retrieve the width and height of the menu item. The WM_DRAWITEM message is then sent to the window procedure of the owner window whenever the appearance of the menu item must be updated.
MF_POPUP 0x00000010L	Specifies that the menu item opens a drop-down menu or submenu. The *uIDNewItem* parameter specifies a handle to the drop-down menu or submenu. This flag is used to add a menu name to a menu bar or a menu item that opens a submenu to a drop-down menu, submenu, or shortcut menu.
MF_SEPARATOR 0x00000800L	Draws a horizontal dividing line. This flag is used only in a drop-down menu, submenu, or shortcut menu. The line cannot be grayed, disabled, or highlighted. The *lpNewItem* and *uIDNewItem* parameters are ignored.
MF_STRING 0x00000000L	Specifies that the menu item is a text string; the *lpNewItem* parameter is a pointer to the string.
MF_UNCHECKED 0x00000000L	Does not place a check mark next to the item (the default). If your application supplies check-mark bitmaps (see the SetMenuItemBitmaps function), this flag displays a clear bitmap next to the menu item.

`[in] uIDNewItem`

Type: **UINT_PTR**

The identifier of the modified menu item or, if the *uFlags* parameter has the **MF_POPUP** flag set, a handle to the drop-down menu or submenu.

`[in, optional] lpNewItem`

Type: **LPCTSTR**

The contents of the changed menu item. The interpretation of this parameter depends on whether the *uFlags* parameter includes the **MF_BITMAP**, **MF_OWNERDRAW**, or **MF_STRING** flag.

Expand table

Value	Meaning
MF_BITMAP	A bitmap handle.

0x00000004L	
MF_OWNERDRAW 0x00000100L	A value supplied by an application that is used to maintain additional data related to the menu item. The value is in the **itemData** member of the structure pointed to by the *lParam* parameter of the WM_MEASUREITEM or WM_DRAWITEM messages sent when the menu item is created or its appearance is updated.
MF_STRING 0x00000000L	A pointer to a null-terminated string (the default).

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

If **ModifyMenu** replaces a menu item that opens a drop-down menu or submenu, the function destroys the old drop-down menu or submenu and frees the memory used by it.

In order for keyboard accelerators to work with bitmap or owner-drawn menu items, the owner of the menu must process the WM_MENUCHAR message. See Owner-Drawn Menus and the WM_MENUCHAR Message for more information.

The application must call the DrawMenuBar function whenever a menu changes, whether the menu is in a displayed window. To change the attributes of existing menu items, it is much faster to use the CheckMenuItem and EnableMenuItem functions.

The following groups of flags cannot be used together:

- **MF_BYCOMMAND** and **MF_BYPOSITION**
- **MF_DISABLED**, **MF_ENABLED**, and **MF_GRAYED**
- **MF_BITMAP**, **MF_STRING**, **MF_OWNERDRAW**, and **MF_SEPARATOR**
- **MF_MENUBARBREAK** and **MF_MENUBREAK**
- **MF_CHECKED** and **MF_UNCHECKED**

Examples

For an example, see Setting Fonts for Menu-Item Text Strings.

> **Note**
>
> The winuser.h header defines ModifyMenu as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-3 (introduced in Windows 10, version 10.0.14393)

See also

AppendMenu

CheckMenuItem

Conceptual

DrawMenuBar

EnableMenuItem

Menus

Reference

SetMenuItemBitmaps

SetMenuItemInfo

Feedback

RemoveMenu function (winuser.h)

Article02/22/2024

Deletes a menu item or detaches a submenu from the specified menu. If the menu item opens a drop-down menu or submenu, **RemoveMenu** does not destroy the menu or its handle, allowing the menu to be reused. Before this function is called, the GetSubMenu function should retrieve a handle to the drop-down menu or submenu.

Syntax

```
C++

BOOL RemoveMenu(
  [in] HMENU hMenu,
  [in] UINT  uPosition,
  [in] UINT  uFlags
);
```

Parameters

`[in] hMenu`

Type: **HMENU**

A handle to the menu to be changed.

`[in] uPosition`

Type: **UINT**

The menu item to be deleted, as determined by the *uFlags* parameter.

`[in] uFlags`

Type: **UINT**

Indicates how the *uPosition* parameter is interpreted. This parameter must be one of the following values.

Expand table

Value	Meaning

MF_BYCOMMAND 0x00000000L	Indicates that *uPosition* gives the identifier of the menu item. If neither the **MF_BYCOMMAND** nor **MF_BYPOSITION** flag is specified, the **MF_BYCOMMAND** flag is the default flag.
MF_BYPOSITION 0x00000400L	Indicates that *uPosition* gives the zero-based relative position of the menu item.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The application must call the DrawMenuBar function whenever a menu changes, whether the menu is in a displayed window.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-0 (introduced in Windows 8)

See also

Conceptual

CreatePopupMenu

DeleteMenu

DrawMenuBar

GetSubMenu

Menus

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

SetMenu function (winuser.h)

Article02/22/2024

Assigns a new menu to the specified window.

Syntax

```
C++
```

```
BOOL SetMenu(
  [in]           HWND  hWnd,
  [in, optional] HMENU hMenu
);
```

Parameters

`[in] hWnd`

Type: **HWND**

A handle to the window to which the menu is to be assigned.

`[in, optional] hMenu`

Type: **HMENU**

A handle to the new menu. If this parameter is **NULL**, the window's current menu is removed.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The window is redrawn to reflect the menu change. A menu can be assigned to any window that is not a child window.

The **SetMenu** function replaces the previous menu, if any, but it does not destroy it. An application should call the DestroyMenu function to accomplish this task.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-3 (introduced in Windows 10, version 10.0.14393)

See also

Conceptual

DestroyMenu

GetMenu

Menus

Reference

Feedback

Was this page helpful?

Yes No

Provide product feedback | Get help at Microsoft Q&A

SetMenuDefaultItem function (winuser.h)

Article02/22/2024

Sets the default menu item for the specified menu.

Syntax

```
C++

BOOL SetMenuDefaultItem(
  [in] HMENU hMenu,
  [in] UINT  uItem,
  [in] UINT  fByPos
);
```

Parameters

`[in] hMenu`

Type: **HMENU**

A handle to the menu to set the default item for.

`[in] uItem`

Type: **UINT**

The identifier or position of the new default menu item or -1 for no default item. The meaning of this parameter depends on the value of *fByPos*.

`[in] fByPos`

Type: **UINT**

The meaning of *uItem*. If this parameter is **FALSE**, *uItem* is a menu item identifier. Otherwise, it is a menu item position. See About Menus for more information.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, use the GetLastError function.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-2 (introduced in Windows 10, version 10.0.10240)

See also

Conceptual

GetMenuDefaultItem

Menus

Reference

Feedback

SetMenuInfo function (winuser.h)

Article02/22/2024

Sets information for a specified menu.

Syntax

```
C++

BOOL SetMenuInfo(
  [in] HMENU       unnamedParam1,
  [in] LPCMENUINFO unnamedParam2
);
```

Parameters

`[in] unnamedParam1`

Type: **HMENU**

A handle to a menu.

`[in] unnamedParam2`

Type: **LPCMENUINFO**

A pointer to a MENUINFO structure for the menu.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-3 (introduced in Windows 10, version 10.0.14393)

See also

Menus

Feedback

SetMenuItemBitmaps function (winuser.h)

Article10/13/2021

Associates the specified bitmap with a menu item. Whether the menu item is selected or clear, the system displays the appropriate bitmap next to the menu item.

Syntax

C++

```
BOOL SetMenuItemBitmaps(
  [in]           HMENU   hMenu,
  [in]           UINT    uPosition,
  [in]           UINT    uFlags,
  [in, optional] HBITMAP hBitmapUnchecked,
  [in, optional] HBITMAP hBitmapChecked
);
```

Parameters

`[in] hMenu`

Type: **HMENU**

A handle to the menu containing the item to receive new check-mark bitmaps.

`[in] uPosition`

Type: **UINT**

The menu item to be changed, as determined by the *uFlags* parameter.

`[in] uFlags`

Type: **UINT**

Specifies how the *uPosition* parameter is to be interpreted. The *uFlags* parameter must be one of the following values.

Expand table

Value	Meaning
MF_BYCOMMAND 0x00000000L	Indicates that *uPosition* gives the identifier of the menu item. If neither **MF_BYCOMMAND** nor **MF_BYPOSITION** is specified, **MF_BYCOMMAND** is the default flag.
MF_BYPOSITION 0x00000400L	Indicates that *uPosition* gives the zero-based relative position of the menu item.

`[in, optional] hBitmapUnchecked`

Type: **HBITMAP**

A handle to the bitmap displayed when the menu item is not selected.

`[in, optional] hBitmapChecked`

Type: **HBITMAP**

A handle to the bitmap displayed when the menu item is selected.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

If either the *hBitmapUnchecked* or *hBitmapChecked* parameter is **NULL**, the system displays nothing next to the menu item for the corresponding check state. If both parameters are **NULL**, the system displays the default check-mark bitmap when the item is selected, and removes the bitmap when the item is not selected.

When the menu is destroyed, these bitmaps are not destroyed; it is up to the application to destroy them.

The selected and clear bitmaps should be monochrome. The system uses the Boolean AND operator to combine bitmaps with the menu so that the white part becomes transparent and the black part becomes the menu-item color. If you use color bitmaps, the results may be undesirable.

Use the GetSystemMetrics function with the **SM_CXMENUCHECK** and **SM_CYMENUCHECK** values to retrieve the bitmap dimensions.

Examples

For an example, see Simulating Check Boxes in a Menu.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Menus

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

SetMenuItemInfoA function (winuser.h)

Article02/22/2024

Changes information about a menu item.

Syntax

```
C++

BOOL SetMenuItemInfoA(
  [in] HMENU            hmenu,
  [in] UINT             item,
       BOOL             fByPositon,
  [in] LPCMENUITEMINFOA lpmii
);
```

Parameters

`[in] hmenu`

Type: **HMENU**

A handle to the menu that contains the menu item.

`[in] item`

Type: **UINT**

The identifier or position of the menu item to change. The meaning of this parameter depends on the value of *fByPosition*.

`fByPositon`

`[in] lpmii`

Type: **LPMENUITEMINFO**

A pointer to a MENUITEMINFO structure that contains information about the menu item and specifies which menu item attributes to change.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, use the GetLastError function.

Remarks

The application must call the DrawMenuBar function whenever a menu changes, whether the menu is in a displayed window.

In order for keyboard accelerators to work with bitmap or owner-drawn menu items, the owner of the menu must process the WM_MENUCHAR message. See Owner-Drawn Menus and the WM_MENUCHAR Message for more information.

Examples

For an example, see Example of Owner-Drawn Menu Items.

> **Note**
>
> The winuser.h header defines SetMenuItemInfo as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

Requirement	Value
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-0 (introduced in Windows 8)

See also

Conceptual

DrawMenuBar

GetMenuItemInfo

MENUITEMINFO

Menus

Reference

Feedback

TrackPopupMenu function (winuser.h)

Article10/13/2021

Displays a shortcut menu at the specified location and tracks the selection of items on the menu. The shortcut menu can appear anywhere on the screen.

Syntax

```
C++

BOOL TrackPopupMenu(
  [in]           HMENU      hMenu,
  [in]           UINT       uFlags,
  [in]           int        x,
  [in]           int        y,
  [in]           int        nReserved,
  [in]           HWND       hWnd,
  [in, optional] const RECT *prcRect
);
```

Parameters

`[in] hMenu`

Type: **HMENU**

A handle to the shortcut menu to be displayed. The handle can be obtained by calling CreatePopupMenu to create a new shortcut menu, or by calling GetSubMenu to retrieve a handle to a submenu associated with an existing menu item.

`[in] uFlags`

Type: **UINT**

Use zero of more of these flags to specify function options.

Use one of the following flags to specify how the function positions the shortcut menu horizontally.

Expand table

Value	Meaning

TPM_CENTERALIGN 0x0004L	Centers the shortcut menu horizontally relative to the coordinate specified by the *x* parameter.
TPM_LEFTALIGN 0x0000L	Positions the shortcut menu so that its left side is aligned with the coordinate specified by the *x* parameter.
TPM_RIGHTALIGN 0x0008L	Positions the shortcut menu so that its right side is aligned with the coordinate specified by the *x* parameter.

Use one of the following flags to specify how the function positions the shortcut menu vertically.

Expand table

Value	Meaning
TPM_BOTTOMALIGN 0x0020L	Positions the shortcut menu so that its bottom side is aligned with the coordinate specified by the *y* parameter.
TPM_TOPALIGN 0x0000L	Positions the shortcut menu so that its top side is aligned with the coordinate specified by the *y* parameter.
TPM_VCENTERALIGN 0x0010L	Centers the shortcut menu vertically relative to the coordinate specified by the *y* parameter.

Use the following flags to control discovery of the user selection without having to set up a parent window for the menu.

Expand table

Value	Meaning
TPM_NONOTIFY 0x0080L	The function does not send notification messages when the user clicks a menu item.
TPM_RETURNCMD 0x0100L	The function returns the menu item identifier of the user's selection in the return value.

Use one of the following flags to specify which mouse button the shortcut menu tracks.

Expand table

Value	Meaning

TPM_LEFTBUTTON 0x0000L	The user can select menu items with only the left mouse button.
TPM_RIGHTBUTTON 0x0002L	The user can select menu items with both the left and right mouse buttons.

Use any reasonable combination of the following flags to modify the animation of a menu. For example, by selecting a horizontal and a vertical flag, you can achieve diagonal animation.

Expand table

Value	Meaning
TPM_HORNEGANIMATION 0x0800L	Animates the menu from right to left.
TPM_HORPOSANIMATION 0x0400L	Animates the menu from left to right.
TPM_NOANIMATION 0x4000L	Displays menu without animation.
TPM_VERNEGANIMATION 0x2000L	Animates the menu from bottom to top.
TPM_VERPOSANIMATION 0x1000L	Animates the menu from top to bottom.

For any animation to occur, the SystemParametersInfo function must set **SPI_SETMENUANIMATION**. Also, all the TPM_*ANIMATION flags, except **TPM_NOANIMATION**, are ignored if menu fade animation is on. For more information, see the **SPI_GETMENUFADE** flag in **SystemParametersInfo**.

Use the **TPM_RECURSE** flag to display a menu when another menu is already displayed. This is intended to support context menus within a menu.

For right-to-left text layout, use **TPM_LAYOUTRTL**. By default, the text layout is left-to-right.

`[in] x`

Type: **int**

The horizontal location of the shortcut menu, in screen coordinates.

`[in] y`

Type: **int**

The vertical location of the shortcut menu, in screen coordinates.

`[in] nReserved`

Type: **int**

Reserved; must be zero.

`[in] hWnd`

Type: **HWND**

A handle to the window that owns the shortcut menu. This window receives all messages from the menu. The window does not receive a WM_COMMAND message from the menu until the function returns. If you specify TPM_NONOTIFY in the *uFlags* parameter, the function does not send messages to the window identified by *hWnd*. However, you must still pass a window handle in *hWnd*. It can be any window handle from your application.

`[in, optional] prcRect`

Type: **const RECT***

Ignored.

Return value

Type: **BOOL**

If you specify **TPM_RETURNCMD** in the *uFlags* parameter, the return value is the menu-item identifier of the item that the user selected. If the user cancels the menu without making a selection, or if an error occurs, the return value is zero.

If you do not specify **TPM_RETURNCMD** in the *uFlags* parameter, the return value is nonzero if the function succeeds and zero if it fails. To get extended error information, call GetLastError.

Remarks

Call GetSystemMetrics with **SM_MENUDROPALIGNMENT** to determine the correct horizontal alignment flag (**TPM_LEFTALIGN** or **TPM_RIGHTALIGN**) and/or horizontal animation direction flag (**TPM_HORPOSANIMATION** or **TPM_HORNEGANIMATION**) to pass to **TrackPopupMenu** or TrackPopupMenuEx. This is essential for creating an optimal user experience, especially when developing Microsoft Tablet PC applications.

To specify an area of the screen that the menu should not overlap, use the TrackPopupMenuEx function

To display a context menu for a notification icon, the current window must be the foreground window before the application calls **TrackPopupMenu** or TrackPopupMenuEx. Otherwise, the menu will not disappear when the user clicks outside of the menu or the window that created the menu (if it is visible). If the current window is a child window, you must set the (top-level) parent window as the foreground window.

However, when the current window is the foreground window, the second time this menu is displayed, it appears and then immediately disappears. To correct this, you must force a task switch to the application that called **TrackPopupMenu**. This is done by posting a benign message to the window or thread, as shown in the following code sample:

```
SetForegroundWindow(hDlg);

// Display the menu
TrackPopupMenu(   hSubMenu,
                  TPM_RIGHTBUTTON,
                  pt.x,
                  pt.y,
                  0,
                  hDlg,
                  NULL);

PostMessage(hDlg, WM_NULL, 0, 0);
```

Examples

For an example, see Displaying a Shortcut Menu.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-menu-l1-1-0 (introduced in Windows 8)

See also

Conceptual

CreatePopupMenu

GetSubMenu

Menus

Reference

TrackPopupMenuEx

WM_COMMAND

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

TrackPopupMenuEx function (winuser.h)

Article10/13/2021

Displays a shortcut menu at the specified location and tracks the selection of items on the shortcut menu. The shortcut menu can appear anywhere on the screen.

Syntax

```
C++

BOOL TrackPopupMenuEx(
  [in]           HMENU       hMenu,
  [in]           UINT        uFlags,
  [in]           int         x,
  [in]           int         y,
  [in]           HWND        hwnd,
  [in, optional] LPTPMPARAMS lptpm
);
```

Parameters

`[in] hMenu`

Type: **HMENU**

A handle to the shortcut menu to be displayed. This handle can be obtained by calling the CreatePopupMenu function to create a new shortcut menu or by calling the GetSubMenu function to retrieve a handle to a submenu associated with an existing menu item.

`[in] uFlags`

Type: **UINT**

Specifies function options.

Use one of the following flags to specify how the function positions the shortcut menu horizontally.

Expand table

Value	Meaning

TPM_CENTERALIGN 0x0004L	Centers the shortcut menu horizontally relative to the coordinate specified by the *x* parameter.
TPM_LEFTALIGN 0x0000L	Positions the shortcut menu so that its left side is aligned with the coordinate specified by the *x* parameter.
TPM_RIGHTALIGN 0x0008L	Positions the shortcut menu so that its right side is aligned with the coordinate specified by the *x* parameter.

Use one of the following flags to specify how the function positions the shortcut menu vertically.

Expand table

Value	Meaning
TPM_BOTTOMALIGN 0x0020L	Positions the shortcut menu so that its bottom side is aligned with the coordinate specified by the *y* parameter.
TPM_TOPALIGN 0x0000L	Positions the shortcut menu so that its top side is aligned with the coordinate specified by the *y* parameter.
TPM_VCENTERALIGN 0x0010L	Centers the shortcut menu vertically relative to the coordinate specified by the *y* parameter.

Use the following flags to control discovery of the user selection without having to set up a parent window for the menu.

Expand table

Value	Meaning
TPM_NONOTIFY 0x0080L	The function does not send notification messages when the user clicks a menu item.
TPM_RETURNCMD 0x0100L	The function returns the menu item identifier of the user's selection in the return value.

Use one of the following flags to specify which mouse button the shortcut menu tracks.

Expand table

Value	Meaning

TPM_LEFTBUTTON 0x0000L	The user can select menu items with only the left mouse button.
TPM_RIGHTBUTTON 0x0002L	The user can select menu items with both the left and right mouse buttons.

Use any reasonable combination of the following flags to modify the animation of a menu. For example, by selecting a horizontal and a vertical flag, you can achieve diagonal animation.

Expand table

Value	Meaning
TPM_HORNEGANIMATION 0x0800L	Animates the menu from right to left.
TPM_HORPOSANIMATION 0x0400L	Animates the menu from left to right.
TPM_NOANIMATION 0x4000L	Displays menu without animation.
TPM_VERNEGANIMATION 0x2000L	Animates the menu from bottom to top.
TPM_VERPOSANIMATION 0x1000L	Animates the menu from top to bottom.

For any animation to occur, the SystemParametersInfo function must set **SPI_SETMENUANIMATION**. Also, all the **TPM_*ANIMATION** flags, except **TPM_NOANIMATION**, are ignored if menu fade animation is on. For more information, see the **SPI_GETMENUFADE** flag in **SystemParametersInfo**.

Use the **TPM_RECURSE** flag to display a menu when another menu is already displayed. This is intended to support context menus within a menu.

Use one of the following flags to specify whether to accommodate horizontal or vertical alignment.

Expand table

Value	Meaning

TPM_HORIZONTAL 0x0000L	If the menu cannot be shown at the specified location without overlapping the excluded rectangle, the system tries to accommodate the requested horizontal alignment before the requested vertical alignment.
TPM_VERTICAL 0x0040L	If the menu cannot be shown at the specified location without overlapping the excluded rectangle, the system tries to accommodate the requested vertical alignment before the requested horizontal alignment.

The excluded rectangle is a portion of the screen that the menu should not overlap; it is specified by the *lptpm* parameter.

For right-to-left text layout, use **TPM_LAYOUTRTL**. By default, the text layout is left-to-right.

`[in] x`

Type: **int**

The horizontal location of the shortcut menu, in screen coordinates.

`[in] y`

Type: **int**

The vertical location of the shortcut menu, in screen coordinates.

`[in] hwnd`

Type: **HWND**

A handle to the window that owns the shortcut menu. This window receives all messages from the menu. The window does not receive a WM_COMMAND message from the menu until the function returns. If you specify TPM_NONOTIFY in the *fuFlags* parameter, the function does not send messages to the window identified by *hwnd*. However, you must still pass a window handle in *hwnd*. It can be any window handle from your application.

`[in, optional] lptpm`

Type: **LPTPMPARAMS**

A pointer to a TPMPARAMS structure that specifies an area of the screen the menu should not overlap. This parameter can be **NULL**.

Return value

Type: **BOOL**

If you specify **TPM_RETURNCMD** in the *fuFlags* parameter, the return value is the menu-item identifier of the item that the user selected. If the user cancels the menu without making a selection, or if an error occurs, the return value is zero.

If you do not specify **TPM_RETURNCMD** in the *fuFlags* parameter, the return value is nonzero if the function succeeds and zero if it fails. To get extended error information, call GetLastError.

Remarks

Call GetSystemMetrics with **SM_MENUDROPALIGNMENT** to determine the correct horizontal alignment flag (**TPM_LEFTALIGN** or **TPM_RIGHTALIGN**) and/or horizontal animation direction flag (**TPM_HORPOSANIMATION** or **TPM_HORNEGANIMATION**) to pass to TrackPopupMenu or **TrackPopupMenuEx**. This is essential for creating an optimal user experience, especially when developing Microsoft Tablet PC applications.

To display a context menu for a notification icon, the current window must be the foreground window before the application calls TrackPopupMenu or **TrackPopupMenuEx**. Otherwise, the menu will not disappear when the user clicks outside of the menu or the window that created the menu (if it is visible). If the current window is a child window, you must set the (top-level) parent window as the foreground window.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

Requirement	Value
API set	ext-ms-win-ntuser-menu-l1-1-1 (introduced in Windows 8.1)

See also

Conceptual

CreatePopupMenu

GetSubMenu

Menus

Reference

TPMPARAMS

WM_COMMAND

Feedback

Menu Notifications

Article • 04/27/2021

In This Section

- WM_COMMAND
- WM_CONTEXTMENU
- WM_ENTERMENULOOP
- WM_EXITMENULOOP
- WM_GETTITLEBARINFOEX
- WM_MENUCOMMAND
- WM_MENUDRAG
- WM_MENUGETOBJECT
- WM_MENURBUTTONUP
- WM_NEXTMENU
- WM_UNINITMENUPOPUP

Example

C

```
BOOL AboutDlg (
    HWND hDlg,
    UINT message,
    WPARAM wParam,
    LPARAM lParam)
{
    BOOL bRet = FALSE;

    switch (message)
    {
        case WM_INITDIALOG:
            bRet = TRUE;
            break;

        case WM_COMMAND:
            if (wParam == IDOK ||
                wParam == IDCANCEL)
            {
                EndDialog(hDlg, TRUE);
                bRet = TRUE;
            }
            break;
    }
```

```
    return bRet;
}
```

Example taken from Windows classic samples on GitHub.

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

WM_COMMAND message

Article • 03/22/2021

Sent when the user selects a command item from a menu, when a control sends a notification message to its parent window, or when an accelerator keystroke is translated.

C++

```
#define WM_COMMAND                      0x0111
```

Parameters

wParam

For a description of this parameter, see Remarks.

lParam

For a description of this parameter, see Remarks.

Return value

If an application processes this message, it should return zero.

Example

C

```
BOOL AboutDlg (
    HWND hDlg,
    UINT message,
    WPARAM wParam,
    LPARAM lParam)
{
    BOOL bRet = FALSE;

    switch (message)
    {
        case WM_INITDIALOG:
            bRet = TRUE;
            break;

        case WM_COMMAND:
```

```
            if (wParam == IDOK ||
                wParam == IDCANCEL)
            {
                EndDialog(hDlg, TRUE);
                bRet = TRUE;
            }
            break;
    }

    return bRet;
}
```

Example taken from Windows classic samples on GitHub.

Remarks

Use of the *wParam* and *lParam* parameters are summarized here.

Expand table

Message Source	wParam (high word)	wParam (low word)	lParam
Menu	0	Menu identifier (IDM_*)	0
Accelerator	1	Accelerator identifier (IDM_*)	0
Control	Control-defined notification code	Control identifier	Handle to the control window

Menus

If an application enables a menu separator, the system sends a **WM_COMMAND** message with the low-word of the *wParam* parameter set to zero when the user selects the separator.

If a menu is defined with a MENUINFO.dwStyle value of **MNS_NOTIFYBYPOS**, WM_MENUCOMMAND is sent instead of **WM_COMMAND**.

Accelerators

Accelerator keystrokes that select items from the window menu are translated into WM_SYSCOMMAND messages.

If an accelerator keystroke occurs that corresponds to a menu item when the window that owns the menu is minimized, no **WM_COMMAND** message is sent. However, if an accelerator keystroke occurs that does not match any of the items in the window's menu or in the window menu, a **WM_COMMAND** message is sent, even if the window is minimized.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Reference

HIWORD

LOWORD

Conceptual

Menus

Feedback

Was this page helpful? Yes No

WM_CONTEXTMENU message

Article • 12/11/2020

Notifies a window that the user desires a context menu to appear. The user may have clicked the right mouse button (right-clicked) in the window, pressed Shift+F10 or pressed the applications key (context menu key) available on some keyboards.

C++

```
#define WM_CONTEXTMENU                  0x007B
```

Parameters

wParam

A handle to the window in which the user right-clicked the mouse. This can be a child window of the window receiving the message. For more information about processing this message, see the Remarks section.

lParam

The low-order word specifies the horizontal position of the cursor, in screen coordinates, at the time of the mouse click.

The high-order word specifies the vertical position of the cursor, in screen coordinates, at the time of the mouse click.

Return value

No return value.

Remarks

A window can process this message by displaying a shortcut menu using the **TrackPopupMenu** or **TrackPopupMenuEx** functions. To obtain the horizontal and vertical positions, use the following code.

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

If a window does not display a shortcut menu it should pass this message to the DefWindowProc function. If a window is a child window, **DefWindowProc** sends the message to the parent. Otherwise, **DefWindowProc** displays a default shortcut menu if the specified position is in the window's caption.

DefWindowProc generates the **WM_CONTEXTMENU** message when it processes the WM_RBUTTONUP or WM_NCRBUTTONUP message or when the user types SHIFT+F10. The **WM_CONTEXTMENU** message is also generated when the user presses and releases the VK_APPS key.

If the context menu is generated from the keyboard for example, if the user types SHIFT+F10 then the x- and y-coordinates are -1 and the application should display the context menu at the location of the current selection rather than at (xPos, yPos).

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Reference

DefWindowProc

GET_X_LPARAM

GET_Y_LPARAM

TrackPopupMenu

TrackPopupMenuEx

WM_NCRBUTTONUP

WM_RBUTTONUP

Conceptual

Menus

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

WM_ENTERMENULOOP message

Article • 12/11/2020

Notifies an application's main window procedure that a menu modal loop has been entered.

```
C++

#define WM_ENTERMENULOOP                0x0211
```

Parameters

wParam

Specifies whether the window menu was entered using the **TrackPopupMenu** function. This parameter has a value of **TRUE** if the window menu was entered using **TrackPopupMenu**, and **FALSE** if it was not.

lParam

This parameter is not used.

Return value

An application should return zero if it processes this message.

Remarks

The **DefWindowProc** function returns zero.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Reference

DefWindowProc

WM_EXITMENULOOP

Conceptual

Menus

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

WM_EXITMENULOOP message

Article • 12/11/2020

Notifies an application's main window procedure that a menu modal loop has been exited.

```
C++

#define WM_EXITMENULOOP                 0x0212
```

Parameters

wParam

Specifies whether the menu is a shortcut menu. This parameter has a value of **TRUE** if it is a shortcut menu, **FALSE** if it is not.

lParam

This parameter is not used.

Return value

An application should return zero if it processes this message.

Remarks

The **DefWindowProc** function returns zero.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Reference

DefWindowProc

WM_ENTERMENULOOP

Conceptual

Menus

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

WM_GETTITLEBARINFOEX message

Article • 12/11/2020

Sent to request extended title bar information. A window receives this message through its **WindowProc** function.

```
C++

#define WM_GETTITLEBARINFOEX            0x033F
```

Parameters

wParam

This parameter is not used and must be 0.

lParam

A pointer to a **TITLEBARINFOEX** structure. The message sender is responsible for allocating memory for this structure. Set the **cbSize** member of this structure to `sizeof(TITLEBARINFOEX)` before passing this structure with the message.

Remarks

The following example shows how the message receiver casts an **LPARAM** value to retrieve the **TITLEBARINFOEX** structure.

```
TITLEBARINFOEX *ptinfo = (TITLEBARINFOEX *)lParam;
```

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Menus Overview

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

WM_MENUCOMMAND message

Article • 12/11/2020

Sent when the user makes a selection from a menu.

```
C++

#define WM_MENUCOMMAND                  0x0126
```

Parameters

wParam

The zero-based index of the item selected.

lParam

A handle to the menu for the item selected.

Remarks

The **WM_MENUCOMMAND** message gives you a handle to the menu so you can access the menu data in the MENUINFO structure and also gives you the index of the selected item, which is typically what applications need. In contrast, the WM_COMMAND message gives you the menu item identifier.

The **WM_MENUCOMMAND** message is sent only for menus that are defined with the **MNS_NOTIFYBYPOS** flag set in the **dwStyle** member of the MENUINFO structure.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Menus Overview

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

WM_MENUDRAG message

Article • 12/11/2020

Sent to the owner of a drag-and-drop menu when the user drags a menu item.

```
C++

#define WM_MENUDRAG                     0x0123
```

Parameters

wParam

The position of the item where the drag operation began.

lParam

A handle to the menu containing the item.

Return value

The application should return one of the following values.

Return code/value	Description
MND_CONTINUE 0	Menu should remain active. If the mouse is released, it should be ignored.
MND_ENDMENU 1	Menu should be ended.

Remarks

The application can call the **DoDragDrop** function in response to this message.

To create a drag-and-drop menu, call **SetMenuInfo** with **MNS_DRAGDROP**.

Requirements

Requirement	Value

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Reference

SetMenuInfo

Conceptual

Menus

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

WM_MENUGETOBJECT message

Article • 12/11/2020

Sent to the owner of a drag-and-drop menu when the mouse cursor enters a menu item or moves from the center of the item to the top or bottom of the item.

```
C++

#define WM_MENUGETOBJECT                0x0124
```

Parameters

wParam

This parameter is not used.

lParam

A pointer to a MENUGETOBJECTINFO structure.

Return value

The application should return one of the following values.

Return code/value	Description
MNGO_NOERROR 0x00000001	An interface pointer was returned in the **pvObj** member of MENUGETOBJECTINFO
MNGO_NOINTERFACE 0x00000000	The interface is not supported.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Menus Overview

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

WM_MENURBUTTONUP message

Article • 12/11/2020

Sent when the user releases the right mouse button while the cursor is on a menu item.

```
C++

#define WM_MENURBUTTONUP                0x0122
```

Parameters

wParam

The zero-based index of the menu item on which the right mouse button was released.

lParam

A handle to the menu containing the item.

Remarks

The **WM_MENURBUTTONUP** message allows applications to provide a context-sensitive menu also known as a shortcut menu for the menu item specified in this message. To display a context-sensitive menu for a menu item, call the **TrackPopupMenuEx** function with **TPM_RECURSE**.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Menus Overview

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

WM_NEXTMENU message

Article • 12/11/2020

Sent to an application when the right or left arrow key is used to switch between the menu bar and the system menu.

```
C++

#define WM_NEXTMENU                     0x0213
```

Parameters

wParam

The virtual-key code of the key. See **Virtual-Key Codes**.

lParam

A pointer to a **MDINEXTMENU** structure that contains information about the menu to be activated.

Remarks

In responding to this message, the application can specify the menu to switch to in the **hmenuNext** member of **MDINEXTMENU** and the window to receive the menu notification messages in the **hwndNext** member of the **MDINEXTMENU** structure. You must set both members for the changes to take effect (they are initially **NULL**).

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Reference

MDINEXTMENU

Conceptual

Menus

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

WM_UNINITMENUPOPUP message

Article • 12/11/2020

Sent when a drop-down menu or submenu has been destroyed.

```
C++

#define WM_UNINITMENUPOPUP              0x0125
```

Parameters

wParam

A handle to the menu

lParam

The high-order word identifies the menu that was destroyed. Currently, this parameter can only be **MF_SYSMENU** (the window menu).

Remarks

If an application receives a **WM_INITMENUPOPUP** message, it will receive a **WM_UNINITMENUPOPUP** message.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winuser.h (include Windows.h)

See also

Reference

HIWORD

Conceptual

Menus

Feedback

Was this page helpful? Yes No

Menu Structures

Article • 04/27/2021

In This Section

- MDINEXTMENU
- MENUBARINFO
- MENUEX_TEMPLATE_HEADER
- MENUEX_TEMPLATE_ITEM
- MENUGETOBJECTINFO
- MENUINFO
- MENUITEMINFO
- MENUITEMTEMPLATE
- MENUITEMTEMPLATEHEADER
- TPMPARAMS

MDINEXTMENU structure (winuser.h)

Article02/22/2024

Contains information about the menu to be activated.

Syntax

```
C++
```

```
typedef struct tagMDINEXTMENU {
  HMENU hmenuIn;
  HMENU hmenuNext;
  HWND  hwndNext;
} MDINEXTMENU, *PMDINEXTMENU, *LPMDINEXTMENU;
```

Members

`hmenuIn`

Type: **HMENU**

A handle to the current menu.

`hmenuNext`

Type: **HMENU**

A handle to the menu to be activated.

`hwndNext`

Type: **HWND**

A handle to the window to receive the menu notification messages.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

Menus

Reference

WM_NEXTMENU

Feedback

MENUBARINFO structure (winuser.h)

Article04/02/2021

Contains menu bar information.

Syntax

```
C++
```

```
typedef struct tagMENUBARINFO {
  DWORD cbSize;
  RECT  rcBar;
  HMENU hMenu;
  HWND  hwndMenu;
  BOOL  fBarFocused : 1;
  BOOL  fFocused : 1;
  BOOL  fUnused : 30;
} MENUBARINFO, *PMENUBARINFO, *LPMENUBARINFO;
```

Members

`cbSize`

Type: **DWORD**

The size of the structure, in bytes. The caller must set this to `sizeof(MENUBARINFO)`.

`rcBar`

Type: **RECT**

The coordinates of the menu bar, popup menu, or menu item.

`hMenu`

Type: **HMENU**

A handle to the menu bar or popup menu.

`hwndMenu`

Type: **HWND**

A handle to the submenu.

`fBarFocused`

Type: **BOOL**

If the menu bar or popup menu has the focus, this member is **TRUE**. Otherwise, the member is **FALSE**.

`fFocused`

Type: **BOOL**

If the menu item has the focus, this member is **TRUE**. Otherwise, the member is **FALSE**.

`fUnused`

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

GetMenuBarInfo

Menus

RECT

Reference

Feedback

Was this page helpful? Yes No

MENUEX_TEMPLATE_HEADER structure

Article • 12/11/2020

Defines the header for an extended menu template. This structure definition is for explanation only; it is not present in any standard header file.

Syntax

```
C++
```

```
typedef struct {
  WORD  wVersion;
  WORD  wOffset;
  DWORD dwHelpId;
} MENUEX_TEMPLATE_HEADER;
```

Members

wVersion

Type: **WORD**

The template version number. This member must be 1 for extended menu templates.

wOffset

Type: **WORD**

The offset to the first MENUEX_TEMPLATE_ITEM structure, relative to the end of this structure member. If the first item definition immediately follows the **dwHelpId** member, this member should be 4.

dwHelpId

Type: **DWORD**

The help identifier of menu bar.

Remarks

An extended menu template consists of a **MENUEX_TEMPLATE_HEADER** structure followed by one or more contiguous MENUEX_TEMPLATE_ITEM structures. The

MENUEX_TEMPLATE_ITEM structures, which are variable in length, are aligned on **DWORD** boundaries. To create a menu from an extended menu template in memory, use the **LoadMenuIndirect** function.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

LoadMenuIndirect

MENUEX_TEMPLATE_ITEM

Conceptual

Menus

Feedback

MENUEX_TEMPLATE_ITEM structure

Article • 12/11/2020

Defines a menu item in an extended menu template. This structure definition is for explanation only; it is not present in any standard header file.

Syntax

```
C++
```

```
typedef struct {
  DWORD dwType;
  DWORD dwState;
  UINT  uId;
  WORD  wFlags;
  WCHAR szText[1];
} MENUEX_TEMPLATE_ITEM;
```

Members

dwType

Type: **DWORD**

The menu item type. This member can be a combination of the type (beginning with MFT) values listed with the MENUITEMINFO structure.

dwState

Type: **DWORD**

The menu item state. This member can be a combination of the state (beginning with MFS) values listed with the MENUITEMINFO structure.

uId

Type: **UINT**

The menu item identifier. This is an application-defined value that identifies the menu item. In an extended menu resource, items that open drop-down menus or submenus as well as command items can have identifiers.

wFlags

Type: **WORD**

Specifies whether the menu item is the last item in the menu bar, drop-down menu, submenu, or shortcut menu and whether it is an item that opens a drop-down menu or submenu. This member can be zero or more of these values. For 32-bit applications, this member is a word; for 16-bit applications, it is a byte.

0x80

The structure defines the last menu item in the menu bar, drop-down menu, submenu, or shortcut menu.

0x01

The structure defines a item that opens a drop-down menu or submenu. Subsequent structures define menu items in the corresponding drop-down menu or submenu.

szText

Type: **WCHAR**

The menu item text. This member is a null-terminated Unicode string, aligned on a word boundary. The size of the menu item definition varies depending on the length of this string.

Remarks

An extended menu template consists of a MENUEX_TEMPLATE_HEADER structure followed by one or more contiguous **MENUEX_TEMPLATE_ITEM** structures. The **MENUEX_TEMPLATE_ITEM** structures, which are variable in length, are aligned on **DWORD** boundaries. To create a menu from an extended menu template in memory, use the LoadMenuIndirect function.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

LoadMenuIndirect

MENUEX_TEMPLATE_HEADER

MENUITEMINFO

Conceptual

Menus

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

MENUGETOBJECTINFO structure (winuser.h)

Article02/22/2024

Contains information about the menu that the mouse cursor is on.

Syntax

C++

```
typedef struct tagMENUGETOBJECTINFO {
  DWORD dwFlags;
  UINT  uPos;
  HMENU hmenu;
  PVOID riid;
  PVOID pvObj;
} MENUGETOBJECTINFO, *PMENUGETOBJECTINFO;
```

Members

`dwFlags`

Type: **DWORD**

The position of the mouse cursor with respect to the item indicated by **uPos**. It is a bitmask of the following values:

Expand table

Value	Meaning
MNGOF_BOTTOMGAP 0x00000002	The mouse is on the bottom of the item indicated by **uPos**.
MNGOF_TOPGAP 0x00000001	The mouse is on the top of the item indicated by **uPos**.

If neither MNGOF_BOTTOMGAP nor MNGOF_TOPGAP is set, then the mouse is directly on the item indicated by **uPos**.

`uPos`

Type: **UINT**

The position of the item the mouse cursor is on.

`hmenu`

Type: **HMENU**

A handle to the menu the mouse cursor is on.

`riid`

Type: **PVOID**

The identifier of the requested interface. Currently it can only be IDropTarget.

`pvObj`

Type: **PVOID**

A pointer to the interface corresponding to the **riid** member. This pointer is to be returned by the application when processing the message.

Remarks

The **MENUGETOBJECTINFO** structure is used only in drag-and-drop menus. When the WM_MENUGETOBJECT message is sent, *lParam* is a pointer to this structure.

To create a drag-and-drop menu, call SetMenuInfo with **MNS_DRAGDROP** set.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MENUINFO structure (winuser.h)

Article04/02/2021

Contains information about a menu.

Syntax

C++

```
typedef struct tagMENUINFO {
  DWORD      cbSize;
  DWORD      fMask;
  DWORD      dwStyle;
  UINT       cyMax;
  HBRUSH     hbrBack;
  DWORD      dwContextHelpID;
  ULONG_PTR dwMenuData;
} MENUINFO, *LPMENUINFO;
```

Members

`cbSize`

Type: **DWORD**

The size of the structure, in bytes. The caller must set this member to `sizeof(MENUINFO)`.

`fMask`

Type: **DWORD**

Indicates the members to be retrieved or set (except for **MIM_APPLYTOSUBMENUS**). This member can be one or more of the following values.

Expand table

Value	Meaning
MIM_APPLYTOSUBMENUS 0x80000000	Settings apply to the menu and all of its submenus. SetMenuInfo uses this flag and GetMenuInfo ignores this flag
MIM_BACKGROUND 0x00000002	Retrieves or sets the **hbrBack** member.

MIM_HELPID 0x00000004	Retrieves or sets the **dwContextHelpID** member.
MIM_MAXHEIGHT 0x00000001	Retrieves or sets the **cyMax** member.
MIM_MENUDATA 0x00000008	Retrieves or sets the **dwMenuData** member.
MIM_STYLE 0x00000010	Retrieves or sets the **dwStyle** member.

`dwStyle`

Type: **DWORD**

The menu style. This member can be one or more of the following values.

Expand table

Value	Meaning
MNS_AUTODISMISS 0x10000000	Menu automatically ends when mouse is outside the menu for approximately 10 seconds.
MNS_CHECKORBMP 0x04000000	The same space is reserved for the check mark and the bitmap. If the check mark is drawn, the bitmap is not. All checkmarks and bitmaps are aligned. Used for menus where some items use checkmarks and some use bitmaps.
MNS_DRAGDROP 0x20000000	Menu items are OLE drop targets or drag sources. Menu owner receives WM_MENUDRAG and WM_MENUGETOBJECT messages.
MNS_MODELESS 0x40000000	Menu is modeless; that is, there is no menu modal message loop while the menu is active.
MNS_NOCHECK 0x80000000	No space is reserved to the left of an item for a check mark. The item can still be selected, but the check mark will not appear next to the item.
MNS_NOTIFYBYPOS 0x08000000	Menu owner receives a WM_MENUCOMMAND message instead of a WM_COMMAND message when the user makes a selection. **MNS_NOTIFYBYPOS** is a menu header style and has no effect when applied to individual sub menus.

`cyMax`

Type: **UINT**

The maximum height of the menu in pixels. When the menu items exceed the space available, scroll bars are automatically used. The default (0) is the screen height.

`hbrBack`

Type: **HBRUSH**

A handle to the brush to be used for the menu's background.

`dwContextHelpID`

Type: **DWORD**

The context help identifier. This is the same value used in the GetMenuContextHelpId and SetMenuContextHelpId functions.

`dwMenuData`

Type: **ULONG_PTR**

An application-defined value.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Menus Overview

Feedback

Was this page helpful?

MENUITEMINFOA structure (winuser.h)

Article07/27/2022

Contains information about a menu item.

Syntax

C++

```
typedef struct tagMENUITEMINFOA {
  UINT      cbSize;
  UINT      fMask;
  UINT      fType;
  UINT      fState;
  UINT      wID;
  HMENU     hSubMenu;
  HBITMAP   hbmpChecked;
  HBITMAP   hbmpUnchecked;
  ULONG_PTR dwItemData;
  LPSTR     dwTypeData;
  UINT      cch;
  HBITMAP   hbmpItem;
} MENUITEMINFOA, *LPMENUITEMINFOA;
```

Members

`cbSize`

Type: **UINT**

The size of the structure, in bytes. The caller must set this member to `sizeof(MENUITEMINFO)`.

`fMask`

Type: **UINT**

Indicates the members to be retrieved or set. This member can be one or more of the following values.

Expand table

Value	Meaning

MIIM_BITMAP 0x00000080	Retrieves or sets the **hbmpItem** member.
MIIM_CHECKMARKS 0x00000008	Retrieves or sets the **hbmpChecked** and **hbmpUnchecked** members.
MIIM_DATA 0x00000020	Retrieves or sets the **dwItemData** member.
MIIM_FTYPE 0x00000100	Retrieves or sets the **fType** member.
MIIM_ID 0x00000002	Retrieves or sets the **wID** member.
MIIM_STATE 0x00000001	Retrieves or sets the **fState** member.
MIIM_STRING 0x00000040	Retrieves or sets the **dwTypeData** member.
MIIM_SUBMENU 0x00000004	Retrieves or sets the **hSubMenu** member.
MIIM_TYPE 0x00000010	Retrieves or sets the **fType** and **dwTypeData** members. **MIIM_TYPE** is replaced by **MIIM_BITMAP**, **MIIM_FTYPE**, and **MIIM_STRING**.

`fType`

Type: **UINT**

The menu item type. This member can be one or more of the following values.

The **MFT_BITMAP**, **MFT_SEPARATOR**, and **MFT_STRING** values cannot be combined with one another. Set **fMask** to **MIIM_TYPE** to use **fType**.

fType is used only if **fMask** has a value of **MIIM_FTYPE**.

Expand table

Value	Meaning
MFT_BITMAP 0x00000004L	Displays the menu item using a bitmap. The low-order word of the **dwTypeData** member is the bitmap handle, and the **cch** member is ignored. **MFT_BITMAP** is replaced by **MIIM_BITMAP** and **hbmpItem**.

MFT_MENUBARBREAK 0x00000020L	Places the menu item on a new line (for a menu bar) or in a new column (for a drop-down menu, submenu, or shortcut menu). For a drop-down menu, submenu, or shortcut menu, a vertical line separates the new column from the old.
MFT_MENUBREAK 0x00000040L	Places the menu item on a new line (for a menu bar) or in a new column (for a drop-down menu, submenu, or shortcut menu). For a drop-down menu, submenu, or shortcut menu, the columns are not separated by a vertical line.
MFT_OWNERDRAW 0x00000100L	Assigns responsibility for drawing the menu item to the window that owns the menu. The window receives a WM_MEASUREITEM message before the menu is displayed for the first time, and a WM_DRAWITEM message whenever the appearance of the menu item must be updated. If this value is specified, the **dwTypeData** member contains an application-defined value.
MFT_RADIOCHECK 0x00000200L	Displays selected menu items using a radio-button mark instead of a check mark if the **hbmpChecked** member is **NULL**.
MFT_RIGHTJUSTIFY 0x00004000L	Right-justifies the menu item and any subsequent items. This value is valid only if the menu item is in a menu bar.
MFT_RIGHTORDER 0x00002000L	Specifies that menus cascade right-to-left (the default is left-to-right). This is used to support right-to-left languages, such as Arabic and Hebrew.
MFT_SEPARATOR 0x00000800L	Specifies that the menu item is a separator. A menu item separator appears as a horizontal dividing line. The **dwTypeData** and **cch** members are ignored. This value is valid only in a drop-down menu, submenu, or shortcut menu.
MFT_STRING 0x00000000L	Displays the menu item using a text string. The **dwTypeData** member is the pointer to a null-terminated string, and the **cch** member is the length of the string. **MFT_STRING** is replaced by **MIIM_STRING**.

`fState`

Type: **UINT**

The menu item state. This member can be one or more of these values. Set **fMask** to **MIIM_STATE** to use **fState**.

Expand table

Value	Meaning
MFS_CHECKED 0x00000008L	Checks the menu item. For more information about selected menu items, see the **hbmpChecked** member.
MFS_DEFAULT 0x00001000L	Specifies that the menu item is the default. A menu can contain only one default menu item, which is displayed in bold.
MFS_DISABLED 0x00000003L	Disables the menu item and grays it so that it cannot be selected. This is equivalent to **MFS_GRAYED**.
MFS_ENABLED 0x00000000L	Enables the menu item so that it can be selected. This is the default state.
MFS_GRAYED 0x00000003L	Disables the menu item and grays it so that it cannot be selected. This is equivalent to **MFS_DISABLED**.
MFS_HILITE 0x00000080L	Highlights the menu item.
MFS_UNCHECKED 0x00000000L	Unchecks the menu item. For more information about clear menu items, see the **hbmpChecked** member.
MFS_UNHILITE 0x00000000L	Removes the highlight from the menu item. This is the default state.

`wID`

Type: **UINT**

An application-defined value that identifies the menu item. Set **fMask** to **MIIM_ID** to use **wID**.

`hSubMenu`

Type: **HMENU**

A handle to the drop-down menu or submenu associated with the menu item. If the menu item is not an item that opens a drop-down menu or submenu, this member is **NULL**. Set **fMask** to **MIIM_SUBMENU** to use **hSubMenu**.

`hbmpChecked`

Type: **HBITMAP**

A handle to the bitmap to display next to the item if it is selected. If this member is **NULL**, a default bitmap is used. If the **MFT_RADIOCHECK** type value is specified, the default bitmap is a bullet. Otherwise, it is a check mark. Set **fMask** to **MIIM_CHECKMARKS** to use **hbmpChecked**.

`hbmpUnchecked`

Type: **HBITMAP**

A handle to the bitmap to display next to the item if it is not selected. If this member is **NULL**, no bitmap is used. Set **fMask** to **MIIM_CHECKMARKS** to use **hbmpUnchecked**.

`dwItemData`

Type: **ULONG_PTR**

An application-defined value associated with the menu item. Set **fMask** to **MIIM_DATA** to use **dwItemData**.

`dwTypeData`

Type: **LPTSTR**

The contents of the menu item. The meaning of this member depends on the value of **fType** and is used only if the **MIIM_TYPE** flag is set in the **fMask** member.

To retrieve a menu item of type **MFT_STRING**, first find the size of the string by setting the **dwTypeData** member of **MENUITEMINFO** to **NULL** and then calling GetMenuItemInfo. The value of **cch**+1 is the size needed. Then allocate a buffer of this size, place the pointer to the buffer in **dwTypeData**, increment **cch**, and call **GetMenuItemInfo** once again to fill the buffer with the string. If the retrieved menu item is of some other type, then **GetMenuItemInfo** sets the **dwTypeData** member to a value whose type is specified by the **fType** member.

When using with the SetMenuItemInfo function, this member should contain a value whose type is specified by the **fType** member.

dwTypeData is used only if the **MIIM_STRING** flag is set in the **fMask** member

`cch`

Type: **UINT**

The length of the menu item text, in characters, when information is received about a menu item of the **MFT_STRING** type. However, **cch** is used only if the **MIIM_TYPE** flag is

set in the **fMask** member and is zero otherwise. Also, **cch** is ignored when the content of a menu item is set by calling SetMenuItemInfo.

Note that, before calling GetMenuItemInfo, the application must set **cch** to the length of the buffer pointed to by the **dwTypeData** member. If the retrieved menu item is of type **MFT_STRING** (as indicated by the **fType** member), then **GetMenuItemInfo** changes **cch** to the length of the menu item text. If the retrieved menu item is of some other type, **GetMenuItemInfo** sets the **cch** field to zero.

The **cch** member is used when the **MIIM_STRING** flag is set in the **fMask** member.

`hbmpItem`

Type: **HBITMAP**

A handle to the bitmap to be displayed, or it can be one of the values in the following table. It is used when the **MIIM_BITMAP** flag is set in the **fMask** member.

Expand table

Value	Meaning
HBMMENU_CALLBACK ((HBITMAP) -1)	A bitmap that is drawn by the window that owns the menu. The application must process the WM_MEASUREITEM and WM_DRAWITEM messages.
HBMMENU_MBAR_CLOSE ((HBITMAP) 5)	Close button for the menu bar.
HBMMENU_MBAR_CLOSE_D ((HBITMAP) 6)	Disabled close button for the menu bar.
HBMMENU_MBAR_MINIMIZE ((HBITMAP) 3)	Minimize button for the menu bar.
HBMMENU_MBAR_MINIMIZE_D ((HBITMAP) 7)	Disabled minimize button for the menu bar.
HBMMENU_MBAR_RESTORE ((HBITMAP) 2)	Restore button for the menu bar.
HBMMENU_POPUP_CLOSE ((HBITMAP) 8)	Close button for the submenu.
HBMMENU_POPUP_MAXIMIZE ((HBITMAP) 10)	Maximize button for the submenu.
HBMMENU_POPUP_MINIMIZE ((HBITMAP) 11)	Minimize button for the submenu.

HBMMENU_POPUP_RESTORE ((HBITMAP) 9)	Restore button for the submenu.
HBMMENU_SYSTEM ((HBITMAP) 1)	Windows icon or the icon of the window specified in **dwItemData**.

Remarks

The **MENUITEMINFO** structure is used with the GetMenuItemInfo, InsertMenuItem, and SetMenuItemInfo functions.

The menu can display items using text, bitmaps, or both.

> **Note**
>
> The winuser.h header defines MENUITEMINFO as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

GetMenuItemInfo

InsertMenuItem

Menus

Reference

SetMenuItemInfo

WM_DRAWITEM

WM_MEASUREITEM

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MENUITEMTEMPLATE structure (winuser.h)

Article02/22/2024

Defines a menu item in a menu template.

Syntax

C++

```
typedef struct {
  WORD  mtOption;
  WORD  mtID;
  WCHAR mtString[1];
} MENUITEMTEMPLATE, *PMENUITEMTEMPLATE;
```

Members

`mtOption`

Type: **WORD**

One or more of the following predefined menu options that control the appearance of the menu item as shown in the following table.

Expand table

Value	Meaning
MF_CHECKED 0x00000008L	Indicates that the menu item has a check mark next to it.
MF_GRAYED 0x00000001L	Indicates that the menu item is initially inactive and drawn with a gray effect.
MF_HELP 0x00004000L	Indicates that the menu item has a vertical separator to its left.
MF_MENUBARBREAK 0x00000020L	Indicates that the menu item is placed in a new column. The old and new columns are separated by a bar.
MF_MENUBREAK 0x00000040L	Indicates that the menu item is placed in a new column.

MF_OWNERDRAW 0x00000100L	Indicates that the owner window of the menu is responsible for drawing all visual aspects of the menu item, including highlighted, selected, and inactive states. This option is not valid for an item in a menu bar.
MF_POPUP 0x00000010L	Indicates that the item is one that opens a drop-down menu or submenu.

`mtID`

Type: **WORD**

The menu item identifier of a command item; a command item sends a command message to its owner window. The **MENUITEMTEMPLATE** structure for an item that opens a drop-down menu or submenu does not contain the **mtID** member.

`mtString[1]`

Type: **WCHAR[1]**

The menu item.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

LoadMenuIndirect

MENUITEMTEMPLATEHEADER

Menus

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MENUITEMTEMPLATEHEADER structure (winuser.h)

Article02/22/2024

Defines the header for a menu template. A complete menu template consists of a header and one or more menu item lists.

Syntax

```
C++
```

```
typedef struct {
  WORD versionNumber;
  WORD offset;
} MENUITEMTEMPLATEHEADER, *PMENUITEMTEMPLATEHEADER;
```

Members

`versionNumber`

Type: **WORD**

The version number. This member must be zero.

`offset`

Type: **WORD**

The offset, in bytes, from the end of the header. The menu item list begins at this offset. Usually, this member is zero, and the menu item list follows immediately after the header.

Remarks

One or more MENUITEMTEMPLATE structures are combined to form the menu item list.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

LoadMenuIndirect

MENUITEMTEMPLATE

Menus

Reference

Feedback

TPMPARAMS structure (winuser.h)

Article02/22/2024

Contains extended parameters for the TrackPopupMenuEx function.

Syntax

```
C++
```

```
typedef struct tagTPMPARAMS {
  UINT cbSize;
  RECT rcExclude;
} TPMPARAMS;
```

Members

`cbSize`

Type: **UINT**

The size of structure, in bytes.

`rcExclude`

Type: **RECT**

The rectangle to be excluded when positioning the window, in screen coordinates.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

Menus

Reference

TrackPopupMenuEx

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Strings

Article • 08/19/2021

This section describes the string functions and explains how to use them in your applications.

In This Section

Name	Description
About Strings	Discusses the string functions.
About Strsafe.h	Discusses the string functions in Strsafe.h.
String Reference	Contains the API reference.

String Functions

Name	Description
CharLower	Converts a character string or a single character to lowercase. If the operand is a character string, the function converts the characters in place.
CharLowerBuff	Converts uppercase characters in a buffer to lowercase characters. The function converts the characters in place.
CharNext	Retrieves a pointer to the next character in a string. This function can handle strings consisting of either single- or multi-byte characters.
CharNextExA	Retrieves the pointer to the next character in a string. This function can handle strings consisting of either single- or multi-byte characters.
CharPrev	Retrieves a pointer to the preceding character in a string. This function can handle strings consisting of either single- or multi-byte characters.
CharPrevExA	Retrieves the pointer to the preceding character in a string. This function can handle strings consisting of either single- or multi-byte characters.
CharToOem	Translates a string into the OEM-defined character set.
CharToOemBuff	Translates a specified number of characters in a string into the OEM-defined character set.
CharUpper	Converts a character string or a single character to uppercase. If the operand is a character string, the function converts the characters in place.

Name	Description
CharUpperBuff	Converts lowercase characters in a buffer to uppercase characters. The function converts the characters in place.
CompareString	Compares two character strings, using the specified locale. **Note:** For compatibility with Unicode, use CompareStringEx or the Unicode version of CompareString.
CompareStringEx	Compares two Unicode (wide character) strings, using the specified locale.
FoldString	Maps one string to another, performing a specified transformation option.
GetStringTypeA	Retrieves character-type information for the characters in the specified source string. For each character in the string, the function sets one or more bits in the corresponding 16-bit element of the output array. Each bit identifies a given character type, such as whether the character is a letter, a digit, or neither.
GetStringTypeEx	Retrieves character-type information for the characters in the specified source string. For each character in the string, the function sets one or more bits in the corresponding 16-bit element of the output array. Each bit identifies a given character type, such as whether the character is a letter, a digit, or neither. Unlike its close relatives GetStringTypeA and GetStringTypeW, GetStringTypeEx exhibits standard behavior through the use of the **#define UNICODE** switch. It is the recommended function.
GetStringTypeW	Retrieves character-type information for the characters in the specified source string. For each character in the string, the function sets one or more bits in the corresponding 16-bit element of the output array. Each bit identifies a given character type, such as whether the character is a letter, a digit, or neither.
IsCharAlpha	Determines whether a character is an alphabetical character. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.
IsCharAlphaNumeric	Determines whether a character is either an alphabetical or a numeric character. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.
IsCharLower	Determines whether a character is lowercase. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.
IsCharUpper	Determines whether a character is uppercase. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

Name	Description
LoadString	Loads a string resource from the executable file associated with a specified module, copies the string into a buffer, and appends a terminating NULL character.
lstrcat	Appends one string to another.
lstrcmp	Compares two character strings. The comparison is case-sensitive.
lstrcmpi	Compares two character strings. The comparison is not case-sensitive.
lstrcpy	Copies a string to a buffer.
lstrcpyn	Copies a specified number of characters from a source string into a buffer.
lstrlen	Determines the length of the specified string (not including the terminating null character).
OemToChar	Translates a string from the OEM-defined character set into either an ANSI or a wide-character string.
OemToCharBuff	Translates a specified number of characters in a string from the OEM-defined character set into either an ANSI or a wide-character string.
wsprintf	Writes formatted data to the specified buffer.
wvsprintf	Writes formatted data to the specified buffer using a pointer to a list of arguments.

Strsafe Functions

Name	Description
StringCbCat	Concatenates one string to another string.
StringCbCatEx	Concatenates one string to another string.
StringCbCatN	Concatenates the specified number of bytes from one string to another string.
StringCbCatNEx	Concatenates the specified number of bytes from one string to another string.
StringCbCopy	Copies one string to another.
StringCbCopyEx	Copies one string to another.
StringCbCopyN	Copies the specified number of bytes from one string to another.
StringCbCopyNEx	Copies the specified number of bytes from one string to another.

Name	Description
StringCbGets	Gets one line of text from stdin, up to and including the newline character ('\n').
StringCbGetsEx	Gets one line of text from stdin, up to and including the newline character ('\n').
StringCbLength	Determines whether a string exceeds the specified length, in bytes.
StringCbPrintf	Writes formatted data to the specified string.
StringCbPrintfEx	Writes formatted data to the specified string.
StringCbVPrintf	Writes formatted data to the specified string using a pointer to a list of arguments.
StringCbVPrintfEx	Writes formatted data to the specified string using a pointer to a list of arguments.
StringCchCat	Concatenates one string to another string.
StringCchCatEx	Concatenates one string to another string.
StringCchCatN	Concatenates the specified number of characters from one string to another string.
StringCchCatNEx	Concatenates the specified number of characters from one string to another string.
StringCchCopy	Copies one string to another.
StringCchCopyEx	Copies one string to another.
StringCchCopyN	Copies the specified number of characters from one string to another.
StringCchCopyNEx	Copies the specified number of characters from one string to another.
StringCchGets	Gets one line of text from stdin, up to and including the newline character ('\n').
StringCchGetsEx	Gets one line of text from stdin, up to and including the newline character ('\n').
StringCchLength	Determines whether a string exceeds the specified length, in characters.
StringCchPrintf	Writes formatted data to the specified string.
StringCchPrintfEx	Writes formatted data to the specified string.
StringCchVPrintf	Writes formatted data to the specified string using a pointer to a list of arguments.

Name	Description
StringCchVPrintfEx	Writes formatted data to the specified string using a pointer to a list of arguments.

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

About Strings

Article • 06/13/2022

The string functions give applications the means to copy, compare, sort, format, and convert character strings as well as the means to determine the character type of each character in a string. All the string functions support the single-byte, double-byte, and Unicode character sets if these character sets are supported by the operating system on which the application is run.

Security Warning: The incorrect use of string functions can cause security problems for your application. Typically this involves a buffer overrun which can allow a denial of service attack against your application or the injection of executable code from an attacker. The Strsafe functions enable the safer handling of strings and are recommended for better security for your application. For more information on these functions, see Using the Strsafe.h Functions.

This section discusses the following topics.

- Comparison with C Run-Time String Functions
- String Resources

Comparison with C Run-Time String Functions

Many string functions duplicate or enhance familiar string functions from the standard C run-time (CRT) library. Many of the enhancements enable the string functions to work with Unicode or extended character sets. The following table shows the CRT functions, the Windows functions (that support Unicode, unlike the CRT functions), and the StrSafe functions.

CRT String Function	Windows String Function	StrSafe Function
strcat	lstrcat	StringCchCat StringCchCatEx StringCbCat StringCbCatEx
strcmp	lstrcmp	(no equivalent function)
strcpy	lstrcpy	StringCchCopy StringCchCopyEx StringCbCopy StringCbCopyEx

CRT String Function	Windows String Function	StrSafe Function
strlen	lstrlen	StringCchLength StringCbLength

The **strlen** function, for example, always returns the number of bytes in a string, but the lstrlen function returns the number of **TCHAR** values, which refers to bytes for ANSI versions of the function or **WCHAR** values for Unicode versions.

The following string functions differ from standard C functions such as **tolower** and **toupper** in that they operate on any character in a character set. By using the CharLower function, for example, an application can convert an uppercase U with an umlaut (Ü) to lowercase (ü). For more information about character sets, see Single-byte Character Sets.

Function	Description
CharLower	Converts a character or string to lowercase.
CharLowerBuff	Converts a character string to lowercase.
CharNext	Moves to the next character in a string.
CharPrev	Moves to the preceding character in a string.
CharUpper	Converts a character or string to uppercase.
CharUpperBuff	Converts a string to uppercase.

The following string functions make determinations about a character based on the semantics of the language selected by the user. These functions are Unicode enabled.

Function	Description
IsCharAlpha	Determines whether a character is alphabetic.
IsCharAlphaNumeric	Determines whether a character is alphanumeric.
IsCharLower	Determines whether a character is lowercase.
IsCharUpper	Determines whether a character is uppercase.

The following table shows the Unicode extensions to the standard C run-time (CRT) functions. As mentioned previously, the StrSafe functions enable safer handling of strings and are recommended for better security for your application.

Standard CRT function	String Function	StrSafe Function
sprintf	wsprintf	StringCchPrintf StringCchPrintfEx StringCbPrintf StringCbPrintfEx
vsprintf	wvsprintf	StringCchVPrintf StringCchVPrintfEx StringCbVPrintf StringCbVPrintfEx

String Resources

An application that maintains character strings in resources can be translated into new languages with minimum effort. Instead of searching for strings in the source modules, you can simply translate the strings in the resource file and relink the application. In addition, using string resources simplifies creation of Unicode and non-Unicode versions of the application from the same source files.

The LoadString function loads a string resource from an application's executable file. The FormatMessage function loads a string resource and interprets formatting options that may be embedded in the string.

Resources in binary form are stored in Unicode format. When loading resources, applications can use the Unicode version of the resource functions (LoadStringW, for example) to obtain resources as Unicode data.

For 16-bit string resources, 255 characters is the maximum length. For 32-bit string resources, 65535 characters is the maximum length.

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

About Strsafe.h

Article • 06/20/2022

Poor buffer handling is implicated in many security issues that involve buffer overruns. The functions defined in Strsafe.h provide additional processing for proper buffer handling in your code. For this reason, they are intended to replace their built-in C/C++ counterparts as well as specific Windows implementations. Strsafe.h is available in the Windows SDK starting with Windows XP with Service Pack 2 (SP2).

The advantages of the Strsafe functions include:

- The size of the destination buffer is always provided to the function to ensure that the function does not write past the end of the buffer.
- Buffers are guaranteed to be null-terminated, even if the operation truncates the intended result.
- All functions return an **HRESULT** value, with only one possible success code (**S_OK**).
- Each function is available in a corresponding character count ("cch") or byte count ("cb") version.
- Most functions have an extended ("Ex") version available for advanced functionality.

See the following sections for details.

Character Count Functions

The following functions use a character count rather than a byte count.

Expand table

Function	Replaces
StringCchCat **StringCchCatEx**	strcat, wcscat, _tcscat **lstrcat** **StrCat**

Function	Replaces
	StrCatBuff
StringCchCatN **StringCchCatNEx**	strncat **StrNCat**
StringCchCopy **StringCchCopyEx**	strcpy, wcscpy, _tcscpy **lstrcpy** **StrCpy**
StringCchCopyN **StringCchCopyNEx**	strncpy, wcsncpy, _tcsncpy
StringCchGets **StringCchGetsEx**	gets, _getws, _getts
StringCchPrintf **StringCchPrintfEx**	sprintf, swprintf, _stprintf **wsprintf** **wnsprintf** _snprintf, _snwprintf, _sntprintf
StringCchVPrintf **StringCchVPrintfEx**	vsprintf, vswprintf, _vstprintf vsnprintf, _vsnwprintf, _vsntprintf **wvsprintf** **wvnsprintf** ,
StringCchLength	strlen, wcslen, _tcslen

Byte Count Functions

The following functions use a byte count rather than a character count.

Expand table

Function	Replaces
StringCbCat **StringCbCatEx**	strcat, wcscat, _tcscat **lstrcat** **StrCat** **StrCatBuff**
StringCbCatN **StringCbCatNEx**	strncat **StrNCat**
StringCbCopy	strcpy, wcscpy, _tcscpy

Function	Replaces
StringCbCopyEx	lstrcpy StrCpy
StringCbCopyN StringCbCopyNEx	strncpy, wcsncpy, _tcsncpy
StringCbGets StringCbGetsEx	gets, _getws, _getts
StringCbPrintf StringCbPrintfEx	sprintf, swprintf, _stprintf wsprintf wnsprintf _snprintf, _snwprintf, _sntprintf
StringCbVPrintf StringCbVPrintfEx	vsprintf, vswprintf, _vstprintf vsnprintf, _vsnwprintf, _vsntprintf wvsprintf wvnsprintf
StringCbLength	strlen, wcslen, _tcslen

Using Strsafe.h

- To use the Strsafe functions inline, include the header file as shown here, following the **#include** statements for all other header files.

  ```
  #include <strsafe.h>
  ```

- To use the functions in library form, include the following statement before including Strsafe.h. However, it is recommended that you use the inline functions.

  ```
  #define STRSAFE_LIB
  ```

  > ⓘ Note
  >
  > : The following functions must be used as inline functions: StringCbGets, StringCbGetsEx, StringCchGets, and StringCchGetsEx.

- When you include Strsafe.h in your file, the older functions replaced by the Strsafe.h functions will be deprecated. Attempts to use these older functions will

result in a compiler error telling you to use the newer functions. If you want to override this behavior, include the following statement before including Strsafe.h.

```
#define STRSAFE_NO_DEPRECATE
```

- To allow only character count functions, include the following statement before including Strsafe.h.

  ```
  #define STRSAFE_NO_CB_FUNCTIONS
  ```

- To allow only byte count functions, include the following statement before including Strsafe.h.

  ```
  #define STRSAFE_NO_CCH_FUNCTIONS
  ```

  > **Note**
  >
  > You can define **STRSAFE_NO_CB_FUNCTIONS** or **STRSAFE_NO_CCH_FUNCTIONS**, but not both.

- Some Strsafe functions have locale-aware versions. By default, the header does not declare these functions. To enable these declarations, include the following macro statement before including Strsafe.h.

  ```
  #define STRSAFE_LOCALE_FUNCTIONS
  ```

- The maximum supported string length is 2,147,483,647 (**STRSAFE_MAX_CCH**) characters, either ANSI or Unicode.

Related topics

Strsafe Functions

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

String Reference

Article • 04/27/2021

In This Section

- String Functions
- Strsafe Functions

Feedback

Was this page helpful? Yes No

String Functions

Article • 04/27/2021

In This Section

- CharLower
- CharLowerBuff
- CharNext
- CharNextExA
- CharPrev
- CharPrevExA
- CharToOem
- CharToOemBuff
- CharUpper
- CharUpperBuff
- IsCharAlpha
- IsCharAlphaNumeric
- IsCharLower
- IsCharUpper
- LoadString
- lstrcat
- lstrcmp
- lstrcmpi
- lstrcpy
- lstrcpyn
- lstrlen
- OemToChar
- OemToCharBuff
- wsprintf
- wvsprintf

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

CharLowerA function (winuser.h)

Article02/22/2024

Converts a character string or a single character to lowercase. If the operand is a character string, the function converts the characters in place.

Syntax

```
C++

LPSTR CharLowerA(
  [in, out] LPSTR lpsz
);
```

Parameters

`[in, out] lpsz`

Type: **LPTSTR**

A null-terminated string, or specifies a single character. If the high-order word of this parameter is zero, the low-order word must contain a single character to be converted.

Return value

Type: **LPTSTR**

If the operand is a character string, the function returns a pointer to the converted string. Because the string is converted in place, the return value is equal to *lpsz*.

If the operand is a single character, the return value is a 32-bit value whose high-order word is zero, and low-order word contains the converted character.

There is no indication of success or failure. Failure is rare. There is no extended error information for this function; do not call GetLastError.

Remarks

Note that **CharLower** always maps uppercase I to lowercase I ("i"), even when the current language is Turkish or Azerbaijani. If you need a function that is linguistically

sensitive in this respect, call LCMapString.

Conversion to Unicode in the ANSI version of the function is done with the system default locale in all cases.

> **Note**
>
> The winuser.h header defines CharLower as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

CharLowerBuff

CharUpper

CharUpperBuff

Conceptual

Reference

Strings

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CharLowerBuffA function (winuser.h)

Article02/22/2024

Converts uppercase characters in a buffer to lowercase characters. The function converts the characters in place.

Syntax

```
C++
```

```
DWORD CharLowerBuffA(
  [in, out] LPSTR lpsz,
  [in]      DWORD cchLength
);
```

Parameters

`[in, out] lpsz`

Type: **LPTSTR**

A buffer containing one or more characters to be processed.

`[in] cchLength`

Type: **DWORD**

The size, in characters, of the buffer pointed to by *lpsz*. The function examines each character, and converts uppercase characters to lowercase characters. The function examines the number of characters indicated by *cchLength*, even if one or more characters are null characters.

Return value

Type: **DWORD**

The return value is the number of characters processed. For example, if **CharLowerBuff**("Acme of Operating Systems", 10) succeeds, the return value is 10.

Remarks

Note that **CharLowerBuff** always maps uppercase I to lowercase I ("i"), even when the current language is Turkish or Azerbaijani. If you need a function that is linguistically sensitive in this respect, call LCMapSting.

Conversion to Unicode in the ANSI version of the function is done with the system default locale in all cases.

Examples

For an example, see "Creating a Spell Dialog Box" in Using Combo Boxes.

> **Note**
>
> The winuser.h header defines CharLowerBuff as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

CharLower

CharUpper

CharUpperBuff

Conceptual

Reference

Strings

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CharNextA function (winuser.h)

Article02/22/2024

Retrieves a pointer to the next character in a string. This function can handle strings consisting of either single- or multi-byte characters.

Syntax

```
C++

LPSTR CharNextA(
  [in] LPCSTR lpsz
);
```

Parameters

`[in] lpsz`

Type: **LPCTSTR**

A character in a null-terminated string.

Return value

Type: **LPTSTR**

The return value is a pointer to the next character in the string, or to the terminating null character if at the end of the string.

If *lpsz* points to the terminating null character, the return value is equal to *lpsz*.

Remarks

When called as an ANSI function, **CharNext** uses the system default code-page, whereas CharNextExA specifies a code-page to use.

This function works with default "user" expectations of characters when dealing with diacritics. For example: A string that contains U+0061 U+030a "LATIN SMALL LETTER A" + COMBINING RING ABOVE" — which looks like "å", will advance two code points, not

one. A string that contains U+0061 U+0301 U+0302 U+0303 U+0304 — which looks like "a´^~¯", will advance five code points, not one, and so on.

> **Note**
>
> The winuser.h header defines CharNext as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

CharNextExA

CharPrev

Conceptual

Reference

Strings

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CharNextExA function (winuser.h)

Article02/22/2024

Retrieves the pointer to the next character in a string. This function can handle strings consisting of either single- or multi-byte characters.

Syntax

```
C++

LPSTR CharNextExA(
  [in] WORD   CodePage,
  [in] LPCSTR lpCurrentChar,
  [in] DWORD  dwFlags
);
```

Parameters

`[in] CodePage`

Type: **WORD**

The identifier of the code page to use to check lead-byte ranges. Can be one of the code-page values provided in Code Page Identifiers, or one of the following predefined values.

Expand table

Value	Meaning
CP_ACP 0	Use system default ANSI code page.
CP_MACCP 2	Use the system default Macintosh code page.
CP_OEMCP 1	Use system default OEM code page.

`[in] lpCurrentChar`

Type: **LPCSTR**

A character in a null-terminated string.

`[in] dwFlags`

Type: **DWORD**

This parameter is reserved and must be 0.

Return value

Type: **LPSTR**

The return value is a pointer to the next character in the string, or to the terminating null character if at the end of the string.

If *lpCurrentChar* points to the terminating null character, the return value is equal to *lpCurrentChar*.

Remarks

CharNextExA specifies a code-page to use, whereas CharNext (if called as an ANSI function) uses the system default code-page.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

CharNext

CharPrevExA

Conceptual

Reference

Strings

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CharPrevA function (winuser.h)

Article02/22/2024

Retrieves a pointer to the preceding character in a string. This function can handle strings consisting of either single- or multi-byte characters.

Syntax

```
C++

LPSTR CharPrevA(
  [in] LPCSTR lpszStart,
  [in] LPCSTR lpszCurrent
);
```

Parameters

`[in] lpszStart`

Type: **LPCTSTR**

The beginning of the string.

`[in] lpszCurrent`

Type: **LPCTSTR**

A character in a null-terminated string.

Return value

Type: **LPTSTR**

The return value is a pointer to the preceding character in the string, or to the first character in the string if the *lpszCurrent* parameter equals the *lpszStart* parameter.

Remarks

When called as an ANSI function, **CharPrev** uses the system default code-page, whereas CharPrevExA specifies a code-page to use.

This function works with default "user" expectations of characters when dealing with diacritics. For example: A string that contains U+0061 U+030a "LATIN SMALL LETTER A" + COMBINING RING ABOVE" — which looks like "å", will advance two code points, not one. A string that contains U+0061 U+0301 U+0302 U+0303 U+0304 — which looks like "a´^~¯", will advance five code points, not one, and so on.

> **Note**
>
> The winuser.h header defines CharPrev as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

CharNext

CharNextExA

CharPrevExA

Conceptual

Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CharPrevExA function (winuser.h)

Article02/22/2024

Retrieves the pointer to the preceding character in a string. This function can handle strings consisting of either single- or multi-byte characters.

Syntax

```
C++

LPSTR CharPrevExA(
  [in] WORD   CodePage,
  [in] LPCSTR lpStart,
  [in] LPCSTR lpCurrentChar,
  [in] DWORD  dwFlags
);
```

Parameters

`[in] CodePage`

Type: **WORD**

The identifier of the code page to use to check lead-byte ranges. Can be one of the code-page values provided in Code Page Identifiers, or one of the following predefined values.

Expand table

Value	Meaning
CP_ACP 0	Use system default ANSI code page.
CP_MACCP 2	Use the system default Macintosh code page.
CP_OEMCP 1	Use system default OEM code page.

`[in] lpStart`

Type: **LPCSTR**

The beginning of the string.

`[in] lpCurrentChar`

Type: **LPCSTR**

A character in a null-terminated string.

`[in] dwFlags`

Type: **DWORD**

This parameter is reserved and must be zero.

Return value

Type: **LPSTR**

The return value is a pointer to the preceding character in the string, or to the first character in the string if the *lpCurrentChar* parameter equals the *lpStart* parameter.

Remarks

CharPrevExA specifies a code-page to use, whereas CharPrev (if called as an ANSI function) uses the system default code-page.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

CharNextExA

CharPrev

Conceptual

Reference

Strings

Feedback

CharToOemA function (winuser.h)

Article02/22/2024

Translates a string into the OEM-defined character set.

> **Warning** Do not use. See Security Considerations.

Syntax

```
C++

BOOL CharToOemA(
  [in]  LPCSTR pSrc,
  [out] LPSTR  pDst
);
```

Parameters

`[in] pSrc`

Type: **LPCTSTR**

The null-terminated string to be translated.

`[out] pDst`

Type: **LPSTR**

The destination buffer, which receives the translated string. If the **CharToOem** function is being used as an ANSI function, the string can be translated in place by setting the *lpszDst* parameter to the same address as the *lpszSrc* parameter. This cannot be done if **CharToOem** is being used as a wide-character function.

Return value

Type: **BOOL**

The return value is always nonzero except when you pass the same address to *lpszSrc* and *lpszDst* in the wide-character version of the function. In this case the function

returns zero and GetLastError returns **ERROR_INVALID_ADDRESS**.

Remarks

> **Note**
>
> The winuser.h header defines CharToOem as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-chartranslation-l1-1-0 (introduced in Windows 8)

See also

CharToOemBuff

Conceptual

OemToChar

OemToCharBuff

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CharToOemBuffA function (winuser.h)

Article02/22/2024

Translates a specified number of characters in a string into the OEM-defined character set.

Syntax

```
C++

BOOL CharToOemBuffA(
  [in]  LPCSTR lpszSrc,
  [out] LPSTR  lpszDst,
  [in]  DWORD  cchDstLength
);
```

Parameters

`[in] lpszSrc`

Type: **LPCTSTR**

The null-terminated string to be translated.

`[out] lpszDst`

Type: **LPSTR**

The buffer for the translated string. If the **CharToOemBuff** function is being used as an ANSI function, the string can be translated in place by setting the *lpszDst* parameter to the same address as the *lpszSrc* parameter. This cannot be done if **CharToOemBuff** is being used as a wide-character function.

`[in] cchDstLength`

Type: **DWORD**

The number of characters to translate in the string identified by the *lpszSrc* parameter.

Return value

Type: **BOOL**

The return value is always nonzero except when you pass the same address to *lpszSrc* and *lpszDst* in the wide-character version of the function. In this case the function returns zero and GetLastError returns **ERROR_INVALID_ADDRESS**.

Remarks

Unlike the CharToOem function, the **CharToOemBuff** function does not stop converting characters when it encounters a null character in the buffer pointed to by *lpszSrc*. The **CharToOemBuff** function converts all *cchDstLength* characters.

> **Note**
>
> The winuser.h header defines CharToOemBuff as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-chartranslation-l1-1-0 (introduced in Windows 8)

See also

CharToOem

Conceptual

OemToChar

OemToCharBuff

Reference

Strings

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CharUpperA function (winuser.h)

Article02/22/2024

Converts a character string or a single character to uppercase. If the operand is a character string, the function converts the characters in place.

Syntax

```
C++

LPSTR CharUpperA(
  [in, out] LPSTR lpsz
);
```

Parameters

`[in, out] lpsz`

Type: **LPTSTR**

A null-terminated string, or a single character. If the high-order word of this parameter is zero, the low-order word must contain a single character to be converted.

Return value

Type: **LPTSTR**

If the operand is a character string, the function returns a pointer to the converted string. Because the string is converted in place, the return value is equal to *lpsz*.

If the operand is a single character, the return value is a 32-bit value whose high-order word is zero, and low-order word contains the converted character.

There is no indication of success or failure. Failure is rare. There is no extended error information for this function; do not call GetLastError.

Remarks

Note that **CharUpper** always maps lowercase I ("i") to uppercase I, even when the current language is Turkish or Azerbaijani. If you need a function that is linguistically

sensitive in this respect, call LCMapString.

Conversion to Unicode in the ANSI version of the function is done with the system default locale in all cases.

> **Note**
>
> The winuser.h header defines CharUpper as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

CharLower

CharLowerBuff

CharUpperBuff

Conceptual

Reference

Strings

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CharUpperBuffA function (winuser.h)

Article02/22/2024

Converts lowercase characters in a buffer to uppercase characters. The function converts the characters in place.

Syntax

```
C++

DWORD CharUpperBuffA(
  [in, out] LPSTR lpsz,
  [in]      DWORD cchLength
);
```

Parameters

`[in, out] lpsz`

Type: **LPTSTR**

A buffer containing one or more characters to be processed.

`[in] cchLength`

Type: **DWORD**

The size, in characters, of the buffer pointed to by *lpsz*.

The function examines each character, and converts lowercase characters to uppercase characters. The function examines the number of characters indicated by *cchLength*, even if one or more characters are null characters.

Return value

Type: **DWORD**

The return value is the number of characters processed.

For example, if **CharUpperBuff**("Zenith of API Sets", 10) succeeds, the return value is 10.

Remarks

Note that **CharUpperBuff** always maps lowercase I ("i") to uppercase I, even when the current language is Turkish or Azerbaijani. If you need a function that is linguistically sensitive in this respect, call LCMapString.

Conversion to Unicode in the ANSI version of the function is done with the system default locale in all cases.

Examples

For an example, see Creating and Using a Temporary File.

> **Note**
>
> The winuser.h header defines CharUpperBuff as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

CharLower

CharLowerBuff

CharUpper

Conceptual

Reference

Strings

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

IsCharAlphaA function (winuser.h)

Article02/22/2024

Determines whether a character is an alphabetical character. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

Syntax

```
C++

BOOL IsCharAlphaA(
  [in] CHAR ch
);
```

Parameters

`[in] ch`

Type: **TCHAR**

The character to be tested.

Return value

Type: **BOOL**

If the character is alphabetical, the return value is nonzero.

If the character is not alphabetical, the return value is zero. To get extended error information, call GetLastError.

Remarks

> **Note**
>
> The winuser.h header defines IsCharAlpha as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with

code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

IsCharAlphaNumeric

Reference

Strings

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

IsCharAlphaNumericA function (winuser.h)

Article02/09/2023

Determines whether a character is either an alphabetical or a numeric character. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

Syntax

```
C++

BOOL IsCharAlphaNumericA(
  [in] CHAR ch
);
```

Parameters

`[in] ch`

Type: **TCHAR**

The character to be tested.

Return value

Type: **BOOL**

If the character is alphanumeric, the return value is nonzero.

If the character is not alphanumeric, the return value is zero. To get extended error information, call GetLastError.

Remarks

> ⓘ **Note**
>
> The winuser.h header defines IsCharAlphaNumeric as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the

UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

IsCharAlpha

Reference

Strings

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

IsCharLowerA function (winuser.h)

Article02/09/2023

Determines whether a character is lowercase. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

Syntax

```
C++

BOOL IsCharLowerA(
  [in] CHAR ch
);
```

Parameters

`[in] ch`

Type: **TCHAR**

The character to be tested.

Return value

Type: **BOOL**

If the character is lowercase, the return value is nonzero.

If the character is not lowercase, the return value is zero. To get extended error information, call GetLastError.

Remarks

> ⓘ **Note**
>
> The winuser.h header defines IsCharLower as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with

code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

IsCharUpper

Reference

Strings

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

IsCharUpperA function (winuser.h)

Article02/22/2024

Determines whether a character is uppercase. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

Syntax

```
C++

BOOL IsCharUpperA(
  [in] CHAR ch
);
```

Parameters

`[in] ch`

Type: **CHAR**

The character to be tested.

Return value

Type: **BOOL**

If the character is uppercase, the return value is nonzero.

If the character is not uppercase, the return value is zero. To get extended error information, call GetLastError.

Remarks

> ⓘ **Note**
>
> The winuser.h header defines IsCharUpper as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with

code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

IsCharLower

Reference

Strings

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

LoadStringA function (winuser.h)

Article12/08/2023

Loads a string resource from the executable file associated with a specified module and either copies the string into a buffer with a terminating null character or returns a read-only pointer to the string resource itself.

Syntax

```
C++

int LoadStringA(
  [in, optional] HINSTANCE hInstance,
  [in]           UINT      uID,
  [out]          LPSTR     lpBuffer,
  [in]           int       cchBufferMax
);
```

Parameters

`[in, optional] hInstance`

Type: **HINSTANCE**

A handle to an instance of the module whose executable file contains the string resource. To get the handle to the application itself, call the GetModuleHandle function with **NULL**.

`[in] uID`

Type: **UINT**

The identifier of the string to be loaded.

`[out] lpBuffer`

Type: **LPTSTR**

The buffer to receive the string. Must be at least *cchBufferMax* characters in size.

`[in] cchBufferMax`

Type: **int**

The size of the buffer, in characters. The string is truncated and null-terminated if it is longer than the number of characters specified. This parameter may not be zero.

Return value

Type: **int**

If the function succeeds, the return value is the number of characters copied into the buffer, not including the terminating null character.

If the string resource does not exist, the return value is zero.

To get extended error information, call GetLastError.

Remarks

Unlike the **LoadStringW** function, the **LoadStringA** function does not support passing a value of zero for *cchBufferMax*. Doing so will corrupt memory.

Security Remarks

Using this function incorrectly can compromise the security of your application. Incorrect use includes specifying the wrong size in the *cchBufferMax* parameter. For example, if *lpBuffer* points to a buffer *szBuffer* which is declared as `TCHAR szBuffer[100]`, then sizeof(szBuffer) gives the size of the buffer in bytes, which could lead to a buffer overflow for the Unicode version of the function. Buffer overflow situations are the cause of many security problems in applications. In this case, using `sizeof(szBuffer)/sizeof(TCHAR)` or `sizeof(szBuffer)/sizeof(szBuffer[0])` would give the proper size of the buffer.

Examples

For an example, see Creating a Child Window

> ⓘ **Note**
>
> The winuser.h header defines LoadString as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that

not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

FormatMessage

LoadAccelerators

LoadBitmap

LoadCursor

LoadIcon

LoadMenu

LoadMenuIndirect

Other Resources

Reference

Strings

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

lstrcatA function (winbase.h)

Article02/22/2024

Appends one string to another.

> **Warning** Do not use. Consider using **StringCchCat** instead. See Security Considerations.

Syntax

```
C++

LPSTR lstrcatA(
  [in, out] LPSTR  lpString1,
  [in]      LPCSTR lpString2
);
```

Parameters

`[in, out] lpString1`

Type: **LPTSTR**

The first null-terminated string. This buffer must be large enough to contain both strings.

`[in] lpString2`

Type: **LPTSTR**

The null-terminated string to be appended to the string specified in the *lpString1* parameter.

Return value

Type: **LPTSTR**

If the function succeeds, the return value is a pointer to the buffer.

If the function fails, the return value is **NULL** and *lpString1* may not be null-terminated.

Remarks

> **Note**
>
> The winbase.h header defines lstrcat as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Conceptual

Reference

StringCbCat

StringCbCatEx

StringCbCatN

StringCbCatNEx

StringCchCat

StringCchCatEx

StringCchCatN

StringCchCatNEx

Strings

lstrcmp

lstrcmpi

lstrlen

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

lstrcmpA function (winbase.h)

Article02/09/2023

Compares two character strings. The comparison is case-sensitive.

To perform a comparison that is not case-sensitive, use the lstrcmpi function.

Syntax

```
C++

int lstrcmpA(
  [in] LPCSTR lpString1,
  [in] LPCSTR lpString2
);
```

Parameters

`[in] lpString1`

Type: **LPCTSTR**

The first null-terminated string to be compared.

`[in] lpString2`

Type: **LPCTSTR**

The second null-terminated string to be compared.

Return value

Type: **int**

If the string pointed to by *lpString1* is less than the string pointed to by *lpString2*, the return value is negative. If the string pointed to by *lpString1* is greater than the string pointed to by *lpString2*, the return value is positive. If the strings are equal, the return value is zero.

Remarks

The **lstrcmp** function compares two strings by checking the first characters against each other, the second characters against each other, and so on until it finds an inequality or reaches the ends of the strings.

Note that the *lpString1* and *lpString2* parameters must be null-terminated, otherwise the string comparison can be incorrect.

The function calls CompareStringEx, using the current thread locale, and subtracts 2 from the result, to maintain the C run-time conventions for comparing strings.

The language (user locale) selected by the user at setup time, or through Control Panel, determines which string is greater (or whether the strings are the same). If no language (user locale) is selected, the system performs the comparison by using default values.

With a double-byte character set (DBCS) version of the system, this function can compare two DBCS strings.

The **lstrcmp** function uses a word sort, rather than a string sort. A word sort treats hyphens and apostrophes differently than it treats other symbols that are not alphanumeric, in order to ensure that words such as "coop" and "co-op" stay together within a sorted list. For a detailed discussion of word sorts and string sorts, see Handling Sorting in Your Applications.

Security Remarks

See Security Considerations: International Features for security considerations regarding choice of comparison functions.

> **Note**
>
> The winbase.h header defines lstrcmp as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

CompareString

CompareStringEx

CompareStringOrdinal

Conceptual

Other Resources

Reference

Strings

lstrcat

lstrcmpi

lstrcpy

lstrlen

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

lstrcmpiA function (winbase.h)

Article02/09/2023

Compares two character strings. The comparison is not case-sensitive.

To perform a comparison that is case-sensitive, use the lstrcmp function.

Syntax

```
C++

int lstrcmpiA(
  [in] LPCSTR lpString1,
  [in] LPCSTR lpString2
);
```

Parameters

`[in] lpString1`

Type: **LPCTSTR**

The first null-terminated string to be compared.

`[in] lpString2`

Type: **LPCTSTR**

The second null-terminated string to be compared.

Return value

Type: **int**

If the string pointed to by *lpString1* is less than the string pointed to by *lpString2*, the return value is negative. If the string pointed to by *lpString1* is greater than the string pointed to by *lpString2*, the return value is positive. If the strings are equal, the return value is zero.

Remarks

The **lstrcmpi** function compares two strings by checking the first characters against each other, the second characters against each other, and so on until it finds an inequality or reaches the ends of the strings.

Note that the *lpString1* and *lpString2* parameters must be null-terminated, otherwise the string comparison can be incorrect.

The function calls CompareStringEx, using the current thread locale, and subtracts 2 from the result, to maintain the C run-time conventions for comparing strings.

For some locales, the **lstrcmpi** function may be insufficient. If this occurs, use CompareStringEx to ensure proper comparison. For example, in Japan call with the **NORM_IGNORECASE**, **NORM_IGNOREKANATYPE**, and **NORM_IGNOREWIDTH** values to achieve the most appropriate non-exact string comparison. The **NORM_IGNOREKANATYPE** and **NORM_IGNOREWIDTH** values are ignored in non-Asian locales, so you can set these values for all locales and be guaranteed to have a culturally correct "insensitive" sorting regardless of the locale. Note that specifying these values slows performance, so use them only when necessary.

With a double-byte character set (DBCS) version of the system, this function can compare two DBCS strings.

The **lstrcmpi** function uses a word sort, rather than a string sort. A word sort treats hyphens and apostrophes differently than it treats other symbols that are not alphanumeric, in order to ensure that words such as "coop" and "co-op" stay together within a sorted list. For a detailed discussion of word sorts and string sorts, see Handling Sorting in Your Applications.

Security Remarks

See Security Considerations: International Features for security considerations regarding choice of comparison functions.

> ⓘ **Note**
>
> The winbase.h header defines lstrcmpi as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

CompareString

CompareStringEx

CompareStringOrdinal

Conceptual

Other Resources

Reference

Strings

lstrcat

lstrcmp

lstrcpy

lstrlen

Feedback

Was this page helpful?

lstrcpyA function (winbase.h)

Article02/09/2023

Copies a string to a buffer.

> **Warning** Do not use. Consider using **StringCchCopy** instead. See Remarks.

Syntax

```
C++

LPSTR lstrcpyA(
  [out] LPSTR  lpString1,
  [in]  LPCSTR lpString2
);
```

Parameters

`[out] lpString1`

Type: **LPTSTR**

A buffer to receive the contents of the string pointed to by the *lpString2* parameter. The buffer must be large enough to contain the string, including the terminating null character.

`[in] lpString2`

Type: **LPTSTR**

The null-terminated string to be copied.

Return value

Type: **LPTSTR**

If the function succeeds, the return value is a pointer to the buffer.

If the function fails, the return value is **NULL** and *lpString1* may not be null-terminated.

Remarks

With a double-byte character set (DBCS) version of the system, this function can be used to copy a DBCS string.

The **lstrcpy** function has an undefined behavior if source and destination buffers overlap.

Security Remarks

Using this function incorrectly can compromise the security of your application. This function uses structured exception handling (SEH) to catch access violations and other errors. When this function catches SEH errors, it returns **NULL** without null-terminating the string and without notifying the caller of the error. The caller is not safe to assume that insufficient space is the error condition.

lpString1 must be large enough to hold *lpString2* and the closing '\0', otherwise a buffer overrun may occur.

Buffer overflow situations are the cause of many security problems in applications and can cause a denial of service attack against the application if an access violation occurs. In the worst case, a buffer overrun may allow an attacker to inject executable code into your process, especially if *lpString1* is a stack-based buffer.

Consider using StringCchCopy instead; use either `StringCchCopy(buffer, sizeof(buffer)/sizeof(buffer[0]), src);`, being aware that `buffer` must not be a pointer or use `StringCchCopy(buffer, ARRAYSIZE(buffer), src);`, being aware that, when copying to a pointer, the caller is responsible for passing in the size of the pointed-to memory in characters.

> ⓘ **Note**
>
> The winbase.h header defines lstrcpy as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Conceptual

Reference

StringCbCopy

StringCbCopyEx

StringCbCopyN

StringCbCopyNEx

StringCchCopy

StringCchCopyEx

StringCchCopyN

StringCchCopyNEx

Strings

lstrcmp

lstrcmpi

lstrlen

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

lstrcpynA function (winbase.h)

Article02/09/2023

Copies a specified number of characters from a source string into a buffer.

> **Warning** Do not use. Consider using **StringCchCopy** instead. See Remarks.

Syntax

```
C++

LPSTR lstrcpynA(
  [out] LPSTR  lpString1,
  [in]  LPCSTR lpString2,
  [in]  int    iMaxLength
);
```

Parameters

`[out] lpString1`

Type: **LPTSTR**

The destination buffer, which receives the copied characters. The buffer must be large enough to contain the number of **TCHAR** values specified by *iMaxLength*, including room for a terminating null character.

`[in] lpString2`

Type: **LPCTSTR**

The source string from which the function is to copy characters.

`[in] iMaxLength`

Type: **int**

The number of **TCHAR** values to be copied from the string pointed to by *lpString2* into the buffer pointed to by *lpString1*, including a terminating null character.

Return value

Type: **LPTSTR**

If the function succeeds, the return value is a pointer to the buffer. The function can succeed even if the source string is greater than *iMaxLength* characters.

If the function fails, the return value is **NULL** and *lpString1* may not be null-terminated.

Remarks

The buffer pointed to by *lpString1* must be large enough to include a terminating null character, and the string length value specified by *iMaxLength* includes room for a terminating null character.

The **lstrcpyn** function has an undefined behavior if source and destination buffers overlap.

Security Warning

Using this function incorrectly can compromise the security of your application. This function uses structured exception handling (SEH) to catch access violations and other errors. When this function catches SEH errors, it returns **NULL** without null-terminating the string and without notifying the caller of the error. The caller is not safe to assume that insufficient space is the error condition.

If the buffer pointed to by *lpString1* is not large enough to contain the copied string, a buffer overrun can occur. When copying an entire string, note that **sizeof** returns the number of bytes. For example, if *lpString1* points to a buffer *szString1* which is declared as `TCHAR szString[100]`, then sizeof(szString1) gives the size of the buffer in bytes rather than **WCHAR**, which could lead to a buffer overflow for the Unicode version of the function.

Buffer overflow situations are the cause of many security problems in applications and can cause a denial of service attack against the application if an access violation occurs. In the worst case, a buffer overrun may allow an attacker to inject executable code into your process, especially if *lpString1* is a stack-based buffer.

Using `sizeof(szString1)/sizeof(szString1[0])` gives the proper size of the buffer.

Consider using StringCchCopy instead; use either `StringCchCopy(buffer, sizeof(buffer)/sizeof(buffer[0]), src);`, being aware that `buffer` must not be a

pointer or use `StringCchCopy(buffer, ARRAYSIZE(buffer), src);`, being aware that, when copying to a pointer, the caller is responsible for passing in the size of the pointed-to memory in characters.

Review Security Considerations: Windows User Interface before continuing.

> **Note**
>
> The winbase.h header defines lstrcpyn as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Conceptual

Reference

StringCbCopy

StringCbCopyEx

StringCbCopyN

StringCbCopyNEx

StringCbLength

StringCchCopy

StringCchCopyEx

StringCchCopyN

StringCchCopyNEx

StringCchLength

Strings

lstrcmp

lstrcmpi

lstrlen

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

lstrlenA function (winbase.h)

Article02/22/2024

Determines the length of the specified string (not including the terminating null character).

Syntax

```
C++

int lstrlenA(
  [in] LPCSTR lpString
);
```

Parameters

`[in] lpString`

Type: **LPCTSTR**

The null-terminated string to be checked.

Return value

Type: **int**

The function returns the length of the string, in characters. If *lpString* is **NULL**, the function returns 0.

Remarks

> **Note**
>
> The winbase.h header defines lstrlen as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Conceptual

Reference

StringCbLength

StringCchLength

Strings

lstrcat

lstrcmp

lstrcmpi

lstrcpy

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

OemToCharA function (winuser.h)

Article02/09/2023

Translates a string from the OEM-defined character set into either an ANSI or a wide-character string.

> **Warning** Do not use. See Security Considerations.

Syntax

```
C++

BOOL OemToCharA(
  [in]  LPCSTR pSrc,
  [out] LPSTR  pDst
);
```

Parameters

`[in] pSrc`

Type: **LPCSTR**

A null-terminated string of characters from the OEM-defined character set.

`[out] pDst`

Type: **LPTSTR**

The destination buffer, which receives the translated string. If the **OemToChar** function is being used as an ANSI function, the string can be translated in place by setting the *lpszDst* parameter to the same address as the *lpszSrc* parameter. This cannot be done if **OemToChar** is being used as a wide-character function.

Return value

Type: **BOOL**

The return value is always nonzero except when you pass the same address to *lpszSrc* and *lpszDst* in the wide-character version of the function. In this case the function returns zero and GetLastError returns **ERROR_INVALID_ADDRESS**.

Remarks

> **Note**
>
> The winuser.h header defines OemToChar as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-chartranslation-l1-1-0 (introduced in Windows 8)

See also

CharToOem

CharToOemBuff

Conceptual

OemToCharBuff

Reference

Strings

Feedback

OemToCharBuffA function (winuser.h)

Article02/09/2023

Translates a specified number of characters in a string from the OEM-defined character set into either an ANSI or a wide-character string.

Syntax

```
C++

BOOL OemToCharBuffA(
  [in]  LPCSTR lpszSrc,
  [out] LPSTR  lpszDst,
  [in]  DWORD  cchDstLength
);
```

Parameters

`[in] lpszSrc`

Type: **LPCSTR**

One or more characters from the OEM-defined character set.

`[out] lpszDst`

Type: **LPTSTR**

The destination buffer, which receives the translated string. If the **OemToCharBuff** function is being used as an ANSI function, the string can be translated in place by setting the *lpszDst* parameter to the same address as the *lpszSrc* parameter. This cannot be done if the **OemToCharBuff** function is being used as a wide-character function.

`[in] cchDstLength`

Type: **DWORD**

The number of characters to be translated in the buffer identified by the *lpszSrc* parameter.

Return value

Type: **BOOL**

The return value is always nonzero except when you pass the same address to *lpszSrc* and *lpszDst* in the wide-character version of the function. In this case the function returns zero and GetLastError returns **ERROR_INVALID_ADDRESS**.

Remarks

Unlike the OemToChar function, the **OemToCharBuff** function does not stop converting characters when it encounters a null character in the buffer pointed to by *lpszSrc*. The **OemToCharBuff** function converts all *cchDstLength* characters.

> **Note**
>
> The winuser.h header defines OemToCharBuff as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-chartranslation-l1-1-0 (introduced in Windows 8)

See also

CharToOem

CharToOemBuff

Conceptual

OemToChar

Reference

Strings

Feedback

wsprintfA function (winuser.h)

Article01/26/2024

Writes formatted data to the specified buffer. Any arguments are converted and copied to the output buffer according to the corresponding format specification in the format string. The function appends a terminating null character to the characters it writes, but the return value does not include the terminating null character in its character count.

> **Note** Do not use. Consider using one of the following functions instead: **StringCbPrintf**, **StringCbPrintfEx**, **StringCchPrintf**, or **StringCchPrintfEx**. See Security Considerations.

Syntax

```
C++

int WINAPIV wsprintfA(
  [out] LPSTR  unnamedParam1,
  [in]  LPCSTR unnamedParam2,
        ...
);
```

Parameters

`[out] unnamedParam1`

Type: **LPTSTR**

The buffer that is to receive the formatted output. The maximum size of the buffer is 1,024 bytes.

`[in] unnamedParam2`

Type: **LPCTSTR**

The format-control specifications. In addition to ordinary ASCII characters, a format specification for each argument appears in this string. For more information about the format specification, see the Remarks section.

`...`

One or more optional arguments. The number and type of argument parameters depend on the corresponding format-control specifications in the *lpFmt* parameter.

Return value

Type: **int**

If the function succeeds, the return value is the number of characters stored in the output buffer, not counting the terminating null character.

If the function fails, the return value is less than the length of the expected output. To get extended error information, call GetLastError.

Remarks

The format-control string contains format specifications that determine the output format for the arguments following the *lpFmt* parameter. Format specifications, discussed below, always begin with a percent sign (%). If a percent sign is followed by a character that has no meaning as a format field, the character is not formatted (for example, %% produces a single percent-sign character).

The format-control string is read from left to right. When the first format specification (if any) is encountered, it causes the value of the first argument after the format-control string to be converted and copied to the output buffer according to the format specification. The second format specification causes the second argument to be converted and copied, and so on. If there are more arguments than format specifications, the extra arguments are ignored. If there are not enough arguments for all of the format specifications, the results are undefined.

A format specification has the following form:

%[-][#][**0**][*width*][.*precision*]*type*

Each field is a single character or a number signifying a particular format option. The *type* characters that appear after the last optional format field determine whether the associated argument is interpreted as a character, a string, or a number. The simplest format specification contains only the percent sign and a type character (for example, %s). The optional fields control other aspects of the formatting. Following are the optional and required fields and their meanings.

Expand table

Field	Meaning
-	Pad the output with blanks or zeros to the right to fill the field width, justifying output to the left. If this field is omitted, the output is padded to the left, justifying it to the right.
#	Prefix hexadecimal values with 0x (lowercase) or 0X (uppercase).
0	Pad the output value with zeros to fill the field width. If this field is omitted, the output value is padded with blank spaces.
width	Copy the specified minimum number of characters to the output buffer. The *width* field is a nonnegative integer. The width specification never causes a value to be truncated; if the number of characters in the output value is greater than the specified width, or if the *width* field is not present, all characters of the value are printed, subject to the precision specification.
.precision	For numbers, copy the specified minimum number of digits to the output buffer. If the number of digits in the argument is less than the specified precision, the output value is padded on the left with zeros. The value is not truncated when the number of digits exceeds the specified precision. If the specified precision is 0 or omitted entirely, or if the period (.) appears without a number following it, the precision is set to 1. For strings, copy the specified maximum number of characters to the output buffer.

type	Output the corresponding argument as a character, a string, or a number. This field can be any of the following values.
	`c`
	Single character. This value is interpreted as type **CHAR** by **wsprintfA** and type **WCHAR** by **wsprintfW**. Note **wsprintf** is a macro defined as **wsprintfA** (Unicode not defined) or **wsprintfW** (Unicode defined).
	`C`
	Single character. This value is interpreted as type **WCHAR** by **wsprintfA** and type **CHAR** by **wsprintfW**. Note **wsprintf** is a macro defined as **wsprintfA** (Unicode not defined) or **wsprintfW** (Unicode defined).
	`d`
	Signed decimal integer. This value is equivalent to `i`.
	`hc`, `hC`
	Single character. If the character has a numeric value of zero it is ignored. This value is always interpreted as type **CHAR**, even when the calling application defines Unicode.
	`hd`
	Signed short integer argument.
	`hs`, `hS`
	String. This value is always interpreted as type **LPSTR**, even when the calling application defines Unicode.
	`hu`
	Unsigned short integer.
	`i`
	Signed decimal integer. This value is equivalent to `d`.
	`Ix`, `IX`

64-bit unsigned hexadecimal integer in lowercase or uppercase on 64-bit platforms, 32-bit unsigned hexadecimal integer in lowercase or uppercase on 32-bit platforms.

`lc`, `lC`

Single character. If the character has a numeric value of zero it is ignored. This value is always interpreted as type **WCHAR**, even when the calling application defines Unicode.

`ld`

Long signed integer. This value is equivalent to `li`.

`li`

Long signed integer. This value is equivalent to `ld`.

`ls`, `lS`

String. This value is always interpreted as type **LPWSTR**, even when the calling application does not define Unicode. This value is equivalent to `ws`.

`lu`

Long unsigned integer.

`lx`, `lX`

Long unsigned hexadecimal integer in lowercase or uppercase.

`p`

Pointer. The address is printed using hexadecimal.

`s`

String. This value is interpreted as type **LPSTR** by **wsprintfA** and type **LPWSTR** by **wsprintfW**. Note **wsprintf** is a macro defined as **wsprintfA** (Unicode not defined) or **wsprintfW** (Unicode defined).

`S`

String. This value is interpreted as type **LPWSTR** by **wsprintfA** and type **LPSTR** by **wsprintfW**. Note **wsprintf** is a macro defined as **wsprintfA** (Unicode not defined) or **wsprintfW** (Unicode defined).

`u`

Unsigned integer argument.

`x`, `X`

Unsigned hexadecimal integer in lowercase or uppercase.

Note It is important to note that **wsprintf** uses the C calling convention (**_cdecl**), rather than the standard call (**_stdcall**) calling convention. As a result, it is the responsibility of the calling process to pop arguments off the stack, and arguments are pushed on the stack from right to left. In C-language modules, the C compiler performs this task.

To use buffers larger than 1024 bytes, use **_snwprintf**. For more information, see the documentation for the C run-time library.

Note

The winuser.h header defines wsprintf as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

Reference

StringCbPrintf

StringCbPrintfEx

StringCbVPrintf

StringCbVPrintfEx

StringCchPrintf

StringCchPrintfEx

StringCchVPrintf

StringCchVPrintfEx

Strings

wvsprintf

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

wvsprintfA function (winuser.h)

Article02/09/2023

Writes formatted data to the specified buffer using a pointer to a list of arguments. The items pointed to by the argument list are converted and copied to an output buffer according to the corresponding format specification in the format-control string. The function appends a terminating null character to the characters it writes, but the return value does not include the terminating null character in its character count.

> **Warning** Do not use. Consider using one of the following functions instead: **StringCbVPrintf**, **StringCbVPrintfEx**, **StringCchVPrintf**, or **StringCchVPrintfEx**. See Security Considerations.

Syntax

C++

```
int wvsprintfA(
  [out] LPSTR   unnamedParam1,
  [in]  LPCSTR  unnamedParam2,
  [in]  va_list arglist
);
```

Parameters

`[out] unnamedParam1`

Type: **LPTSTR**

The buffer that is to receive the formatted output. The maximum size of the buffer is 1,024 bytes.

`[in] unnamedParam2`

Type: **LPCTSTR**

The format-control specifications. In addition to ordinary ASCII characters, a format specification for each argument appears in this string. For more information about the format specification, see the wsprintf function.

`[in] arglist`

Type: **va_list**

Each element of this list specifies an argument for the format-control string. The number, type, and interpretation of the arguments depend on the corresponding format-control specifications in the *lpFmt* parameter.

Return value

Type: **int**

If the function succeeds, the return value is the number of characters stored in the buffer, not counting the terminating null character.

If the function fails, the return value is less than the length of the expected output. To get extended error information, call GetLastError.

Remarks

The function copies the format-control string into the output buffer character by character, starting with the first character in the string. When it encounters a format specification in the string, the function retrieves the value of the next available argument (starting with the first argument in the list), converts that value into the specified format, and copies the result to the output buffer. The function continues to copy characters and expand format specifications in this way until it reaches the end of the format-control string. If there are more arguments than format specifications, the extra arguments are ignored. If there are not enough arguments for all of the format specifications, the results are undefined.

> ⓘ **Note**
>
> The winuser.h header defines wvsprintf as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

Reference

StringCbPrintf

StringCbPrintfEx

StringCbVPrintf

StringCbVPrintfEx

StringCchPrintf

StringCchPrintfEx

StringCchVPrintf

StringCchVPrintfEx

Strings

wsprintf

Feedback

Was this page helpful? Yes No

Strsafe Functions

Article • 04/27/2021

In this section

- StringCbCat
- StringCbCatEx
- StringCbCatN
- StringCbCatNEx
- StringCbCopy
- StringCbCopyEx
- StringCbCopyN
- StringCbCopyNEx
- StringCbGets
- StringCbGetsEx
- StringCbLength
- StringCbPrintf_l
- StringCbPrintf_lEx
- StringCbPrintf
- StringCbPrintfEx
- StringCbVPrintf_l
- StringCbVPrintf_lEx
- StringCbVPrintf
- StringCbVPrintfEx
- StringCchCat
- StringCchCatEx
- StringCchCatN
- StringCchCatNEx
- StringCchCopy
- StringCchCopyEx
- StringCchCopyN
- StringCchCopyNEx
- StringCchGets
- StringCchGetsEx
- StringCchLength
- StringCchPrintf_l
- StringCchPrintf_lEx
- StringCchPrintf
- StringCchPrintfEx

- StringCchVPrintf_l
- StringCchVPrintf_lEx
- StringCchVPrintf
- StringCchVPrintfEx
- UnalignedStringCbLength
- UnalignedStringCchLength

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

StringCbCatA function (strsafe.h)

Article02/09/2023

Concatenates one string to another string. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCbCat is a replacement for the following functions:

- strcat, wcscat, _tcsat
- lstrcat
- StrCat
- StrCatBuff

Syntax

```
C++

STRSAFEAPI StringCbCatA(
  [in, out] STRSAFE_LPSTR  pszDest,
  [in]      size_t         cbDest,
  [in]      STRSAFE_LPCSTR pszSrc
);
```

Parameters

`[in, out] pszDest`

Type: **LPTSTR**

The string to which *pszSrc* is to be concatenated, and that will receive the entire resultant string. The string at *pszSrc* is added to the end of the string at *pszDest*.

`[in] cbDest`

Type: **size_t**

The size of the destination buffer, in bytes. This value must consider the length of *pszSrc* plus the length of *pszDest* plus the terminating null character. The maximum number of bytes allowed is `STRSAFE_MAX_CCH * sizeof(TCHAR)`.

`[in] pszSrc`

Type: **LPCTSTR**

The source string that is to be concatenated to the end of *pszDest*. This string must be null-terminated.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Source data was present, the strings were fully concatenated without truncation, and the resultant destination buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cbDest* is either less than `sizeof(TCHAR)` or larger than the maximum allowed value.
STRSAFE_E_INSUFFICIENT_BUFFER	The concatenation operation failed due to insufficient buffer space. The destination buffer contains a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

StringCbCat provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. It always null-terminates and never overflows a valid destination buffer, even if the contents of the source string change during the operation.

StringCbCat can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use, as shown in the following table.

Expand table

String Data Type	String Literal	Function

char	"string"	**StringCbCatA**
TCHAR	TEXT("string")	**StringCbCat**
WCHAR	L"string"	**StringCbCatW**

Behavior is undefined if the strings pointed to by *pszSrc* and *pszDest* overlap.

Neither *pszSrc* nor *pszDest* should be **NULL**. See StringCbCatEx if you require the handling of null string pointer values.

> **Note**
>
> The strsafe.h header defines StringCbCat as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbCatEx

StringCbCatN

StringCchCat

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCbCatExA function (strsafe.h)

Article02/09/2023

Concatenates one string to another string. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCbCatEx adds to the functionality of StringCbCat by returning a pointer to the end of the destination string as well as the number of bytes left unused in that string. Flags may also be passed to the function for additional control.

StringCbCatEx is a replacement for the following functions:

- strcat, wcscat, _tcsat
- lstrcat
- StrCat
- StrCatBuff

Syntax

```
C++

STRSAFEAPI StringCbCatExA(
  [in, out]       STRSAFE_LPSTR  pszDest,
  [in]            size_t         cbDest,
  [in]            STRSAFE_LPCSTR pszSrc,
  [out, optional] STRSAFE_LPSTR  *ppszDestEnd,
  [out, optional] size_t         *pcbRemaining,
  [in]            DWORD          dwFlags
);
```

Parameters

`[in, out] pszDest`

Type: **LPTSTR**

The string to which *pszSrc* is to be concatenated, and that will receive the entire resultant string. The string at *pszSrc* is added to the end of the string at *pszDest*.

`[in] cbDest`

Type: **size_t**

The size of the destination buffer, in bytes. This value must consider the length of *pszSrc* plus the length of *pszDest* plus the terminating null character. The maximum number of bytes allowed is `STRSAFE_MAX_CCH * sizeof(TCHAR)`.

`[in] pszSrc`

Type: **LPCTSTR**

The source string that is concatenated to the end of *pszDest*. This string must be null-terminated.

`[out, optional] ppszDestEnd`

Type: **LPTSTR***

The address of a pointer to the end of *pszDest*. If *ppszDestEnd* is non-**NULL** and any data is appended to the destination buffer, this points to the terminating null character at the end of the string.

`[out, optional] pcbRemaining`

Type: **size_t***

The number of unused bytes in *pszDest*, including those used for the terminating null character. If *pcbRemaining* is **NULL**, the count is not kept or returned.

`[in] dwFlags`

Type: **DWORD**

One or more of the following values.

Expand table

Value	Meaning
STRSAFE_FILL_BEHIND_NULL 0x00000200	If the function succeeds, the low byte of *dwFlags* (0) is used to fill the uninitialized portion of *pszDest* following the terminating null character.
STRSAFE_IGNORE_NULLS 0x00000100	Treat **NULL** string pointers like empty strings (TEXT("")). This flag is useful for emulating functions such as lstrcpy.
STRSAFE_FILL_ON_FAILURE 0x00000400	If the function fails, the low byte of *dwFlags* (0) is used to fill the entire *pszDest* buffer, and the buffer is null-terminated. In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any pre-

	existing or truncated string in the destination buffer is overwritten.
STRSAFE_NULL_ON_FAILURE 0x00000800	If the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any pre-existing or truncated string in the destination buffer is overwritten.
STRSAFE_NO_TRUNCATION 0x00001000	If the function fails, *pszDest* is untouched. Nothing is added to the original contents.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Source data was present, the strings were fully concatenated without truncation, and the resultant destination buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cbDest* is either less than `sizeof(TCHAR)` or larger than the maximum allowed value.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. Depending on the value of *dwFlags*, the destination buffer may contain a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

StringCbCatEx provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns.

StringCbCatEx always null-terminates and never overflows a valid destination buffer, even if the contents of the source string change during the operation.

Behavior is undefined if the strings pointed to by *pszSrc* and *pszDest* overlap.

Neither *pszSrc* nor *pszDest* should be **NULL** unless the **STRSAFE_IGNORE_NULLS** flag is specified, in which case both may be **NULL**. However, an error due to insufficient space may still be returned even though **NULL** values are ignored.

StringCbCatEx can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use, as shown in the following table.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCbCatExA**
TCHAR	TEXT("string")	**StringCbCatEx**
WCHAR	L"string"	**StringCbCatExW**

> **Note**
>
> The strsafe.h header defines StringCbCatEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows

Requirement	Value
Header	strsafe.h

See also

Reference

StringCbCat

StringCbCatNEx

StringCchCatEx

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCbCatNA function (strsafe.h)

Article02/09/2023

Concatenates the specified number of bytes from one string to another string. The size, in bytes, of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCbCatN is a replacement for the following functions:

- strncat
- StrNCat

Syntax

```
C++

STRSAFEAPI StringCbCatNA(
  [in, out] STRSAFE_LPSTR  pszDest,
  [in]      size_t         cbDest,
  [in]      STRSAFE_PCNZCH pszSrc,
  [in]      size_t         cbToAppend
);
```

Parameters

`[in, out] pszDest`

Type: **LPTSTR**

The destination buffer, which contains the string that is to be concatenated to *pszSrc*, and will receive the resultant string. The string at *pszSrc*, up to *cbMaxAppend* bytes, is added to the end of the string at *pszDest*.

`[in] cbDest`

Type: **size_t**

The size of the destination buffer, in bytes. This value must consider the length of *pszSrc* plus either the length of *pszDest* or *cbMaxAppend* (whichever is smaller) plus the terminating null character. The maximum number of bytes allowed is `STRSAFE_MAX_CCH * sizeof(TCHAR)`.

`[in] pszSrc`

Type: **LPCTSTR**

The source string that is to be concatenated to the end of *pszDest*. This source string must be null-terminated.

`[in] cbToAppend`

Type: **size_t**

The maximum number of bytes to be appended to *pszDest*.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Source data was present, the strings were concatenated without truncation, and the resultant destination buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cbDest* is larger than the maximum allowed value, or the destination buffer is already full.
STRSAFE_E_INSUFFICIENT_BUFFER	The concatenation operation failed due to insufficient buffer space. The destination buffer contains a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

Compared to the functions it replaces, **StringCbCatN** provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCbCatN** always null-terminates and never

overflows a valid destination buffer, even if the contents of the source string change during the operation.

Behavior is undefined if the strings pointed to by *pszSrc* and *pszDest* overlap.

Neither *pszSrc* nor *pszDest* should be **NULL**. See StringCbCatNEx if you require the handling of null string pointer values.

StringCbCatN can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCbCatNA**
TCHAR	TEXT("string")	**StringCbCatN**
WCHAR	L"string"	**StringCbCatNW**

> **Note**
>
> The strsafe.h header defines StringCbCatN as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbCat

StringCbCatNEx

StringCchCatN

Feedback

StringCbCatNExA function (strsafe.h)

Article02/09/2023

Concatenates the specified number of bytes from one string to another string. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCbCatNEx is a replacement for the following functions:

- strncat
- StrNCat

StringCbCatNEx adds to the functionality of StringCbCatN by returning a pointer to the end of the destination string as well as the number of bytes left unused in that string. Flags may also be passed to the function for additional control.

Syntax

```
C++
```

```
STRSAFEAPI StringCbCatNExA(
  [in, out]       STRSAFE_LPSTR  pszDest,
  [in]            size_t         cbDest,
  [in]            STRSAFE_PCNZCH pszSrc,
  [in]            size_t         cbToAppend,
  [out, optional] STRSAFE_LPSTR  *ppszDestEnd,
  [out, optional] size_t         *pcbRemaining,
  [in]            DWORD          dwFlags
);
```

Parameters

`[in, out] pszDest`

Type: **LPTSTR**

The destination buffer, which contains the string that is to be concatenated with *pszSrc*, and will receive the entire resultant string. The string at *pszSrc* is added to the end of the string at *pszDest*.

`[in] cbDest`

Type: **size_t**

The size of the destination buffer, in bytes. This value must consider the length of *pszSrc* plus the length of *pszDest* plus the bytes used for the terminating null character. The maximum number of bytes allowed is `STRSAFE_MAX_CCH * sizeof(TCHAR)`.

`[in] pszSrc`

Type: **LPCTSTR**

The source string that is to be concatenated to the end of *pszDest*. This string must be null-terminated.

`[in] cbToAppend`

Type: **size_t**

The maximum number of bytes to append to *pszDest*.

`[out, optional] ppszDestEnd`

Type: **LPTSTR***

The address of a pointer to the end of *pszDest*. If *ppszDestEnd* is non-**NULL** and any data is appended to the destination buffer, this points to the terminating null character at the end of the string.

`[out, optional] pcbRemaining`

Type: **size_t***

The number of unused bytes in *pszDest*, including those used for the terminating null character. If *pcbRemaining* is **NULL**, the count is not kept or returned.

`[in] dwFlags`

Type: **DWORD**

One or more of the following values.

Expand table

Value	Meaning
STRSAFE_FILL_BEHIND_NULL 0x00000200	If the function succeeds, the low byte of *dwFlags* (0) is used to fill the uninitialized portion of *pszDest* following the terminating null character.
STRSAFE_IGNORE_NULLS	Treat **NULL** string pointers like empty strings (TEXT("")).

0x00000100	
STRSAFE_FILL_ON_FAILURE 0x00000400	If the function fails, the low byte of *dwFlags* (0) is used to fill the entire *pszDest* buffer, and the buffer is null-terminated. In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any pre-existing or truncated string in the destination buffer is overwritten.
STRSAFE_NULL_ON_FAILURE 0x00000800	If the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any pre-existing or truncated string in the destination buffer is overwritten.
STRSAFE_NO_TRUNCATION 0x00001000	If the function fails, *pszDest* is untouched. Nothing is added to the original contents.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Source data was present, the strings were concatenated without truncation, and the resultant destination buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cbDest* is larger than `STRSAFE_MAX_CCH * sizeof(TCHAR)`, an invalid flag was passed, or there are discrepancies between the size of the *pszDest*, *cbDest*, and the amount of material to append in *pszSrc*.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. Depending on the value of *dwFlags*, the destination buffer may contain a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

Compared to the functions it replaces, **StringCbCatNEx** provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCbCatNEx** always null-terminates and never overflows a valid destination buffer, even if the contents of the source string change during the operation.

Behavior is undefined if the strings pointed to by *pszSrc* and *pszDest* overlap.

Neither *pszSrc* nor *pszDest* should be **NULL** unless the **STRSAFE_IGNORE_NULLS** flag is specified, in which case both may be **NULL**. However, an error due to insufficient space may still be returned even though **NULL** values are ignored.

StringCbCatNEx can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCbCatNExA**
TCHAR	TEXT("string")	**StringCbCatNEx**
WCHAR	L"string"	**StringCbCatNExW**

> **Note**
>
> The strsafe.h header defines StringCbCatNEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]

Requirement	Value
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbCatEx

StringCbCatN

StringCchCatNEx

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCbCopyA function (strsafe.h)

Article02/09/2023

Copies one string to another. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCbCopy is a replacement for the following functions:

- strcpy, wcscpy, _tcscpy
- lstrcpy
- StrCpy

Syntax

```
C++

STRSAFEAPI StringCbCopyA(
  [out] STRSAFE_LPSTR  pszDest,
  [in]  size_t         cbDest,
  [in]  STRSAFE_LPCSTR pszSrc
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which receives the copied string.

`[in] cbDest`

Type: **size_t**

The size of the destination buffer, in bytes. This value must consider the length of *pszSrc* plus the terminating null character. The maximum number of bytes allowed is `STRSAFE_MAX_CCH * sizeof(TCHAR)`.

`[in] pszSrc`

Type: **LPCTSTR**

A pointer to a buffer containing the source string. This source string must be null-terminated.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Source data was present, fully copied without truncation, and the resultant destination buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cbDest* is either less than `sizeof(TCHAR)` or larger than the maximum allowed value.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. The destination buffer contains a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

Compared to the functions it replaces, **StringCbCopy** provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCbCopy** always null-terminates and never overflows a valid destination buffer, even if the contents of the source string change during the operation.

Behavior is undefined if the strings pointed to by *pszSrc* and *pszDest* overlap.

Neither *pszSrc* nor *pszDest* should be **NULL**. See StringCbCopyEx if you require the handling of null string pointer values.

StringCbCopy can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use.

Expand table

String Data Type	String Literal	Function
char	"string"	StringCbCopyA
TCHAR	TEXT("string")	StringCbCopy
WCHAR	L"string"	StringCbCopyW

> **Note**
>
> The strsafe.h header defines StringCbCopy as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbCopyEx

StringCchCopy

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCbCopyExA function (strsafe.h)

Article02/09/2023

Copies one string to another. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCbCopyEx adds to the functionality of StringCbCopy by returning a pointer to the end of the destination string as well as the number of bytes left unused in that string. Flags may also be passed to the function for additional control.

StringCbCopyEx is a replacement for the following functions:

- strcpy, wcscpy, _tcscpy
- lstrcpy
- StrCpy

Syntax

C++

```
STRSAFEAPI StringCbCopyExA(
  [out]           STRSAFE_LPSTR  pszDest,
  [in]            size_t         cbDest,
  [in]            STRSAFE_LPCSTR pszSrc,
  [out, optional] STRSAFE_LPSTR  *ppszDestEnd,
  [out, optional] size_t         *pcbRemaining,
  [in]            DWORD          dwFlags
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which receives the copied string.

`[in] cbDest`

Type: **size_t**

The size of the destination buffer, in bytes. This value must consider the length of *pszSrc* plus the terminating null character. The maximum number of bytes allowed is

`STRSAFE_MAX_CCH * sizeof(TCHAR)`.

`[in] pszSrc`

Type: **LPCTSTR**

The source string. This string must be null-terminated.

`[out, optional] ppszDestEnd`

Type: **LPTSTR***

The address of a pointer to the end of *pszDest*. If *ppszDestEnd* is non-**NULL** and any data is copied into the destination buffer, this points to the terminating null character at the end of the string.

`[out, optional] pcbRemaining`

Type: **size_t***

The number of unused bytes in *pszDest*, including those used for the terminating null character. If *pcbRemaining* is **NULL**, the count is not kept or returned.

`[in] dwFlags`

Type: **DWORD**

One or more of the following values.

Expand table

Value	Meaning
STRSAFE_FILL_BEHIND_NULL 0x00000200	If the function succeeds, the low byte of *dwFlags* (0) is used to fill the uninitialized portion of *pszDest* following the terminating null character.
STRSAFE_IGNORE_NULLS 0x00000100	Treat **NULL** string pointers like empty strings (TEXT("")). This flag is useful for emulating functions such as lstrcpy.
STRSAFE_FILL_ON_FAILURE 0x00000400	If the function fails, the low byte of *dwFlags* (0) is used to fill the entire *pszDest* buffer, and the buffer is null-terminated. In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string returned is overwritten.
STRSAFE_NULL_ON_FAILURE 0x00000800	If the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a

	STRSAFE_E_INSUFFICIENT_BUFFER failure, any truncated string is overwritten.
STRSAFE_NO_TRUNCATION 0x00001000	As in the case of **STRSAFE_NULL_ON_FAILURE**, if the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Source data was present, fully copied without truncation, and the resultant destination buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	Either *pszDest* is **NULL** when there is source data present to copy, or an invalid flag was passed.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. Depending on the value of *dwFlags*, the destination buffer may contain a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

Compared to the functions it replaces, **StringCbCopyEx** provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCbCopyEx** always null-terminates and never overflows a valid destination buffer, even if the contents of the source string change during the operation.

Behavior is undefined if the strings pointed to by *pszSrc* and *pszDest* overlap.

Neither *pszSrc* nor *pszDest* should be **NULL** unless the **STRSAFE_IGNORE_NULLS** flag is specified, in which case both may be **NULL**. However, an error due to insufficient space may still be returned even though **NULL** values are ignored.

StringCbCopyEx can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCbCopyExA**
TCHAR	TEXT("string")	**StringCbCopyEx**
WCHAR	L"string"	**StringCbCopyExW**

> **Note**
>
> The strsafe.h header defines StringCbCopyEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbCopy

StringCchCopyEx

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCbCopyNA function (strsafe.h)

Article02/09/2023

Copies the specified number of bytes from one string to another. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCbCopyN is a replacement for the following functions:

- strncpy, wcsncpy, _tcsncpy

Syntax

```
C++

STRSAFEAPI StringCbCopyNA(
  [out] STRSAFE_LPSTR  pszDest,
  [in]  size_t         cbDest,
  [in]  STRSAFE_PCNZCH pszSrc,
  [in]  size_t         cbToCopy
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which receives the copied characters.

`[in] cbDest`

Type: **size_t**

The size of *pszDest*, in bytes. This value must be large enough to hold the copied bytes (the size of *pszSrc* or the value of *cbSrc*, whichever is smaller) and also account for the terminating null character. The maximum number of characters allowed is `STRSAFE_MAX_CCH * sizeof(TCHAR)`.

`[in] pszSrc`

Type: **LPCTSTR**

The source string. This string must be null-terminated.

`[in] cbToCopy`

Type: **size_t**

The maximum number of bytes to be copied from *pszSrc* to *pszDest*.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Source data was present, the data was copied from *pszSrc* without truncation, and the resultant destination buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cbDest* is either larger than `STRSAFE_MAX_CCH * sizeof(TCHAR)`, or the destination buffer is already full.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. The destination buffer contains a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

StringCbCopyN provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCbCopyN** always null-terminates and never overflows a valid destination buffer, even if the contents of the source string change during the operation.

While this routine is meant as a replacement for strncpy, there are differences in behavior. If *cbSrc* is larger than the number of bytes in *pszSrc*, **StringCbCopyN**—unlike

strncpy—does not continue to pad *pszDest* with null characters until *cbSrc* bytes have been copied.

Behavior is undefined if the strings pointed to by *pszSrc* and *pszDest* overlap.

Neither *pszSrc* nor *pszDest* should be **NULL**. See StringCbCopyNEx if you require the handling of null string pointer values.

StringCbCopyN can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use, as shown in the following table.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCbCopyNA**
TCHAR	TEXT("string")	**StringCbCopyN**
WCHAR	L"string"	**StringCbCopyNW**

> **Note**
>
> The strsafe.h header defines StringCbCopyN as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbCopy

StringCbCopyNEx

StringCchCopyN

Feedback

StringCbCopyNExA function (strsafe.h)

Article02/09/2023

Copies the specified number of bytes from one string to another. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCbCopyNEx adds to the functionality of StringCbCopyN by returning a pointer to the end of the destination string as well as the number of bytes left unused in that string. Flags may also be passed to the function for additional control.

StringCbCopyNEx is a replacement for the following functions:

- strncpy, wcsncpy, _tcsncpy

Syntax

C++

```
STRSAFEAPI StringCbCopyNExA(
  [out]           STRSAFE_LPSTR  pszDest,
  [in]            size_t         cbDest,
  [in]            STRSAFE_PCNZCH pszSrc,
  [in]            size_t         cbToCopy,
  [out, optional] STRSAFE_LPSTR  *ppszDestEnd,
  [out, optional] size_t         *pcbRemaining,
  [in]            DWORD          dwFlags
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which receives the copied string.

`[in] cbDest`

Type: **size_t**

The size of *pszDest*, in bytes. This value must be at least large enough to hold the copied bytes (the length of *pszSrc* or the value of *cbSrc*, whichever is smaller) as well as to

account for the terminating null character. The maximum number of bytes allowed is `STRSAFE_MAX_CCH * sizeof(TCHAR)`.

`[in] pszSrc`

Type: **LPCTSTR**

The source string. This string must be null-terminated.

`[in] cbToCopy`

Type: **size_t**

The maximum number of bytes to be copied from *pszSrc* to *pszDest*.

`[out, optional] ppszDestEnd`

Type: **LPTSTR***

The address of a pointer to the end of *pszDest*. If *ppszDestEnd* is non-**NULL** and any data is copied into the destination buffer, this points to the terminating null character at the end of the string.

`[out, optional] pcbRemaining`

Type: **size_t***

The number of unused bytes in *pszDest*, including those used for the terminating null character. If *pcbRemaining* is **NULL**, the count is not kept or returned.

`[in] dwFlags`

Type: **DWORD**

One or more of the following values.

Expand table

Value	Meaning
STRSAFE_FILL_BEHIND_NULL 0x00000200	If the function succeeds, the low byte of *dwFlags* (0) is used to fill the uninitialized portion of *pszDest* following the terminating null character.
STRSAFE_IGNORE_NULLS 0x00000100	Treat **NULL** string pointers like empty strings (TEXT("")).

STRSAFE_FILL_ON_FAILURE 0x00000400	If the function fails, the low byte of *dwFlags* (0) is used to fill the entire *pszDest* buffer, and the buffer is null-terminated. In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string returned is overwritten.
STRSAFE_NULL_ON_FAILURE 0x00000800	If the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.
STRSAFE_NO_TRUNCATION 0x00001000	As in the case of **STRSAFE_NULL_ON_FAILURE**, if the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Source data was present, fully copied without truncation, and the resultant destination buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	Either *pszDest* or *pszSrc* is greater than `STRSAFE_MAX_CCH * sizeof(TCHAR)`, *pszDest* is **NULL** when there is source data present to copy, or an invalid flag was passed.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. Depending on the value of *dwFlags*, the destination buffer may contain a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

StringCbCopyNEx provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCbCopyNEx** always null-terminates and never overflows a valid destination buffer, even if the contents of the source string change during the operation.

While this routine is meant as a replacement for strncpy, there are differences in behavior. If *cbSrc* is larger than the number of bytes in *pszSrc*, **StringCbCopyNEx**—unlike **strncpy**—does not continue to pad *pszDest* with null characters until *cbSrc* bytes have been copied.

Behavior is undefined if the strings pointed to by *pszSrc* and *pszDest* overlap.

Neither *pszSrc* nor *pszDest* should be **NULL** unless the **STRSAFE_IGNORE_NULLS** flag is specified, in which case both may be **NULL**. However, an error due to insufficient space may still be returned even though **NULL** values are ignored.

StringCbCopyNEx can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use, as shown in the following table.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCbCopyNExA**
TCHAR	TEXT("string")	**StringCbCopyNEx**
WCHAR	L"string"	**StringCbCopyNExW**

> **Note**
>
> The strsafe.h header defines StringCbCopyNEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbCopyN

StringCchCopyNEx

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCbGetsA function (strsafe.h)

Article02/09/2023

Gets one line of text from stdin, up to and including the newline character ('\n'). The line of text is copied to the destination buffer, and the newline character is replaced with a null character. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

> **Note** This function can only be used inline.

StringCbGets is a replacement for the following functions:

- gets, _getws, _getts

StringCbGets is not a replacement for **fgets**, which does not replace newline characters with a terminating null character.

Syntax

```
C++
```

```
STRSAFEAPI StringCbGetsA(
  [out] STRSAFE_LPSTR pszDest,
  [in]  size_t        cbDest
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which receives the input.

`[in] cbDest`

Type: **size_t**

The size of the destination buffer, in bytes. This value must be greater than `sizeof(TCHAR)` for the function to succeed. The maximum number of bytes allowed is

`STRSAFE_MAX_CCH * sizeof(TCHAR)`. If *cbDest* is too small to hold the full line of text, the data is truncated.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Data was read from stdin, was copied to the buffer at *pszDest*, and the buffer was null-terminated.
STRSAFE_E_END_OF_FILE	Indicates an error or end-of-file condition. Use feof or ferror to determine which one has occurred.
STRSAFE_E_INVALID_PARAMETER	The value in *cbDest* is larger than the maximum allowed value.
STRSAFE_E_INSUFFICIENT_BUFFER	The value in *cbDest* is `sizeof(TCHAR)` or less.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

StringCbGets provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCbGets** always null-terminates a nonzero-length destination buffer.

The value of *pszDest* should not be **NULL**. See StringCbGetsEx if you require the handling of null string pointer values.

StringCbGets can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use, as shown in the following table.

Expand table

String Data Type	String Literal	Function

char	"string"	**StringCbGetsA**
TCHAR	TEXT("string")	**StringCbGets**
WCHAR	L"string"	**StringCbGetsW**

> **Note**
>
> The strsafe.h header defines StringCbGets as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbGetsEx

StringCchGets

Feedback

StringCbGetsExA function (strsafe.h)

Article02/09/2023

Gets one line of text from stdin, up to and including the newline character ('\n'). The line of text is copied to the destination buffer, and the newline character is replaced with a null character. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

> **Note** This function can only be used inline.

StringCbGetsEx adds to the functionality of StringCbGets by returning a pointer to the end of the destination string as well as the number of bytes left unused in that string. Flags may also be passed to the function for additional control.

StringCbGetsEx is a replacement for the following functions:

- gets, _getws, _getts

StringCbGetsEx is not a replacement for **fgets**, which does not replace newline characters with a terminating null character.

Syntax

C++

```
STRSAFEAPI StringCbGetsExA(
  [out]           STRSAFE_LPSTR pszDest,
  [in]            size_t        cbDest,
  [out, optional] STRSAFE_LPSTR *ppszDestEnd,
  [out, optional] size_t        *pcbRemaining,
  [in]            DWORD         dwFlags
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which is to receive the input.

`[in] cbDest`

Type: **size_t**

The size of the destination buffer, in bytes. This value must be greater than `sizeof(TCHAR)` for the function to succeed. The maximum number of bytes allowed is `STRSAFE_MAX_CCH * sizeof(TCHAR)`. If *cbDest* is too small to hold the full line of text, the data is truncated.

`[out, optional] ppszDestEnd`

Type: **LPTSTR***

The address of a pointer to the end of *pszDest*. If *ppszDestEnd* is non-**NULL** and any data is copied into the destination buffer, this points to the terminating null character at the end of the string.

`[out, optional] pcbRemaining`

Type: **size_t***

The number of unused bytes in *pszDest*, including those used for the terminating null character. If *pcbRemaining* is **NULL**, the count is not kept or returned.

`[in] dwFlags`

Type: **DWORD**

One or more of the following values.

Expand table

Value	Meaning
STRSAFE_FILL_BEHIND_NULL 0x00000200	If the function succeeds, the low byte of *dwFlags* (0) is used to fill the uninitialized portion of *pszDest* following the terminating null character.
STRSAFE_IGNORE_NULLS 0x00000100	Treat **NULL** string pointers like empty strings (TEXT("")). This flag is useful for emulating functions such as lstrcpy.
STRSAFE_FILL_ON_FAILURE 0x00000400	If the function fails, the low byte of *dwFlags* (0) is used to fill the entire *pszDest* buffer, and the buffer is null-terminated. In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string returned is overwritten.

STRSAFE_NULL_ON_FAILURE 0x00000800	If the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.
STRSAFE_NO_TRUNCATION 0x00001000	As in the case of **STRSAFE_NULL_ON_FAILURE**, if the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Data was read from stdin, was copied to the buffer at *pszDest*, and the buffer was null-terminated.
STRSAFE_E_END_OF_FILE	Indicates an error or end-of-file condition. Use feof or ferror to determine which one has occurred.
STRSAFE_E_INVALID_PARAMETER	The value in *cbDest* is larger than the maximum allowed value or an invalid flag was passed.
STRSAFE_E_INSUFFICIENT_BUFFER	The value in *cbDest* is 1 or less.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

StringCbGetsEx provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCbGetsEx** always null-terminates a nonzero-length destination buffer.

The value of *pszDest* should not be **NULL** unless the **STRSAFE_IGNORE_NULLS** flag is specified. However, an error due to insufficient space may still be returned even though **NULL** values are ignored.

StringCbGetsEx can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use, as shown in the following table.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCbGetsExA**
TCHAR	TEXT("string")	**StringCbGetsEx**
WCHAR	L"string"	**StringCbGetsExW**

> **Note**
>
> The strsafe.h header defines StringCbGetsEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbGets

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCbLengthA function (strsafe.h)

Article02/22/2024

Determines whether a string exceeds the specified length, in bytes.

StringCbLength is a replacement for the following functions:

- strlen, wcslen, _tcslen

Syntax

```
C++

STRSAFEAPI StringCbLengthA(
  [in]  STRSAFE_PCNZCH psz,
  [in]  size_t         cbMax,
  [out] size_t         *pcbLength
);
```

Parameters

`[in] psz`

Type: **LPCTSTR**

The string whose length is to be checked.

`[in] cbMax`

Type: **size_t**

The maximum number of bytes allowed in *psz*, including those used for the terminating null character. This value cannot exceed `STRSAFE_MAX_CCH * sizeof(TCHAR)`.

`[out] pcbLength`

Type: **size_t***

The number of bytes in *psz*, excluding those used for the terminating null character. This value is valid only if *pcb* is not **NULL** and the function succeeds.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	The string at *psz* was not **NULL**, and the length of the string (including the terminating null character) is less than or equal to *cbMax* characters.
STRSAFE_E_INVALID_PARAMETER	The value in *psz* is **NULL**, *cbMax* is larger than `STRSAFE_MAX_CCH * sizeof(TCHAR)`, or *psz* is longer than *cbMax*.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

Compared to the functions it replaces, **StringCbLength** is an additional tool for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns.

StringCbLength can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCbLengthA**
TCHAR	TEXT("string")	**StringCbLength**
WCHAR	L"string"	**StringCbLengthW**

UnalignedStringCbLength is an alias for this function.

> **Note**

The strsafe.h header defines StringCbLength as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

StringCchLength

Feedback

StringCbPrintf_lA function (strsafe.h)

Article02/09/2023

Writes formatted data to the specified string. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCbPrintf_l is similar to StringCbPrintf but includes a parameter for locale information.

Syntax

```
C++

STRSAFEAPI StringCbPrintf_lA(
  [out] STRSAFE_LPSTR                                    pszDest,
  [in]  size_t                                           cbDest,
  [in]  _Printf_format_string_params_(2)STRSAFE_LPCSTR pszFormat,
  [in]  _locale_t                                        locale,
        ...
);
```

Parameters

`[out] pszDest`

The destination buffer, which receives the formatted, null-terminated string created from *pszFormat* and its arguments.

`[in] cbDest`

The size of the destination buffer, in bytes. This value must be sufficiently large to accommodate the final formatted string plus the terminating null character. The maximum number of bytes allowed is `STRSAFE_MAX_CCH * sizeof(TCHAR)`.

`[in] pszFormat`

The format string. This string must be null-terminated. For more information, see Format Specification Syntax.

`[in] locale`

The locale object. For more information, see **_create_locale**.

`...`

The arguments to be inserted into the *pszFormat* string.

Return value

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	There was sufficient space for the result to be copied to *pszDest* without truncation, and the buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cbDest* is either 0 or larger than `STRSAFE_MAX_CCH * sizeof(TCHAR)`.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. The destination buffer contains a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Remarks

Behavior is undefined if the strings pointed to by *pszDest*, *pszFormat*, or any argument strings overlap.

Neither *pszFormat* nor *pszDest* should be **NULL**. See StringCbPrintf_lEx if you require the handling of null string pointer values.

In order to use this function, you must define the following macro in your header file, before including StrSafe.h.

```
#define STRSAFE_LOCALE_FUNCTIONS
```

> **Note**
>
> The strsafe.h header defines StringCbPrintf_l as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with

code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCbPrintf_lExA function (strsafe.h)

Article02/09/2023

Writes formatted data to the specified string. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCbPrintf_lEx is similar to StringCbPrintfEx but includes a parameter for locale information.

Syntax

```
C++

STRSAFEAPI StringCbPrintf_lExA(
  [out]           STRSAFE_LPSTR                                    pszDest,
  [in]            size_t                                           cbDest,
  [out]           STRSAFE_LPSTR
*ppszDestEnd,
  [out, optional] size_t
*pcbRemaining,
  [in]            DWORD                                            dwFlags,
  [in]            _Printf_format_string_params_(2)STRSAFE_LPCSTR pszFormat,
  [in]            _locale_t                                        locale,
                  ...
);
```

Parameters

`[out] pszDest`

The destination buffer, which receives the formatted, null-terminated string created from *pszFormat* and its arguments.

`[in] cbDest`

The size of the destination buffer, in bytes. This value must be sufficiently large to accommodate the final formatted string plus the terminating null character. The maximum number of bytes allowed is `STRSAFE_MAX_CCH * sizeof(TCHAR)`.

`[out] ppszDestEnd`

The address of a pointer to the end of *pszDest*. If *ppszDestEnd* is non-**NULL** and any data is copied into the destination buffer, this points to the terminating null character at the

end of the string.

`[out, optional] pcbRemaining`

The number of unused bytes in *pszDest*, including those used for the terminating null character. If *pcbRemaining* is **NULL**, the count is not kept or returned.

`[in] dwFlags`

One or more of the following values.

Expand table

Value	Meaning
STRSAFE_FILL_BEHIND_NULL 0x00000200	If the function succeeds, the low byte of *dwFlags* (0) is used to fill the uninitialized portion of *pszDest* following the terminating null character.
STRSAFE_IGNORE_NULLS 0x00000100	Treat **NULL** string pointers like empty strings (TEXT("")).
STRSAFE_FILL_ON_FAILURE 0x00000400	If the function fails, the low byte of *dwFlags* (0) is used to fill the entire *pszDest* buffer, and the buffer is null-terminated. In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string returned is overwritten.
STRSAFE_NULL_ON_FAILURE 0x00000800	If the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.
STRSAFE_NO_TRUNCATION 0x00001000	As in the case of **STRSAFE_NULL_ON_FAILURE**, if the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.

`[in] pszFormat`

The format string. This string must be null-terminated. For more information, see Format Specification Syntax.

`[in] locale`

The locale object. For more information, see **_create_locale**.

`...`

The arguments to be inserted into the *pszFormat* string.

Return value

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	There was sufficient space for the result to be copied to *pszDest* without truncation, and the buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cbDest* is either 0 or larger than `STRSAFE_MAX_CCH * sizeof(TCHAR)`, or the destination buffer is already full.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. Depending on the value of *dwFlags*, the destination buffer may contain a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Remarks

Behavior is undefined if the strings pointed to by *pszDest*, *pszFormat*, or any argument strings overlap.

Neither *pszFormat* nor *pszDest* should be **NULL** unless the **STRSAFE_IGNORE_NULLS** flag is specified, in which case both may be **NULL**. However, an error due to insufficient space may still be returned even though **NULL** values are ignored.

In order to use this function, you must define the following macro in your header file, before including StrSafe.h.

```
#define STRSAFE_LOCALE_FUNCTIONS
```

> **Note**
>
> The strsafe.h header defines StringCbPrintf_lEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the

UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCbPrintfA function (strsafe.h)

Article02/09/2023

Writes formatted data to the specified string. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCbPrintf is a replacement for the following functions:

- sprintf, swprintf, _stprintf
- wsprintf
- wnsprintf
- _snprintf, _snwprintf, _sntprintf

Syntax

```
C++

STRSAFEAPI StringCbPrintfA(
  [out] STRSAFE_LPSTR  pszDest,
  [in]  size_t         cbDest,
  [in]  STRSAFE_LPCSTR pszFormat,
        ...
);
```

Parameters

`[out] pszDest`

Type: **LPSTR**

The destination buffer, which receives the formatted, null-terminated string created from *pszFormat* and its arguments.

`[in] cbDest`

Type: **size_t**

The size of the destination buffer, in bytes. This value must be sufficiently large to accommodate the final formatted string plus the terminating null character. The maximum number of bytes allowed is `STRSAFE_MAX_CCH * sizeof(TCHAR)`.

`[in] pszFormat`

Type: **LPCSTR**

The format string. This string must be null-terminated. For more information, see Format Specification Syntax.

`...`

The arguments to be inserted into the *pszFormat* string.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	There was sufficient space for the result to be copied to *pszDest* without truncation, and the buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cbDest* is either 0 or larger than `STRSAFE_MAX_CCH * sizeof(TCHAR)`.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. The destination buffer contains a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

Compared to the functions it replaces, **StringCbPrintf** provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCbPrintf** always null-terminates a nonzero-length destination buffer.

Behavior is undefined if the strings pointed to by *pszDest*, *pszFormat*, or any argument strings overlap.

Neither *pszFormat* nor *pszDest* should be **NULL**. See StringCbPrintfEx if you require the handling of null string pointer values.

StringCbPrintf can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCbPrintfA**
TCHAR	TEXT("string")	**StringCbPrintf**
WCHAR	L"string"	**StringCbPrintfW**

Examples

The following example shows a basic use of **StringCbPrintf**, using four arguments.

C++

```
int const arraysize = 30;
TCHAR pszDest[arraysize];
size_t cbDest = arraysize * sizeof(TCHAR);

LPCTSTR pszFormat = TEXT("%s %d + %d = %d.");
TCHAR* pszTxt = TEXT("The answer is");

HRESULT hr = StringCbPrintf(pszDest, cbDest, pszFormat, pszTxt, 1, 2, 3);

// The resultant string at pszDest is "The answer is 1 + 2 = 3."
```

> ⓘ Note
>
> The strsafe.h header defines StringCbPrintf as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbPrintfEx

StringCbVPrintf

StringCchPrintf

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCbPrintfExA function (strsafe.h)

Article02/09/2023

Writes formatted data to the specified string. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCbPrintfEx adds to the functionality of StringCbPrintf by returning a pointer to the end of the destination string as well as the number of bytes left unused in that string. Flags may also be passed to the function for additional control.

StringCbPrintfEx is a replacement for the following functions:

- sprintf, swprintf, _stprintf
- wsprintf
- wnsprintf
- _snprintf, _snwprintf, _sntprintf

Syntax

```
C++

STRSAFEAPI StringCbPrintfExA(
  [out]           STRSAFE_LPSTR  pszDest,
  [in]            size_t         cbDest,
  [out, optional] STRSAFE_LPSTR  *ppszDestEnd,
  [out, optional] size_t         *pcbRemaining,
  [in]            DWORD          dwFlags,
  [in]            STRSAFE_LPCSTR pszFormat,
                  ...
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which receives the formatted, null-terminated string created from *pszFormat* and its arguments.

`[in] cbDest`

Type: **size_t**

The size of the destination buffer, in bytes. This value must be sufficiently large to accommodate the final formatted string plus the terminating null character. The maximum number of bytes allowed is `STRSAFE_MAX_CCH * sizeof(TCHAR)`.

`[out, optional] ppszDestEnd`

Type: **LPTSTR***

The address of a pointer to the end of *pszDest*. If *ppszDestEnd* is non-**NULL** and any data is copied into the destination buffer, this points to the terminating null character at the end of the string.

`[out, optional] pcbRemaining`

Type: **size_t***

The number of unused bytes in *pszDest*, including those used for the terminating null character. If *pcbRemaining* is **NULL**, the count is not kept or returned.

`[in] dwFlags`

Type: **DWORD**

One or more of the following values.

Expand table

Value	Meaning
STRSAFE_FILL_BEHIND_NULL 0x00000200	If the function succeeds, the low byte of *dwFlags* (0) is used to fill the uninitialized portion of *pszDest* following the terminating null character.
STRSAFE_IGNORE_NULLS 0x00000100	Treat **NULL** string pointers like empty strings (TEXT("")).
STRSAFE_FILL_ON_FAILURE 0x00000400	If the function fails, the low byte of *dwFlags* (0) is used to fill the entire *pszDest* buffer, and the buffer is null-terminated. In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string returned is overwritten.
STRSAFE_NULL_ON_FAILURE 0x00000800	If the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.

STRSAFE_NO_TRUNCATION 0x00001000	As in the case of **STRSAFE_NULL_ON_FAILURE**, if the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.

`[in] pszFormat`

Type: **LPCTSTR**

The format string. This string must be null-terminated. For more information, see Format Specification Syntax.

`...`

The arguments to be inserted into the *pszFormat* string.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	There was sufficient space for the result to be copied to *pszDest* without truncation, and the buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cbDest* is either 0 or larger than `STRSAFE_MAX_CCH * sizeof(TCHAR)`, or the destination buffer is already full.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. Depending on the value of *dwFlags*, the destination buffer may contain a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

Compared to the functions it replaces, **StringCbPrintfEx** provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCbPrintfEx** always null-terminates a nonzero-length destination buffer.

Behavior is undefined if the strings pointed to by *pszDest*, *pszFormat*, or any argument strings overlap.

Neither *pszFormat* nor *pszDest* should be **NULL** unless the **STRSAFE_IGNORE_NULLS** flag is specified, in which case both may be **NULL**. However, an error due to insufficient space may still be returned even though **NULL** values are ignored.

StringCbPrintfEx can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCbPrintfExA**
TCHAR	TEXT("string")	**StringCbPrintfEx**
WCHAR	L"string"	**StringCbPrintfExW**

> **Note**
>
> The strsafe.h header defines StringCbPrintfEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]

Requirement	Value
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbVPrintfEx

StringCchPrintf

StringCchPrintfEx

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCbVPrintf_lA function (strsafe.h)

Article02/09/2023

Writes formatted data to the specified string using a pointer to a list of arguments. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCbVPrintf_l is similar to StringCbVPrintf but includes a parameter for locale information.

Syntax

```
C++

STRSAFEAPI StringCbVPrintf_lA(
  [out] STRSAFE_LPSTR                                  pszDest,
  [in]  size_t                                         cbDest,
  [in]  _Printf_format_string_params_(2)STRSAFE_LPCSTR pszFormat,
  [in]  _locale_t                                      locale,
  [in]  va_list                                        argList
);
```

Parameters

`[out] pszDest`

The destination buffer, which receives the formatted, null-terminated string created from *pszFormat* and *argList*.

`[in] cbDest`

The size of the destination buffer, in bytes. This value must be sufficiently large to accommodate the final formatted string plus the terminating null character. The maximum number of bytes allowed is `STRSAFE_MAX_CCH * sizeof(TCHAR)`.

`[in] pszFormat`

The format string. This string must be null-terminated. For more information, see Format Specification Syntax.

`[in] locale`

The locale object. For more information, see **_create_locale**.

`[in] argList`

The arguments to be inserted into the *pszFormat* string.

Return value

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	There was sufficient space for the result to be copied to *pszDest* without truncation, and the buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cbDest* is either 0 or larger than `STRSAFE_MAX_CCH * sizeof(TCHAR)`.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. The destination buffer contains a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Remarks

For more information on va_lists, see the conventions defined in Stdarg.h.

Behavior is undefined if the strings pointed to by *pszDest*, *pszFormat*, or any argument strings overlap.

Neither *pszFormat* nor *pszDest* should be **NULL**. See StringCbVPrintf_lEx if you require the handling of null string pointer values.

In order to use this function, you must define the following macro in your header file, before including StrSafe.h.

```
#define STRSAFE_LOCALE_FUNCTIONS
```

> **Note**

The strsafe.h header defines StringCbVPrintf_l as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCbVPrintf_lExA function (strsafe.h)

Article02/09/2023

Writes formatted data to the specified string using a pointer to a list of arguments. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCbVPrintf_lEx is similar to StringCbVPrintfEx but includes a parameter for locale information.

Syntax

```
C++

STRSAFEAPI StringCbVPrintf_lExA(
  [out]           STRSAFE_LPSTR                                    pszDest,
  [in]            size_t                                           cbDest,
  [out]           STRSAFE_LPSTR
*ppszDestEnd,
  [out, optional] size_t
*pcbRemaining,
  [in]            DWORD                                            dwFlags,
  [in]            _Printf_format_string_params_(2)STRSAFE_LPCSTR pszFormat,
  [in]            _locale_t                                        locale,
  [in]            va_list                                          argList
);
```

Parameters

`[out] pszDest`

The destination buffer, which receives the formatted, null-terminated string created from *pszFormat* and *argList*.

`[in] cbDest`

The size of the destination buffer, in bytes. This value must be sufficiently large to accommodate the final formatted string plus the terminating null character. The maximum number of bytes allowed is `STRSAFE_MAX_CCH * sizeof(TCHAR)`.

`[out] ppszDestEnd`

The address of a pointer to the end of *pszDest*. If *ppszDestEnd* is non-**NULL** and any data is copied into the destination buffer, this points to the terminating null character at the end of the string.

`[out, optional] pcbRemaining`

The number of unused bytes in *pszDest*, including those used for the terminating null character. If *pcbRemaining* is **NULL**, the count is not kept or returned.

`[in] dwFlags`

One or more of the following values.

Expand table

Value	Meaning
STRSAFE_FILL_BEHIND_NULL 0x00000200	If the function succeeds, the low byte of *dwFlags* (0) is used to fill the uninitialized portion of *pszDest* following the terminating null character.
STRSAFE_IGNORE_NULLS 0x00000100	Treat **NULL** string pointers like empty strings (TEXT("")).
STRSAFE_FILL_ON_FAILURE 0x00000400	If the function fails, the low byte of *dwFlags* (0) is used to fill the entire *pszDest* buffer, and the buffer is null-terminated. In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string returned is overwritten.
STRSAFE_NULL_ON_FAILURE 0x00000800	If the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.
STRSAFE_NO_TRUNCATION 0x00001000	As in the case of **STRSAFE_NULL_ON_FAILURE**, if the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.

`[in] pszFormat`

The format string. This string must be null-terminated. For more information, see Format Specification Syntax.

`[in] locale`

The locale object. For more information, see **_create_locale**.

`[in] argList`

The arguments to be inserted into the *pszFormat* string.

Return value

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	There is sufficient space for the result to be copied to *pszDest* without truncation, and the buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cbDest* is either 0 or larger than `STRSAFE_MAX_CCH * sizeof(TCHAR)`, or the destination buffer is already full.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. Depending on the value of *dwFlags*, the destination buffer may contain a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Remarks

For more information on va_lists, see the conventions defined in Stdarg.h.

Behavior is undefined if the strings pointed to by *pszDest*, *pszFormat*, or any argument strings overlap.

Neither *pszFormat* nor *pszDest* should be **NULL** unless the **STRSAFE_IGNORE_NULLS** flag is specified, in which case both may be **NULL**. However, an error due to insufficient space may still be returned even though **NULL** values are ignored.

In order to use this function, you must define the following macro in your header file, before including StrSafe.h.

```
#define STRSAFE_LOCALE_FUNCTIONS
```

> **Note**
>
> The strsafe.h header defines StringCbVPrintf_lEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

Feedback

StringCbVPrintfA function (strsafe.h)

Article02/09/2023

Writes formatted data to the specified string using a pointer to a list of arguments. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCbVPrintf is a replacement for the following functions:

- vsprintf, vswprintf, _vstprintf
- vsnprintf, _vsnwprintf, _vsntprintf
- wvsprintf
- wvnsprintf

Syntax

```
C++
```

```
STRSAFEAPI StringCbVPrintfA(
  [out] STRSAFE_LPSTR  pszDest,
  [in]  size_t         cbDest,
  [in]  STRSAFE_LPCSTR pszFormat,
  [in]  va_list        argList
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which receives the formatted, null-terminated string created from *pszFormat* and *argList*.

`[in] cbDest`

Type: **size_t**

The size of the destination buffer, in bytes. This value must be sufficiently large to accommodate the final formatted string plus the terminating null character. The maximum number of bytes allowed is `STRSAFE_MAX_CCH * sizeof(TCHAR)`.

`[in] pszFormat`

Type: **LPCTSTR**

The format string. This string must be null-terminated. For more information, see Format Specification Syntax.

`[in] argList`

Type: **va_list**

The arguments to be inserted into the *pszFormat* string.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	There was sufficient space for the result to be copied to *pszDest* without truncation, and the buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cbDest* is either 0 or larger than `STRSAFE_MAX_CCH * sizeof(TCHAR)`.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. The destination buffer contains a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

StringCbVPrintf provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCbVPrintf** always null-terminates a nonzero-length destination buffer.

For more information on va_lists, see the conventions defined in Stdarg.h.

Behavior is undefined if the strings pointed to by *pszDest*, *pszFormat*, or any argument strings overlap.

Neither *pszFormat* nor *pszDest* should be **NULL**. See StringCbVPrintfEx if you require the handling of null string pointer values.

StringCbVPrintf can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use, as shown in the following table.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCbVPrintfA**
TCHAR	TEXT("string")	**StringCbVPrintf**
WCHAR	L"string"	**StringCbVPrintfW**

> **Note**
>
> The strsafe.h header defines StringCbVPrintf as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbPrintf

StringCbVPrintfEx

StringCchVPrintf

Feedback

StringCbVPrintfExA function (strsafe.h)

Article02/09/2023

Writes formatted data to the specified string using a pointer to a list of arguments. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCbVPrintfEx adds to the functionality of StringCbVPrintf by returning a pointer to the end of the destination string as well as the number of bytes left unused in that string. Flags may also be passed to the function for additional control.

StringCbVPrintfEx is a replacement for the following functions:

- vsprintf, vswprintf, _vstprintf
- vsnprintf, _vsnwprintf, _vsntprintf
- wvsprintf
- wvnsprintf

Syntax

C++

```
STRSAFEAPI StringCbVPrintfExA(
  [out]           STRSAFE_LPSTR  pszDest,
  [in]            size_t         cbDest,
  [out, optional] STRSAFE_LPSTR  *ppszDestEnd,
  [out, optional] size_t         *pcbRemaining,
  [in]            DWORD          dwFlags,
  [in]            STRSAFE_LPCSTR pszFormat,
  [in]            va_list        argList
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which receives the formatted, null-terminated string created from *pszFormat* and *argList*.

`[in] cbDest`

Type: **size_t**

The size of the destination buffer, in bytes. This value must be sufficiently large to accommodate the final formatted string plus the terminating null character. The maximum number of bytes allowed is `STRSAFE_MAX_CCH * sizeof(TCHAR)`.

`[out, optional] ppszDestEnd`

Type: **LPTSTR***

The address of a pointer to the end of *pszDest*. If *ppszDestEnd* is non-**NULL** and any data is copied into the destination buffer, this points to the terminating null character at the end of the string.

`[out, optional] pcbRemaining`

Type: **size_t***

The number of unused bytes in *pszDest*, including those used for the terminating null character. If *pcbRemaining* is **NULL**, the count is not kept or returned.

`[in] dwFlags`

Type: **DWORD**

One or more of the following values.

Expand table

Value	Meaning
STRSAFE_FILL_BEHIND_NULL 0x00000200	If the function succeeds, the low byte of *dwFlags* (0) is used to fill the uninitialized portion of *pszDest* following the terminating null character.
STRSAFE_IGNORE_NULLS 0x00000100	Treat **NULL** string pointers like empty strings (TEXT("")).
STRSAFE_FILL_ON_FAILURE 0x00000400	If the function fails, the low byte of *dwFlags* (0) is used to fill the entire *pszDest* buffer, and the buffer is null-terminated. In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string returned is overwritten.
STRSAFE_NULL_ON_FAILURE 0x00000800	If the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.

STRSAFE_NO_TRUNCATION 0x00001000	As in the case of **STRSAFE_NULL_ON_FAILURE**, if the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.

`[in] pszFormat`

Type: **LPCTSTR**

The format string. This string must be null-terminated. For more information, see Format Specification Syntax.

`[in] argList`

Type: **va_list**

The arguments to be inserted into the *pszFormat* string.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	There is sufficient space for the result to be copied to *pszDest* without truncation, and the buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cbDest* is either 0 or larger than `STRSAFE_MAX_CCH * sizeof(TCHAR)`, or the destination buffer is already full.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. Depending on the value of *dwFlags*, the destination buffer may contain a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

StringCbVPrintfEx provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCbVPrintfEx** always null-terminates a nonzero-length destination buffer.

For more information on va_lists, see the conventions defined in Stdarg.h.

Behavior is undefined if the strings pointed to by *pszDest*, *pszFormat*, or any argument strings overlap.

Neither *pszFormat* nor *pszDest* should be **NULL** unless the **STRSAFE_IGNORE_NULLS** flag is specified, in which case both may be **NULL**. However, an error due to insufficient space may still be returned even though **NULL** values are ignored.

StringCbVPrintfEx can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use, as shown in the following table.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCbVPrintfExA**
TCHAR	TEXT("string")	**StringCbVPrintfEx**
WCHAR	L"string"	**StringCbVPrintfExW**

> **Note**
>
> The strsafe.h header defines StringCbVPrintfEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbPrintfEx

StringCbVPrintf

StringCchVPrintfEx

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCchCatA function (strsafe.h)

Article02/09/2023

Concatenates one string to another string. The size of the destination buffer is provided to the function to ensure that **StringCchCat** does not write past the end of this buffer.

StringCchCat is a replacement for the following functions:

- strcat, wcscat, _tcsat
- lstrcat
- StrCat
- StrCatBuff

Syntax

```
C++

STRSAFEAPI StringCchCatA(
  [in, out] STRSAFE_LPSTR  pszDest,
  [in]      size_t         cchDest,
  [in]      STRSAFE_LPCSTR pszSrc
);
```

Parameters

`[in, out] pszDest`

Type: **LPTSTR**

The destination buffer, which contains the string to which *pszSrc* is to be concatenated, and that will receive the entire resultant string. The string at *pszSrc* is added to the end of the string at *pszDest*.

`[in] cchDest`

Type: **size_t**

The size of the destination buffer, in characters. This value must be greater than or equal the length of *pszSrc* plus the length of *pszDest* plus 1 to account for both strings and the terminating null character. The maximum number of characters allowed is **STRSAFE_MAX_CCH**.

`[in] pszSrc`

Type: **LPCTSTR**

The source string that is to be concatenated to the end of *pszDest*. This string must be null-terminated.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Source data was present, the strings were fully concatenated without truncation, and the resultant destination buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cchDest* is either 0 or larger than **STRSAFE_MAX_CCH**, or the destination buffer is already full.
STRSAFE_E_INSUFFICIENT_BUFFER	The concatenation operation failed due to insufficient buffer space. The destination buffer contains a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

StringCchCat provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCchCat** always null-terminates and never overflows a valid destination buffer, even if the contents of the source string change during the operation.

Behavior is undefined if the strings pointed to by *pszSrc* and *pszDest* overlap.

Neither *pszSrc* nor *pszDest* should be **NULL**. See StringCchCatEx if you require the handling of null string pointer values.

StringCchCat can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use, as shown in the following table.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCchCatA**
TCHAR	TEXT("string")	**StringCchCat**
WCHAR	L"string"	**StringCchCatW**

> **Note**
>
> The strsafe.h header defines StringCchCat as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbCat

StringCchCatEx

StringCchCatN

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCchCatExA function (strsafe.h)

Article02/09/2023

Concatenates one string to another string. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCchCatEx adds to the functionality of StringCchCat by returning a pointer to the end of the destination string as well as the number of characters left unused in that string. Flags may also be passed to the function for additional control.

StringCchCatEx is a replacement for the following functions:

- strcat, wcscat, _tcsat
- lstrcat
- StrCat
- StrCatBuff

Syntax

```
C++

STRSAFEAPI StringCchCatExA(
  [in, out]       STRSAFE_LPSTR  pszDest,
  [in]            size_t         cchDest,
  [in]            STRSAFE_LPCSTR pszSrc,
  [out, optional] STRSAFE_LPSTR  *ppszDestEnd,
  [out, optional] size_t         *pcchRemaining,
  [in]            DWORD          dwFlags
);
```

Parameters

`[in, out] pszDest`

Type: **LPTSTR**

The destination buffer, which contains the string that is to be concatenated to *pszSrc*, and that will receive the entire resultant string. The string at *pszSrc* is added to the end of the string at *pszDest*.

`[in] cchDest`

Type: **size_t**

The size of the destination buffer, in characters. This value must equal the length of *pszSrc* plus the length of *pszDest* plus 1 to account for both strings and the terminating null character. The maximum number of characters allowed is **STRSAFE_MAX_CCH**.

`[in] pszSrc`

Type: **LPCTSTR**

The source string that is to be concatenated to the end of *pszDest*. This string must be null-terminated.

`[out, optional] ppszDestEnd`

Type: **LPTSTR***

The address of a pointer to the end of *pszDest*. If *ppszDestEnd* is non-**NULL** and any data is appended to the destination buffer, this points to the terminating null character at the end of the string.

`[out, optional] pcchRemaining`

Type: **size_t***

The number of unused characters in *pszDest*, including the terminating null character. If *pcchRemaining* is **NULL**, the count is not kept or returned.

`[in] dwFlags`

Type: **DWORD**

One or more of the following values.

Expand table

Value	Meaning
STRSAFE_FILL_BEHIND_NULL 0x00000200	If the function succeeds, the low byte of *dwFlags* (0) is used to fill the uninitialized portion of *pszDest* following the terminating null character.
STRSAFE_IGNORE_NULLS 0x00000100	Treat **NULL** string pointers like empty strings (TEXT("")). This flag is useful for emulating functions such as lstrcpy.
STRSAFE_FILL_ON_FAILURE 0x00000400	If the function fails, the low byte of *dwFlags* (0) is used to fill the entire *pszDest* buffer, and the buffer is null-terminated. In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any pre-

	existing or truncated string in the destination buffer is overwritten.
STRSAFE_NULL_ON_FAILURE 0x00000800	If the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any pre-existing or truncated string in the destination buffer is overwritten.
STRSAFE_NO_TRUNCATION 0x00001000	If the function fails, *pszDest* is untouched. Nothing is added to the original contents.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Source data was present, the strings were fully concatenated without truncation, and the resultant destination buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cchDest* is either 0 or larger than **STRSAFE_MAX_CCH**, or the destination buffer is already full.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. Depending on the value of *dwFlags*, the destination buffer may contain a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

StringCchCatEx provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns.

StringCchCatEx always null-terminates and never overflows a valid destination buffer, even if the contents of the source string change during the operation.

Behavior is undefined if the strings pointed to by *pszSrc* and *pszDest* overlap.

Neither *pszSrc* nor *pszDest* should be **NULL** unless the **STRSAFE_IGNORE_NULLS** flag is specified, in which case both may be **NULL**. However, an error due to insufficient space may still be returned even though **NULL** values are ignored.

StringCchCatEx can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use, as shown in the following table.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCchCatExA**
TCHAR	TEXT("string")	**StringCchCatEx**
WCHAR	L"string"	**StringCchCatExW**

> **Note**
>
> The strsafe.h header defines StringCchCatEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows

Requirement	Value
Header	strsafe.h

See also

Reference

StringCbCatEx

StringCchCat

StringCchCatNEx

Feedback

StringCchCatNA function (strsafe.h)

Article02/09/2023

Concatenates the specified number of characters from one string to another string. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCchCatN is a replacement for the following functions:

- strncat
- StrNCat

Syntax

```
C++

STRSAFEAPI StringCchCatNA(
  [in, out] STRSAFE_LPSTR  pszDest,
  [in]      size_t         cchDest,
  [in]      STRSAFE_PCNZCH pszSrc,
  [in]      size_t         cchToAppend
);
```

Parameters

`[in, out] pszDest`

Type: **LPTSTR**

The destination buffer, which contains the string that is to be concatenated with *pszSrc*, and will receive the entire resultant string. The string at *pszSrc*, up to *cchMaxAppend* characters, is added to the end of the string at *pszDest*.

`[in] cchDest`

Type: **size_t**

The size of the destination buffer, in characters. This value must equal the length of *pszSrc* plus either the length of *pszDest* or *cchMaxAppend* (whichever is smaller). To this sum add 1 to account for the terminating null character. The maximum number of characters allowed is **STRSAFE_MAX_CCH**.

`[in] pszSrc`

Type: **LPCTSTR**

The source string that is concatenated to the end of *pszDest*. This string must be null-terminated.

`[in] cchToAppend`

Type: **size_t**

The maximum number of characters to be appended to *pszDest*.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Source data was present, the strings were concatenated without truncation, and the resultant destination buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cchDest* is either larger than **STRSAFE_MAX_CCH**, or the destination buffer is already full.
STRSAFE_E_INSUFFICIENT_BUFFER	The concatenation operation failed due to insufficient buffer space. The destination buffer contains a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

Compared to the functions it replaces, **StringCchCatN** provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCchCatN** always null-terminates and

never overflows a valid destination buffer, even if the contents of the source string change during the operation.

Behavior is undefined if the strings pointed to by *pszSrc* and *pszDest* overlap.

Neither *pszSrc* nor *pszDest* should be **NULL**. See StringCchCatNEx if you require the handling of null string pointer values.

StringCchCatN can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCchCatNA**
TCHAR	TEXT("string")	**StringCchCatN**
WCHAR	L"string"	**StringCchCatNW**

> **Note**
>
> The strsafe.h header defines StringCchCatN as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbCatN

StringCchCat

StringCchCatNEx

Feedback

StringCchCatNExA function (strsafe.h)

Article02/09/2023

Concatenates the specified number of characters from one string to another string. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCchCatNEx adds to the functionality of StringCchCatN by returning a pointer to the end of the destination string as well as the number of characters left unused in that string. Flags may also be passed to the function for additional control.

StringCchCatNEx is a replacement for the following functions:

- strncat
- StrNCat

Syntax

```
C++
```

```
STRSAFEAPI StringCchCatNExA(
  [in, out]       STRSAFE_LPSTR  pszDest,
  [in]            size_t         cchDest,
  [in]            STRSAFE_PCNZCH pszSrc,
  [in]            size_t         cchToAppend,
  [out, optional] STRSAFE_LPSTR  *ppszDestEnd,
  [out, optional] size_t         *pcchRemaining,
  [in]            DWORD          dwFlags
);
```

Parameters

`[in, out] pszDest`

Type: **LPTSTR**

The destination buffer, which contains the string that is to be concatenated with *pszSrc*, and will receive the entire resultant string. The string at *pszSrc* is added to the end of the string at *pszDest*.

`[in] cchDest`

Type: **size_t**

The size of the destination buffer, in characters. This value must equal the length of *pszSrc* plus the length of *pszDest* plus 1 to account for the terminating null character. The maximum number of characters allowed is **STRSAFE_MAX_CCH**.

`[in] pszSrc`

Type: **LPCTSTR**

The source string that is concatenated to the end of *pszDest*. This string must be null-terminated.

`[in] cchToAppend`

Type: **size_t**

The maximum number of characters to be appended to *pszDest*.

`[out, optional] ppszDestEnd`

Type: **LPTSTR***

The address of a pointer to the end of *pszDest*. If *ppszDestEnd* is non-**NULL** and any data is appended to the destination buffer, this points to the terminating null character at the end of the string.

`[out, optional] pcchRemaining`

Type: **size_t***

The number of unused characters in *pszDest*, including the terminating null character. If *pcchRemaining* is **NULL**, the count is not kept or returned.

`[in] dwFlags`

Type: **DWORD**

One or more of the following values.

Expand table

Value	Meaning
STRSAFE_FILL_BEHIND_NULL 0x00000200	If the function succeeds, the low byte of *dwFlags* (0) is used to fill the uninitialized portion of *pszDest* following the terminating null character.
STRSAFE_IGNORE_NULLS	Treat **NULL** string pointers like empty strings (TEXT("")).

0x00000100	
STRSAFE_FILL_ON_FAILURE 0x00000400	If the function fails, the low byte of *dwFlags* (0) is used to fill the entire *pszDest* buffer, and the buffer is null-terminated. In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any pre-existing or truncated string in the destination buffer is overwritten.
STRSAFE_NULL_ON_FAILURE 0x00000800	If the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any pre-existing or truncated string in the destination buffer is overwritten.
STRSAFE_NO_TRUNCATION 0x00001000	If the function fails, *pszDest* is untouched. Nothing is added to the original contents.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Source data was present, the strings were concatenated without truncation, and the resultant destination buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cchDest* is larger than **STRSAFE_MAX_CCH**, an invalid flag was passed, or there are discrepancies between the size of the *pszDest*, *cchDest*, and the amount of material to append in *pszSrc*.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. Depending on the value of *dwFlags*, the destination buffer may contain a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

Compared to the functions it replaces, **StringCchCatNEx** provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCchCatNEx** always null-terminates and never overflows a valid destination buffer, even if the contents of the source string change during the operation.

Behavior is undefined if the strings pointed to by *pszSrc* and *pszDest* overlap.

Neither *pszSrc* nor *pszDest* should be **NULL** unless the **STRSAFE_IGNORE_NULLS** flag is specified, in which case both may be **NULL**. However, an error due to insufficient space may still be returned even though **NULL** values are ignored.

StringCchCatNEx can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCchCatNExA**
TCHAR	TEXT("string")	**StringCchCatNEx**
WCHAR	L"string"	**StringCchCatNExW**

> **Note**
>
> The strsafe.h header defines StringCchCatNEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]

Requirement	Value
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbCatNEx

StringCchCatEx

StringCchCatN

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCchCopyA function (strsafe.h)

Article02/09/2023

Copies one string to another. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCchCopy is a replacement for the following functions:

- strcpy, wcscpy, _tcscpy
- lstrcpy
- StrCpy

Syntax

```
C++

STRSAFEAPI StringCchCopyA(
  [out] STRSAFE_LPSTR  pszDest,
  [in]  size_t         cchDest,
  [in]  STRSAFE_LPCSTR pszSrc
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which receives the copied string.

`[in] cchDest`

Type: **size_t**

The size of the destination buffer, in characters. This value must equal the length of *pszSrc* plus 1 to account for the copied source string and the terminating null character. The maximum number of characters allowed is **STRSAFE_MAX_CCH**.

`[in] pszSrc`

Type: **LPCTSTR**

The source string. This string must be null-terminated.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Source data was present, fully copied without truncation, and the resultant destination buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cchDest* is either 0 or larger than **STRSAFE_MAX_CCH**.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. The destination buffer contains a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

Compared to the functions it replaces, **StringCchCopy** provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCchCopy** always null-terminates and never overflows a valid destination buffer, even if the contents of the source string change during the operation.

Behavior is undefined if the strings pointed to by *pszSrc* and *pszDest* overlap.

Neither *pszSrc* nor *pszDest* should be **NULL**. See StringCchCopyEx if you require the handling of null string pointer values.

StringCchCopy can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use.

Expand table

String Data Type	String Literal	Function

char	"string"	**StringCchCopyA**
TCHAR	TEXT("string")	**StringCchCopy**
WCHAR	L"string"	**StringCchCopyW**

> **Note**
>
> The strsafe.h header defines StringCchCopy as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbCopy

StringCchCopyEx

Feedback

Was this page helpful? Yes No

StringCchCopyExA function (strsafe.h)

Article02/09/2023

Copies one string to another. The size of the destination buffer is provided to the function to ensure that it does not write past the end of the buffer.

StringCchCopyEx adds to the functionality of StringCchCopy by returning a pointer to the end of the destination string as well as the number of characters left unused in that string. Flags may also be passed to this function for additional control.

StringCchCopyEx is a replacement for the following functions:

- strcpy, wcscpy, _tcscpy
- lstrcpy
- StrCpy

Syntax

C++

```
STRSAFEAPI StringCchCopyExA(
  [out]           STRSAFE_LPSTR  pszDest,
  [in]            size_t         cchDest,
  [in]            STRSAFE_LPCSTR pszSrc,
  [out, optional] STRSAFE_LPSTR  *ppszDestEnd,
  [out, optional] size_t         *pcchRemaining,
  [in]            DWORD          dwFlags
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which receives the copied string.

`[in] cchDest`

Type: **size_t**

The size of the destination buffer, in characters. This value must equal the length of *pszSrc* plus 1 to account for the copied source string and the terminating null character.

The maximum number of characters allowed is **STRSAFE_MAX_CCH**.

`[in] pszSrc`

Type: **LPCTSTR**

The source string. This string must be null-terminated.

`[out, optional] ppszDestEnd`

Type: **LPTSTR***

The address of a pointer to the end of *pszDest*. If *ppszDestEnd* is non-**NULL** and any data is copied into the destination buffer, this points to the terminating null character at the end of the string.

`[out, optional] pcchRemaining`

Type: **size_t***

The number of unused characters in *pszDest*, including the terminating null character. If *pcchRemaining* is **NULL**, the count is not kept or returned.

`[in] dwFlags`

Type: **DWORD**

One or more of the following values.

Expand table

Value	Meaning
STRSAFE_FILL_BEHIND_NULL 0x00000200	If the function succeeds, the low byte of *dwFlags* (0) is used to fill the uninitialized portion of *pszDest* following the terminating null character.
STRSAFE_IGNORE_NULLS 0x00000100	Treat **NULL** string pointers like empty strings (TEXT("")). This flag is useful for emulating functions such as lstrcpy.
STRSAFE_FILL_ON_FAILURE 0x00000400	If the function fails, the low byte of *dwFlags* (0) is used to fill the entire *pszDest* buffer, and the buffer is null-terminated. In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string returned is overwritten.
STRSAFE_NULL_ON_FAILURE 0x00000800	If the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a

	STRSAFE_E_INSUFFICIENT_BUFFER failure, any truncated string is overwritten.
STRSAFE_NO_TRUNCATION 0x00001000	As in the case of **STRSAFE_NULL_ON_FAILURE**, if the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.

Return value

Type: **HRESULT**

It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Source data was present, fully copied without truncation, and the resultant destination buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	Either *pszDest* is **NULL** when there is source data present to copy, or an invalid flag was passed.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. Depending on the value of *dwFlags*, the destination buffer may contain a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

Compared to the functions it replaces, **StringCchCopyEx** provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCchCopyEx** always null-terminates and never overflows a valid destination buffer, even if the contents of the source string change during the operation.

Behavior is undefined if the strings pointed to by *pszSrc* and *pszDest* overlap.

Neither *pszSrc* nor *pszDest* should be **NULL** unless the **STRSAFE_IGNORE_NULLS** flag is specified, in which case both may be **NULL**. However, an error due to insufficient space may still be returned even though **NULL** values are ignored.

StringCchCopyEx can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCchCopyExA**
TCHAR	TEXT("string")	**StringCchCopyEx**
WCHAR	L"string"	**StringCchCopyExW**

> **Note**
>
> The strsafe.h header defines StringCchCopyEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbCopyEx

StringCchCopy

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCchCopyNA function (strsafe.h)

Article02/09/2023

Copies the specified number of characters from one string to another. The size of the destination buffer is provided to the function to ensure that **StringCchCopyN** does not write past the end of this buffer.

StringCchCopyN is a replacement for the following functions:

- strncpy, wcsncpy, _tcsncpy

Syntax

```
C++

STRSAFEAPI StringCchCopyNA(
  [out] STRSAFE_LPSTR  pszDest,
  [in]  size_t         cchDest,
  [in]  STRSAFE_PCNZCH pszSrc,
  [in]  size_t         cchToCopy
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which receives the copied characters.

`[in] cchDest`

Type: **size_t**

The size of *pszDest*, in characters. This value must be large enough to hold the copied characters (the length of *pszSrc* or the value of *cchSrc*, whichever is smaller) plus 1 to account for the terminating null character. The maximum number of characters allowed is **STRSAFE_MAX_CCH**.

`[in] pszSrc`

Type: **LPCTSTR**

The source string. This string must be readable up to *cchSrc* characters or a null terminator, whichever comes first.

`[in] cchToCopy`

Type: **size_t**

The maximum number of characters to be copied from *pszSrc* to *pszDest*.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Source data was present, the characters were copied from *pszSrc* without truncation, and the resultant destination buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cchDest* is either larger than **STRSAFE_MAX_CCH**, or the destination buffer is already full.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. The destination buffer contains a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

StringCchCopyN provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCchCopyN** always null-terminates and never overflows a valid destination buffer, even if the contents of the source string change during the operation.

While this routine is meant as a replacement for strncpy, there are differences in behavior. If *cchSrc* is larger than the number of characters in *pszSrc*, **StringCchCopyN**—unlike **strncpy**—does not continue to pad *pszDest* with null characters until *cchSrc* characters have been copied.

Behavior is undefined if the strings pointed to by *pszSrc* and *pszDest* overlap.

Neither *pszSrc* nor *pszDest* should be **NULL**. See StringCchCopyNEx if you require the handling of null string pointer values.

StringCchCopyN can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use, as shown in the following table.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCchCopyNA**
TCHAR	TEXT("string")	**StringCchCopyN**
WCHAR	L"string"	**StringCchCopyNW**

> **Note**
>
> The strsafe.h header defines StringCchCopyN as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]

Requirement	Value
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbCopyN

StringCchCopy

StringCchCopyNEx

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCchCopyNExA function (strsafe.h)

Article02/09/2023

Copies the specified number of characters from one string to another. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCchCopyNEx adds to the functionality of StringCchCopyN by returning a pointer to the end of the destination string as well as the number of characters left unused in that string. Flags may also be passed to the function for additional control.

StringCchCopyNEx is a replacement for the following functions:

- strncpy, wcsncpy, _tcsncpy

Syntax

C++

```
STRSAFEAPI StringCchCopyNExA(
  [out]           STRSAFE_LPSTR  pszDest,
  [in]            size_t         cchDest,
  [in]            STRSAFE_PCNZCH pszSrc,
  [in]            size_t         cchToCopy,
  [out, optional] STRSAFE_LPSTR  *ppszDestEnd,
  [out, optional] size_t         *pcchRemaining,
  [in]            DWORD          dwFlags
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which receives the copied string.

`[in] cchDest`

Type: **size_t**

The size of *pszDest*, in characters. This value must be at least large enough to hold the copied characters (the length of *pszSrc* or the value of *cchSrc*, whichever is smaller) plus

1 to account for the terminating null character. The maximum number of characters allowed is **STRSAFE_MAX_CCH**.

`[in] pszSrc`

Type: **LPCTSTR**

The source string. This string must be readable up to *cchSrc* characters or a null terminator, whichever comes first.

`[in] cchToCopy`

Type: **size_t**

The maximum number of characters to be copied from *pszSrc* to *pszDest*.

`[out, optional] ppszDestEnd`

Type: **LPTSTR***

The address of a pointer to the end of *pszDest*. If *ppszDestEnd* is non-**NULL** and any data is copied into the destination buffer, this points to the terminating null character at the end of the string.

`[out, optional] pcchRemaining`

Type: **size_t***

The number of unused characters in *pszDest*, including the terminating null character. If *pcchRemaining* is **NULL**, the count is not kept or returned.

`[in] dwFlags`

Type: **DWORD**

One or more of the following values.

Expand table

Value	Meaning
STRSAFE_FILL_BEHIND_NULL 0x00000200	If the function succeeds, the low byte of *dwFlags* (0) is used to fill the uninitialized portion of *pszDest* following the terminating null character.
STRSAFE_IGNORE_NULLS 0x00000100	Treat **NULL** string pointers like empty strings (TEXT("")).

STRSAFE_FILL_ON_FAILURE 0x00000400	If the function fails, the low byte of *dwFlags* (0) is used to fill the entire *pszDest* buffer, and the buffer is null-terminated. In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string returned is overwritten.
STRSAFE_NULL_ON_FAILURE 0x00000800	If the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.
STRSAFE_NO_TRUNCATION 0x00001000	As in the case of **STRSAFE_NULL_ON_FAILURE**, if the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Source data was present, fully copied without truncation, and the resultant destination buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	Either *pszDest* or *pszSrc* is greater than **STRSAFE_MAX_CCH**, *pszDest* is **NULL** when there is source data present to copy, or an invalid flag was passed.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. Depending on the value of *dwFlags*, the destination buffer may contain a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

StringCchCopyNEx provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCchCopyNEx** always null-terminates and never overflows a valid destination buffer, even if the contents of the source string change during the operation.

While this routine is meant as a replacement for strncpy, there are differences in behavior. If *cchSrc* is larger than the number of characters in *pszSrc*, **StringCchCopyNEx**—unlike **strncpy**—does not continue to pad *pszDest* with null characters until *cchSrc* characters have been copied.

Behavior is undefined if the strings pointed to by *pszSrc* and *pszDest* overlap.

Neither *pszSrc* nor *pszDest* should be **NULL** unless the **STRSAFE_IGNORE_NULLS** flag is specified, in which case both may be **NULL**. However, an error due to insufficient space may still be returned even though **NULL** values are ignored.

StringCchCopyNEx can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use, as shown in the following table.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCchCopyNExA**
TCHAR	TEXT("string")	**StringCchCopyNEx**
WCHAR	L"string"	**StringCchCopyNExW**

> **Note**
>
> The strsafe.h header defines StringCchCopyNEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbCopyNEx

StringCchCopyN

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCchGetsA function (strsafe.h)

Article02/09/2023

Gets one line of text from stdin, up to and including the newline character ('\n'). The line of text is copied to the destination buffer, and the newline character is replaced with a null character. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

> **Note** This function can only be used inline.

StringCchGets is a replacement for the following functions:

- gets, _getws, _getts

StringCchGets is not a replacement for **fgets**, which does not replace newline characters with a terminating null character.

Syntax

```
C++

STRSAFEAPI StringCchGetsA(
  [out] STRSAFE_LPSTR pszDest,
  [in]  size_t        cchDest
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which receives the copied characters.

`[in] cchDest`

Type: **size_t**

The size of the destination buffer, in characters. This value must be at least 2 for the function to succeed. The maximum number of characters allowed, including the

terminating null character, is **STRSAFE_MAX_CCH**. If *cchDest* is too small to hold the full line of text, the data is truncated.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Characters were read from stdin, were copied to the buffer at *pszDest*, and the buffer was null-terminated.
STRSAFE_E_END_OF_FILE	Indicates an error or end-of-file condition. Use feof or ferror to determine which one has occurred.
STRSAFE_E_INVALID_PARAMETER	The value in *cchDest* is larger than the maximum allowed value.
STRSAFE_E_INSUFFICIENT_BUFFER	The value in *cchDest* is 1 or less.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

StringCchGets provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCchGets** always null-terminates a nonzero-length destination buffer.

The value of *pszDest* should not be **NULL**. See StringCchGetsEx if you require the handling of null string pointer values.

StringCchGets can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use, as shown in the following table.

Expand table

String Data Type	String Literal	Function

char	"string"	**StringCchGetsA**
TCHAR	TEXT("string")	**StringCchGets**
WCHAR	L"string"	**StringCchGetsW**

> **Note**
>
> The strsafe.h header defines StringCchGets as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbGets

StringCchGetsEx

Feedback

Was this page helpful? Yes No

StringCchGetsExA function (strsafe.h)

Article02/09/2023

Gets one line of text from stdin, up to and including the newline character ('\n'). The line of text is copied to the destination buffer, and the newline character is replaced with a null character. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

> **Note** This function can only be used inline.

StringCchGetsEx adds to the functionality of StringCchGets by returning a pointer to the end of the destination string as well as the number of characters left unused in that string. Flags may also be passed to the function for additional control.

StringCchGetsEx is a replacement for the following functions:

- gets, _getws, _getts

StringCchGetsEx is not a replacement for **fgets**, which does not replace newline characters with a terminating null character.

Syntax

C++

```
STRSAFEAPI StringCchGetsExA(
  [out]           STRSAFE_LPSTR pszDest,
  [in]            size_t        cchDest,
  [out, optional] STRSAFE_LPSTR *ppszDestEnd,
  [out, optional] size_t        *pcchRemaining,
  [in]            DWORD         dwFlags
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which receives the copied characters.

`[in] cchDest`

Type: **size_t**

The size of the destination buffer, in characters. This value must be at least 2 for the function to succeed. The maximum number of characters allowed, including the terminating null character, is **STRSAFE_MAX_CCH**. If *cchDest* is too small to hold the full line of text, the data is truncated.

`[out, optional] ppszDestEnd`

Type: **LPTSTR***

The address of a pointer to the end of *pszDest*. If *ppszDestEnd* is non-**NULL** and any data is copied into the destination buffer, this points to the terminating null character at the end of the string.

`[out, optional] pcchRemaining`

Type: **size_t***

The number of unused characters in *pszDest*, including the terminating null character. If *pcchRemaining* is **NULL**, the count is not kept or returned.

`[in] dwFlags`

Type: **DWORD**

One or more of the following values.

Expand table

Value	Meaning
STRSAFE_FILL_BEHIND_NULL 0x00000200	If the function succeeds, the low byte of *dwFlags* (0) is used to fill the uninitialized portion of *pszDest* following the terminating null character.
STRSAFE_IGNORE_NULLS 0x00000100	Treat **NULL** string pointers like empty strings (TEXT("")). This flag is useful for emulating functions such as lstrcpy.
STRSAFE_FILL_ON_FAILURE 0x00000400	If the function fails, the low byte of *dwFlags* (0) is used to fill the entire *pszDest* buffer, and the buffer is null-terminated. In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string returned is overwritten.

STRSAFE_NULL_ON_FAILURE 0x00000800	If the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.
STRSAFE_NO_TRUNCATION 0x00001000	As in the case of STRSAFE_NULL_ON_FAILURE, if the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	Characters were read from stdin, were copied to the buffer at *pszDest*, and the buffer was null-terminated.
STRSAFE_E_END_OF_FILE	Indicates an error or end-of-file condition. Use feof or ferror to determine which one has occurred.
STRSAFE_E_INVALID_PARAMETER	The value in *cchDest* is larger than the maximum allowed value or an invalid flag was passed.
STRSAFE_E_INSUFFICIENT_BUFFER	The value in *cchDest* is 1 or less.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

StringCchGetsEx provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCchGetsEx** always null-terminates a nonzero-length destination buffer.

The value of *pszDest* should not be **NULL** unless the **STRSAFE_IGNORE_NULLS** flag is specified. However, an error due to insufficient space may still be returned even though **NULL** values are ignored.

StringCchGetsEx can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use, as shown in the following table.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCchGetsExA**
TCHAR	TEXT("string")	**StringCchGetsEx**
WCHAR	L"string"	**StringCchGetsExW**

> **Note**
>
> The strsafe.h header defines StringCchGetsEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbGetsEx

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCchLengthA function (strsafe.h)

Article02/09/2023

Determines whether a string exceeds the specified length, in characters.

StringCchLength is a replacement for the following functions:

- strlen, wcslen, _tcslen

Syntax

```
C++

STRSAFEAPI StringCchLengthA(
  [in]  STRSAFE_PCNZCH psz,
  [in]  size_t         cchMax,
  [out] size_t         *pcchLength
);
```

Parameters

`[in] psz`

Type: **LPCTSTR**

The string whose length is to be checked.

`[in] cchMax`

Type: **size_t**

The maximum number of characters allowed in *psz*, including the terminating null character. This value cannot exceed **STRSAFE_MAX_CCH**.

`[out] pcchLength`

Type: **size_t***

The number of characters in *psz*, not including the terminating null character. This value is valid only if *pcch* is not **NULL** and the function succeeds.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	The string at *psz* was not **NULL**, and the length of the string (including the terminating null character) is less than or equal to *cchMax* characters.
STRSAFE_E_INVALID_PARAMETER	The value in *psz* is **NULL**, *cchMax* is larger than **STRSAFE_MAX_CCH**, or *psz* is longer than *cchMax*.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

Compared to the functions it replaces, **StringCchLength** is an additional tool for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns.

StringCchLength can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCchLengthA**
TCHAR	TEXT("string")	**StringCchLength**
WCHAR	L"string"	**StringCchLengthW**

UnalignedStringCchLength is an alias for this function.

> **Note**
>
> The strsafe.h header defines StringCchLength as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the

UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

StringCbLength

Feedback

StringCchPrintf_lA function (strsafe.h)

Article02/09/2023

Writes formatted data to the specified string. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCchPrintf_l is similar to StringCchPrintf but includes a parameter for locale information.

Syntax

C++

```
STRSAFEAPI StringCchPrintf_lA(
  [out] STRSAFE_LPSTR                                    pszDest,
  [in]  size_t                                           cchDest,
  [in]  _Printf_format_string_params_(2)STRSAFE_LPCSTR pszFormat,
  [in]  _locale_t                                        locale,
        ...
);
```

Parameters

`[out] pszDest`

The destination buffer, which receives the formatted, null-terminated string created from *pszFormat* and its arguments.

`[in] cchDest`

The size of the destination buffer, in characters. This value must be sufficiently large to accommodate the final formatted string plus 1 to account for the terminating null character. The maximum number of characters allowed is **STRSAFE_MAX_CCH**.

`[in] pszFormat`

The format string. This string must be null-terminated. For more information, see Format Specification Syntax.

`[in] locale`

The locale object. For more information, see **_create_locale**.

...

The arguments to be inserted into the *pszFormat* string.

Return value

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	There was sufficient space for the result to be copied to *pszDest* without truncation, and the buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cchDest* is either 0 or larger than **STRSAFE_MAX_CCH**.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. The destination buffer contains a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Remarks

Behavior is undefined if the strings pointed to by *pszDest*, *pszFormat*, or any argument strings overlap.

Neither *pszFormat* nor *pszDest* should be **NULL**. See StringCchPrintf_lEx if you require the handling of null string pointer values.

In order to use this function, you must define the following macro in your header file, before including StrSafe.h.

```
#define STRSAFE_LOCALE_FUNCTIONS
```

> **Note**
>
> The strsafe.h header defines StringCchPrintf_l as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with

code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCchPrintf_lExA function (strsafe.h)

Article02/09/2023

Writes formatted data to the specified string. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCchPrintf_lEx is similar to StringCchPrintfEx but includes a parameter for locale information.

Syntax

```
C++

STRSAFEAPI StringCchPrintf_lExA(
  [out]           STRSAFE_LPSTR                                    pszDest,
  [in]            size_t                                           cchDest,
  [out]           STRSAFE_LPSTR
*ppszDestEnd,
  [out, optional] size_t
*pcchRemaining,
  [in]            DWORD                                            dwFlags,
  [in]            _Printf_format_string_params_(2)STRSAFE_LPCSTR pszFormat,
  [in]            _locale_t                                        locale,
                  ...
);
```

Parameters

`[out] pszDest`

The destination buffer, which receives the formatted, null-terminated string created from *pszFormat* and its arguments.

`[in] cchDest`

The size of the destination buffer, in characters. This value must be sufficiently large to accommodate the final formatted string plus 1 to account for the terminating null character. The maximum number of characters allowed is **STRSAFE_MAX_CCH**.

`[out] ppszDestEnd`

The address of a pointer to the end of *pszDest*. If *ppszDestEnd* is non-**NULL** and any data is copied into the destination buffer, this points to the terminating null character at the

end of the string.

`[out, optional] pcchRemaining`

The number of unused characters in *pszDest*, including the terminating null character. If *pcchRemaining* is **NULL**, the count is not kept or returned.

`[in] dwFlags`

One or more of the following values.

Expand table

Value	Meaning
STRSAFE_FILL_BEHIND_NULL 0x00000200	If the function succeeds, the low byte of *dwFlags* (0) is used to fill the uninitialized portion of *pszDest* following the terminating null character.
STRSAFE_IGNORE_NULLS 0x00000100	Treat **NULL** string pointers like empty strings (TEXT("")).
STRSAFE_FILL_ON_FAILURE 0x00000400	If the function fails, the low byte of *dwFlags* (0) is used to fill the entire *pszDest* buffer, and the buffer is null-terminated. In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string returned is overwritten.
STRSAFE_NULL_ON_FAILURE 0x00000800	If the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.
STRSAFE_NO_TRUNCATION 0x00001000	As in the case of **STRSAFE_NULL_ON_FAILURE**, if the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.

`[in] pszFormat`

The format string. This string must be null-terminated. For more information, see Format Specification Syntax.

`[in] locale`

The locale object. For more information, see **_create_locale**.

`...`

The Arguments to be inserted into the *pszFormat* string.

Return value

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	There was sufficient space for the result to be copied to *pszDest* without truncation and the buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cchDest* is either 0 or larger than **STRSAFE_MAX_CCH**, or the destination buffer is already full.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. Depending on the value of *dwFlags*, the destination buffer may contain a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Remarks

Behavior is undefined if the strings pointed to by *pszDest*, *pszFormat*, or any argument strings overlap.

Neither *pszFormat* nor *pszDest* should be **NULL** unless the **STRSAFE_IGNORE_NULLS** flag is specified, in which case both may be **NULL**. However, an error due to insufficient space may still be returned even though **NULL** values are ignored.

In order to use this function, you must define the following macro in your header file, before including StrSafe.h.

```
#define STRSAFE_LOCALE_FUNCTIONS
```

> **Note**
>
> The strsafe.h header defines StringCchPrintf_lEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the

UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCchPrintfA function (strsafe.h)

Article02/09/2023

Writes formatted data to the specified string. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCchPrintf is a replacement for the following functions:

- sprintf, swprintf, _stprintf
- wsprintf
- wnsprintf
- _snprintf, _snwprintf, _sntprintf

Syntax

C++

```
STRSAFEAPI StringCchPrintfA(
  [out] STRSAFE_LPSTR  pszDest,
  [in]  size_t         cchDest,
  [in]  STRSAFE_LPCSTR pszFormat,
        ...
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which receives the formatted, null-terminated string created from *pszFormat* and its arguments.

`[in] cchDest`

Type: **size_t**

The size of the destination buffer, in characters. This value must be sufficiently large to accommodate the final formatted string plus 1 to account for the terminating null character. The maximum number of characters allowed is **STRSAFE_MAX_CCH**.

`[in] pszFormat`

Type: **LPCTSTR**

The format string. This string must be null-terminated. For more information, see Format Specification Syntax.

`...`

The arguments to be inserted into the *pszFormat* string.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	There was sufficient space for the result to be copied to *pszDest* without truncation, and the buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cchDest* is either 0 or larger than **STRSAFE_MAX_CCH**.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. The destination buffer contains a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

Compared to the functions it replaces, **StringCchPrintf** provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCchPrintf** always null-terminates a nonzero-length destination buffer.

Behavior is undefined if the strings pointed to by *pszDest*, *pszFormat*, or any argument strings overlap.

Neither *pszFormat* nor *pszDest* should be **NULL**. See StringCchPrintfEx if you require the handling of null string pointer values.

StringCchPrintf can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCchPrintfA**
TCHAR	TEXT("string")	**StringCchPrintf**
WCHAR	L"string"	**StringCchPrintfW**

Examples

The following example shows a simple use of **StringCchPrintf**, using four arguments.

C++

```
TCHAR pszDest[30];
size_t cchDest = 30;

LPCTSTR pszFormat = TEXT("%s %d + %d = %d.");
TCHAR* pszTxt = TEXT("The answer is");

HRESULT hr = StringCchPrintf(pszDest, cchDest, pszFormat, pszTxt, 1, 2, 3);

// The resultant string at pszDest is "The answer is 1 + 2 = 3."
```

> **Note**
>
> The strsafe.h header defines StringCchPrintf as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbPrintf

StringCchPrintfEx

StringCchVPrintf

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCchPrintfExA function (strsafe.h)

Article02/09/2023

Writes formatted data to the specified string. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCchPrintfEx adds to the functionality of StringCchPrintf by returning a pointer to the end of the destination string as well as the number of characters left unused in that string. Flags may also be passed to the function for additional control.

StringCchPrintfEx is a replacement for the following functions:

- sprintf, swprintf, _stprintf
- wsprintf
- wnsprintf
- _snprintf, _snwprintf, _sntprintf

Syntax

```
C++

STRSAFEAPI StringCchPrintfExA(
  [out]           STRSAFE_LPSTR  pszDest,
  [in]            size_t         cchDest,
  [out, optional] STRSAFE_LPSTR  *ppszDestEnd,
  [out, optional] size_t         *pcchRemaining,
  [in]            DWORD          dwFlags,
  [in]            STRSAFE_LPCSTR pszFormat,
                  ...
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which receives the formatted, null-terminated string created from *pszFormat* and its arguments.

`[in] cchDest`

Type: **size_t**

The size of the destination buffer, in characters. This value must be sufficiently large to accommodate the final formatted string plus 1 to account for the terminating null character. The maximum number of characters allowed is **STRSAFE_MAX_CCH**.

`[out, optional] ppszDestEnd`

Type: **LPTSTR***

The address of a pointer to the end of *pszDest*. If *ppszDestEnd* is non-**NULL** and any data is copied into the destination buffer, this points to the terminating null character at the end of the string.

`[out, optional] pcchRemaining`

Type: **size_t***

The number of unused characters in *pszDest*, including the terminating null character. If *pcchRemaining* is **NULL**, the count is not kept or returned.

`[in] dwFlags`

Type: **DWORD**

One or more of the following values.

Expand table

Value	Meaning
STRSAFE_FILL_BEHIND_NULL 0x00000200	If the function succeeds, the low byte of *dwFlags* (0) is used to fill the uninitialized portion of *pszDest* following the terminating null character.
STRSAFE_IGNORE_NULLS 0x00000100	Treat **NULL** string pointers like empty strings (TEXT("")).
STRSAFE_FILL_ON_FAILURE 0x00000400	If the function fails, the low byte of *dwFlags* (0) is used to fill the entire *pszDest* buffer, and the buffer is null-terminated. In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string returned is overwritten.
STRSAFE_NULL_ON_FAILURE 0x00000800	If the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.
STRSAFE_NO_TRUNCATION 0x00001000	As in the case of **STRSAFE_NULL_ON_FAILURE**, if the function fails, *pszDest* is set to an empty string (TEXT("")).

	In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.

`[in] pszFormat`

Type: **LPCTSTR**

The format string. This string must be null-terminated. For more information, see Format Specification Syntax.

`...`

The Arguments to be inserted into the *pszFormat* string.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	There was sufficient space for the result to be copied to *pszDest* without truncation and the buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cchDest* is either 0 or larger than **STRSAFE_MAX_CCH**, or the destination buffer is already full.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. Depending on the value of *dwFlags*, the destination buffer may contain a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

Compared to the functions it replaces, **StringCchPrintfEx** provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCchPrintfEx** always null-terminates a nonzero-length destination buffer.

Behavior is undefined if the strings pointed to by *pszDest*, *pszFormat*, or any argument strings overlap.

Neither *pszFormat* nor *pszDest* should be **NULL** unless the **STRSAFE_IGNORE_NULLS** flag is specified, in which case both may be **NULL**. However, an error due to insufficient space may still be returned even though **NULL** values are ignored.

StringCchPrintfEx can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCchPrintfExA**
TCHAR	TEXT("string")	**StringCchPrintfEx**
WCHAR	L"string"	**StringCchPrintfExW**

> **Note**
>
> The strsafe.h header defines StringCchPrintfEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]

Requirement	Value
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbPrintfEx

StringCchPrintf

StringCchVPrintfEx

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCchVPrintf_lA function (strsafe.h)

Article02/09/2023

Writes formatted data to the specified string using a pointer to a list of arguments. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCchVPrintf_l is similar to StringCchVPrintf but includes a parameter for locale information.

Syntax

```
C++

STRSAFEAPI StringCchVPrintf_lA(
  [out] STRSAFE_LPSTR                                    pszDest,
  [in]  size_t                                           cchDest,
  [in]  _Printf_format_string_params_(2)STRSAFE_LPCSTR pszFormat,
  [in]  _locale_t                                        locale,
  [in]  va_list                                          argList
);
```

Parameters

`[out] pszDest`

The destination buffer, which receives the formatted, null-terminated string created from *pszFormat* and *argList*.

`[in] cchDest`

The size of the destination buffer, in characters. This value must be sufficiently large to accommodate the final formatted string plus 1 to account for the terminating null character. The maximum number of characters allowed is **STRSAFE_MAX_CCH**.

`[in] pszFormat`

The format string. This string must be null-terminated. For more information, see Format Specification Syntax.

`[in] locale`

The locale object. For more information, see **_create_locale**.

`[in] argList`

The arguments to be inserted into the *pszFormat* string.

Return value

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	There was sufficient space for the result to be copied to *pszDest* without truncation and the buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cchDest* is either 0 or larger than **STRSAFE_MAX_CCH**.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. The destination buffer contains a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Remarks

For more information on va_lists, see the conventions defined in Stdarg.h.

Behavior is undefined if the strings pointed to by *pszDest*, *pszFormat*, or any argument strings overlap.

Neither *pszFormat* nor *pszDest* should be **NULL**. See StringCchVPrintf_lEx if you require the handling of null string pointer values.

In order to use this function, you must define the following macro in your header file, before including StrSafe.h.

```
#define STRSAFE_LOCALE_FUNCTIONS
```

> **Note**

The strsafe.h header defines StringCchVPrintf_l as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCchVPrintf_lExA function (strsafe.h)

Article02/09/2023

Writes formatted data to the specified string using a pointer to a list of arguments. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCchVPrintf_lEx is similar to StringCchVPrintfEx but includes a parameter for locale information.

Syntax

```
C++

STRSAFEAPI StringCchVPrintf_lExA(
  [out]           STRSAFE_LPSTR                                    pszDest,
  [in]            size_t                                           cchDest,
  [out]           STRSAFE_LPSTR
*ppszDestEnd,
  [out, optional] size_t
*pcchRemaining,
  [in]            DWORD                                            dwFlags,
  [in]            _Printf_format_string_params_(2)STRSAFE_LPCSTR pszFormat,
  [in]            _locale_t                                        locale,
  [in]            va_list                                          argList
);
```

Parameters

`[out] pszDest`

The destination buffer, which receives the formatted, null-terminated string created from *pszFormat* and *argList*.

`[in] cchDest`

The size of the destination buffer, in characters. This value must be sufficiently large to accommodate the final formatted string plus 1 to account for the terminating null character. The maximum number of characters allowed is **STRSAFE_MAX_CCH**.

`[out] ppszDestEnd`

The address of a pointer to the end of *pszDest*. If *ppszDestEnd* is non-**NULL** and any data is copied into the destination buffer, this points to the terminating null character at the end of the string.

`[out, optional] pcchRemaining`

The number of unused characters in *pszDest*, including the terminating null character. If *pcchRemaining* is **NULL**, the count is not kept or returned.

`[in] dwFlags`

One or more of the following values.

Expand table

Value	Meaning
STRSAFE_FILL_BEHIND_NULL 0x00000200	If the function succeeds, the low byte of *dwFlags* (0) is used to fill the uninitialized portion of *pszDest* following the terminating null character.
STRSAFE_IGNORE_NULLS 0x00000100	Treat **NULL** string pointers like empty strings (TEXT("")).
STRSAFE_FILL_ON_FAILURE 0x00000400	If the function fails, the low byte of *dwFlags* (0) is used to fill the entire *pszDest* buffer, and the buffer is null-terminated. In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string returned is overwritten.
STRSAFE_NULL_ON_FAILURE 0x00000800	If the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.
STRSAFE_NO_TRUNCATION 0x00001000	As in the case of **STRSAFE_NULL_ON_FAILURE**, if the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.

`[in] pszFormat`

The format string. This string must be null-terminated. For more information, see Format Specification Syntax.

`[in] locale`

The locale object. For more information, see **_create_locale**.

`[in] argList`

The arguments to be inserted into the *pszFormat* string.

Return value

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	There was sufficient space for the result to be copied to *pszDest* without truncation and the buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cchDest* is either 0 or larger than **STRSAFE_MAX_CCH**, or the destination buffer is already full.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. Depending on the value of *dwFlags*, the destination buffer may contain a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Remarks

For more information on va_lists, see the conventions defined in Stdarg.h.

Behavior is undefined if the strings pointed to by *pszDest*, *pszFormat*, or any argument strings overlap.

Neither *pszFormat* nor *pszDest* should be **NULL** unless the **STRSAFE_IGNORE_NULLS** flag is specified, in which case both may be **NULL**. However, an error due to insufficient space may still be returned even though **NULL** values are ignored.

In order to use this function, you must define the following macro in your header file, before including StrSafe.h.

```
#define STRSAFE_LOCALE_FUNCTIONS
```

> **Note**
>
> The strsafe.h header defines StringCchVPrintf_lEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

Feedback

StringCchVPrintfA function (strsafe.h)

Article02/09/2023

Writes formatted data to the specified string using a pointer to a list of arguments. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCchVPrintf is a replacement for the following functions:

- vsprintf, vswprintf, _vstprintf
- vsnprintf, _vsnwprintf, _vsntprintf
- wvsprintf
- wvnsprintf

Syntax

```
C++
```

```
STRSAFEAPI StringCchVPrintfA(
  [out] STRSAFE_LPSTR  pszDest,
  [in]  size_t         cchDest,
  [in]  STRSAFE_LPCSTR pszFormat,
  [in]  va_list        argList
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which receives the formatted, null-terminated string created from *pszFormat* and *argList*.

`[in] cchDest`

Type: **size_t**

The size of the destination buffer, in characters. This value must be sufficiently large to accommodate the final formatted string plus 1 to account for the terminating null character. The maximum number of characters allowed is **STRSAFE_MAX_CCH**.

`[in] pszFormat`

Type: **LPCTSTR**

The format string. This string must be null-terminated. For more information, see Format Specification Syntax.

`[in] argList`

Type: **va_list**

The arguments to be inserted into the *pszFormat* string.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	There was sufficient space for the result to be copied to *pszDest* without truncation and the buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cchDest* is either 0 or larger than **STRSAFE_MAX_CCH**.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. The destination buffer contains a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

StringCchVPrintf provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCchVPrintf** always null-terminates a nonzero-length destination buffer.

For more information on va_lists, see the conventions defined in Stdarg.h.

Behavior is undefined if the strings pointed to by *pszDest*, *pszFormat*, or any argument strings overlap.

Neither *pszFormat* nor *pszDest* should be **NULL**. See StringCchVPrintfEx if you require the handling of null string pointer values.

StringCchVPrintf can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use, as shown in the following table.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCchVPrintfA**
TCHAR	TEXT("string")	**StringCchVPrintf**
WCHAR	L"string"	**StringCchVPrintfW**

> **Note**
>
> The strsafe.h header defines StringCchVPrintf as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbVPrintf

StringCchPrintf

StringCchVPrintfEx

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

StringCchVPrintfExA function (strsafe.h)

Article02/09/2023

Writes formatted data to the specified string using a pointer to a list of arguments. The size of the destination buffer is provided to the function to ensure that it does not write past the end of this buffer.

StringCchVPrintfEx adds to the functionality of StringCchVPrintf by returning a pointer to the end of the destination string as well as the number of characters left unused in that string. Flags may also be passed to the function for additional control.

StringCchVPrintfEx is a replacement for the following functions:

- vsprintf, vswprintf, _vstprintf
- vsnprintf, _vsnwprintf, _vsntprintf
- wvsprintf
- wvnsprintf

Syntax

C++

```
STRSAFEAPI StringCchVPrintfExA(
  [out]           STRSAFE_LPSTR  pszDest,
  [in]            size_t         cchDest,
  [out, optional] STRSAFE_LPSTR  *ppszDestEnd,
  [out, optional] size_t         *pcchRemaining,
  [in]            DWORD          dwFlags,
  [in]            STRSAFE_LPCSTR pszFormat,
  [in]            va_list        argList
);
```

Parameters

`[out] pszDest`

Type: **LPTSTR**

The destination buffer, which receives the formatted, null-terminated string created from *pszFormat* and *argList*.

`[in] cchDest`

Type: **size_t**

The size of the destination buffer, in characters. This value must be sufficiently large to accommodate the final formatted string plus 1 to account for the terminating null character. The maximum number of characters allowed is **STRSAFE_MAX_CCH**.

`[out, optional] ppszDestEnd`

Type: **LPTSTR***

The address of a pointer to the end of *pszDest*. If *ppszDestEnd* is non-**NULL** and any data is copied into the destination buffer, this points to the terminating null character at the end of the string.

`[out, optional] pcchRemaining`

Type: **size_t***

The number of unused characters in *pszDest*, including the terminating null character. If *pcchRemaining* is **NULL**, the count is not kept or returned.

`[in] dwFlags`

Type: **DWORD**

One or more of the following values.

Expand table

Value	Meaning
STRSAFE_FILL_BEHIND_NULL 0x00000200	If the function succeeds, the low byte of *dwFlags* (0) is used to fill the uninitialized portion of *pszDest* following the terminating null character.
STRSAFE_IGNORE_NULLS 0x00000100	Treat **NULL** string pointers like empty strings (TEXT("")).
STRSAFE_FILL_ON_FAILURE 0x00000400	If the function fails, the low byte of *dwFlags* (0) is used to fill the entire *pszDest* buffer, and the buffer is null-terminated. In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string returned is overwritten.
STRSAFE_NULL_ON_FAILURE 0x00000800	If the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.

STRSAFE_NO_TRUNCATION 0x00001000	As in the case of **STRSAFE_NULL_ON_FAILURE**, if the function fails, *pszDest* is set to an empty string (TEXT("")). In the case of a **STRSAFE_E_INSUFFICIENT_BUFFER** failure, any truncated string is overwritten.

`[in] pszFormat`

Type: **LPCTSTR**

The format string. This string must be null-terminated. For more information, see Format Specification Syntax.

`[in] argList`

Type: **va_list**

The arguments to be inserted into the *pszFormat* string.

Return value

Type: **HRESULT**

This function can return one of the following values. It is strongly recommended that you use the SUCCEEDED and FAILED macros to test the return value of this function.

Expand table

Return code	Description
S_OK	There was sufficient space for the result to be copied to *pszDest* without truncation and the buffer is null-terminated.
STRSAFE_E_INVALID_PARAMETER	The value in *cchDest* is either 0 or larger than **STRSAFE_MAX_CCH**, or the destination buffer is already full.
STRSAFE_E_INSUFFICIENT_BUFFER	The copy operation failed due to insufficient buffer space. Depending on the value of *dwFlags*, the destination buffer may contain a truncated, null-terminated version of the intended result. In situations where truncation is acceptable, this may not necessarily be seen as a failure condition.

Note that this function returns an **HRESULT** value, unlike the functions that it replaces.

Remarks

StringCchVPrintfEx provides additional processing for proper buffer handling in your code. Poor buffer handling is implicated in many security issues that involve buffer overruns. **StringCchVPrintfEx** always null-terminates a nonzero-length destination buffer.

For more information on va_lists, see the conventions defined in Stdarg.h.

Behavior is undefined if the strings pointed to by *pszDest*, *pszFormat*, or any argument strings overlap.

Neither *pszFormat* nor *pszDest* should be **NULL** unless the **STRSAFE_IGNORE_NULLS** flag is specified, in which case both may be **NULL**. However, an error due to insufficient space may still be returned even though **NULL** values are ignored.

StringCchVPrintfEx can be used in its generic form, or in its more specific forms. The data type of the string determines the form of this function that you should use, as shown in the following table.

Expand table

String Data Type	String Literal	Function
char	"string"	**StringCchVPrintfExA**
TCHAR	TEXT("string")	**StringCchVPrintfEx**
WCHAR	L"string"	**StringCchVPrintfExW**

> **Note**
>
> The strsafe.h header defines StringCchVPrintfEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP with SP2 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	strsafe.h

See also

Reference

StringCbVPrintfEx

StringCchPrintfEx

StringCchVPrintf

Feedback

Version Information

Article • 08/23/2019

Version information makes it easier for applications to install files properly and enables setup programs to analyze files currently installed. The version-information resource contains the version number of the file, its intended operating system, and the original file name.

In This Section

Name	Description
About Version Information	Discusses the version information functions.
Using Version Information	Discusses how to use the version information functions.
Version Information Reference	Contains the API reference.

Version Information Functions

Name	Description
GetFileVersionInfo	Retrieves version information for the specified file.
GetFileVersionInfoEx	Retrieves version information for the specified file.
GetFileVersionInfoSize	Determines whether the operating system can retrieve version information for a specified file. If version information is available, GetFileVersionInfoSize returns the size, in bytes, of that information.
GetFileVersionInfoSizeEx	Determines whether the operating system can retrieve version information for a specified file. If version information is available, GetFileVersionInfoSizeEx returns the size, in bytes, of that information.
VerFindFile	Determines where to install a file based on whether it locates another version of the file in the system. The values VerFindFile returns in the specified buffers are used in a subsequent call to the VerInstallFile function.
VerInstallFile	Installs the specified file based on information returned from the VerFindFile function. VerInstallFile decompresses the file, if necessary, assigns a unique filename, and checks for errors, such as outdated files.

Name	Description
VerLanguageName	Retrieves a description string for the language associated with a specified binary Microsoft language identifier.
VerQueryValue	Retrieves specified version information from the specified version-information resource. To retrieve the appropriate resource, before you call VerQueryValue, you must first call the GetFileVersionInfoSize function, and then the GetFileVersionInfo function.

Version Information Structures

Name	Description
String	Depicts the organization of data in a file-version resource. It contains a string that describes a specific aspect of a file, for example, a file's version, its copyright notices, or its trademarks.
StringFileInfo	Depicts the organization of data in a file-version resource. It contains version information that can be displayed for a particular language and code page.
StringTable	Depicts the organization of data in a file-version resource. It contains language and code page formatting information for the strings specified by the **Children** member. A code page is an ordered character set.
Var	Depicts the organization of data in a file-version resource. It typically contains a list of language and code page identifier pairs that the version of the application or DLL supports.
VarFileInfo	Depicts the organization of data in a file-version resource. It contains version information not dependent on a particular language and code page combination.
VS_FIXEDFILEINFO	Contains version information about a file. This information is language and code page independent.
VS_VERSIONINFO	Depicts the organization of data in a file-version resource. It is the root structure that contains all other file-version information structures.

About Version Information

Article • 08/19/2020

You can use the version information functions to determine where a file should be installed and identify conflicts with currently installed files. These functions enable you to avoid the following problems:

- installing older versions of components over newer versions
- changing the language in a mixed-language system without notification
- installing multiple copies of a library in different directories
- copying files to network directories shared by multiple users

The version information functions enable applications to query a version resource for file information and present the information in a clear format. This information includes the file's purpose, author, version number, and so on.

You can add version information to any files that can have Windows resources, such as DLLs, executable files, or .fon font files. To add the information, create a VERSIONINFO Resource and use the resource compiler to compile the resource.

Feedback

Using Version Information

Article • 08/19/2020

An installation program typically has the following goals:

- To place files in the correct location.
- To notify the user if the installation program is replacing an existing file with a version that is significantly different—for example, replacing a German-language file with an English-language file, or replacing a newer file with an older file.

When writing the installation program, you must have the following information for each file:

- The name and location of the file (referred to as the source file).
- The name of the equivalent file on the user's hard disk (referred to as the destination file). This name is usually the same as the filename on the installation disk.
- The sharing status of the file—that is, whether the file is private to the application being installed or could be shared by multiple applications.

The installation program can use the VerFindFile function to determine where the file should be copied on the disk. This function can also be used to specify whether the file is private to the application or can be shared. If a problem occurs in finding the file, **VerFindFile** returns an error value. For example, if the system is using the destination file, **VerFindFile** returns **VFF_FILEINUSE**. The installation program must notify the user of the problem and respond to the user's decision to continue or to end the installation.

The VerInstallFile function copies the source file to a temporary file in the directory specified by VerFindFile. If necessary, **VerInstallFile** expands the file by using the functions in the data decompression library.

VerInstallFile compares the version information of the temporary file to that of the destination file. If the two differ, **VerInstallFile** returns one or more error values. For example, it returns **VIF_SRCOLD** if the temporary file is older than the destination file and **VIF_DIFFLANG** if the files have different language identifiers or code-page values. The installation program must notify the user of the problem and respond to the user's decision to continue or to end the installation.

Some VerInstallFile errors are recoverable. That is, the installation program can call **VerInstallFile** again, specifying the **VIFF_FORCEINSTALL** option, to install the file regardless of the version conflict. If **VerInstallFile** returns **VIF_TEMPFILE** and the user

chooses not to force the installation, the installation program should delete the temporary file.

VerInstallFile could encounter a nonrecoverable error when attempting to force installation, even though the error did not exist previously. For example, the file could be locked by another user before the installation program attempted to force installation. If an installation program attempts to force installation after a non-recoverable error, **VerInstallFile** fails. The installation program must contain routines to recover from this type of error.

The recommended solution is to display a dialog box with the buttons **Install**, **Skip**, and **Install All**. (Another solution is a dialog box with the buttons **Yes**, **Yes to All**, **Skip**, and **Cancel**.) The **Install All** button should prevent the installation program from prompting the user about similar errors by including the **VIFF_FORCEINSTALL** option in all subsequent uses of VerInstallFile. For nonrecoverable errors, the **Install** and **Install All** buttons should be disabled.

To display a useful error message to the user, the installation program usually must retrieve information from the version resources of the conflicting files. There are four functions the installation program can use for this purpose:

- GetFileVersionInfoSize
- GetFileVersionInfo
- VerQueryValue
- VerLanguageName

GetFileVersionInfoSize returns the size of the version information. GetFileVersionInfo uses information retrieved by **GetFileVersionInfoSize** to retrieve a structure that contains the version information. VerQueryValue retrieves a specific member from that structure.

For example, if VerInstallFile returns the **VIF_DIFFTYPE** error, the installation program should use the GetFileVersionInfoSize, GetFileVersionInfo, and VerQueryValue functions on the temporary and destination files to obtain the general type of each file. If the languages of the files conflict, the installation program should also use VerLanguageName to translate the binary language identifier into a text representation of the language. (For example, 0x040C translates to the string "French.")

If VerInstallFile returns a file error, such as **VIF_ACCESSVIOLATION**, the installation program should use the GetLastError function to retrieve the most recent error value. The program should translate this value into an informative message to display to the user. The program must not yield control between the calls to **VerInstallFile** and **GetLastError**.

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Version Information Reference

Article • 04/27/2021

In This Section

- Version Information Functions
- Version Information Structures

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Version Information Functions

Article • 04/27/2021

In This Section

- GetFileVersionInfo
- GetFileVersionInfoEx
- GetFileVersionInfoSize
- GetFileVersionInfoSizeEx
- VerFindFile
- VerInstallFile
- VerLanguageName
- VerQueryValue

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

GetFileVersionInfoA function (winver.h)

Article02/09/2023

Retrieves version information for the specified file.

Syntax

```
C++

BOOL GetFileVersionInfoA(
  [in]  LPCSTR lptstrFilename,
        DWORD  dwHandle,
  [in]  DWORD  dwLen,
  [out] LPVOID lpData
);
```

Parameters

`[in] lptstrFilename`

Type: **LPCTSTR**

The name of the file. If a full path is not specified, the function uses the search sequence specified by the LoadLibrary function.

`dwHandle`

Type: **DWORD**

This parameter is ignored.

`[in] dwLen`

Type: **DWORD**

The size, in bytes, of the buffer pointed to by the *lpData* parameter.

Call the GetFileVersionInfoSize function first to determine the size, in bytes, of a file's version information. The *dwLen* member should be equal to or greater than that value.

If the buffer pointed to by *lpData* is not large enough, the function truncates the file's version information to the size of the buffer.

`[out] lpData`

Type: **LPVOID**

Pointer to a buffer that receives the file-version information.

You can use this value in a subsequent call to the VerQueryValue function to retrieve data from the buffer.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

File version info has fixed and non-fixed part. The fixed part contains information like version number. The non-fixed part contains things like strings. In the past **GetFileVersionInfo** was taking version information from the binary (exe/dll). Currently, it is querying fixed version from language neutral file (exe/dll) and the non-fixed part from mui file, merges them and returns to the user. If the given binary does not have a mui file then behavior is as in previous version.

Call the GetFileVersionInfoSize function before calling the **GetFileVersionInfo** function. To retrieve information from the file-version information buffer, use the VerQueryValue function.

> **Note**
>
> The winver.h header defines GetFileVersionInfo as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winver.h (include Windows.h)
Library	Version.lib
DLL	Api-ms-win-core-version-l1-1-0.dll

See also

Conceptual

GetFileVersionInfoSize

Reference

VS_VERSIONINFO

VerQueryValue

Version Information

Feedback

GetFileVersionInfoByHandle function

Article • 04/13/2024

Retrieves version information for the file associated with the specified handle.

Syntax

```
C++
```

```
BOOL GetFileVersionInfoByHandle(
    __in                          DWORD    dwFlags,
    __in                          HANDLE   hFile,
    __deref_out_bcount(*pdwLen) LPVOID * lplpData,
    __out                         PDWORD   pdwLen
    )
```

Parameters

dwFlags [in]

Controls the MUI DLLs (if any) from which the version resource is extracted. Zero or more of the following flags.

Expand table

Flag	Value	Meaning
FILE_VER_GET_LOCALISED	0x01	Loads the entire version resource (both strings and binary version information) from the corresponding MUI file, if available.
FILE_VER_GET_NEUTRAL	0x02	Loads the version resource strings from the corresponding MUI file, if available, and loads the binary version information (VS_FIXEDFILEINFO) from the corresponding language-neutral file, if available.
FILE_VER_GET_PREFETCHED	0x04	Indicates a preference for version.dll to attempt to preload the image outside of the loader lock to avoid contention. This flag does not change the behavior or semantics of the function.

hFile [in]

The handle to the file.

lplpData [out]

When this function returns, contains a pointer to a buffer that contains the file-version information.

You can use this value in a subsequent call to the VerQueryValue function to retrieve data from the buffer.

The caller is responsible for freeing this buffer by LocalFree when it is no longer being used.

pdwLen [out]

The size of the buffer returned in *lplpData*.

Return value

A boolean. If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

This function has no associated import library or header file; you must call it using the **LoadLibrary** and **GetProcAddress** functions. The API is exported from version.dll.

This function should be called with `*lplpdata = NULL`. If the function returns a non-NULL value the caller is responsible for freeing it with **LocalFree**. It is possible for this function to fail but still return an allocated buffer.

Requirements

Expand table

Requirement	Value
DLL	version.dll

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetFileVersionInfoExA function (winver.h)

Article02/22/2024

Retrieves version information for the specified file.

Syntax

```
C++
```

```
BOOL GetFileVersionInfoExA(
  [in]  DWORD  dwFlags,
  [in]  LPCSTR lpwstrFilename,
        DWORD  dwHandle,
  [in]  DWORD  dwLen,
  [out] LPVOID lpData
);
```

Parameters

`[in] dwFlags`

Type: **DWORD**

Controls the MUI DLLs (if any) from which the version resource is extracted. The value of this flag must match the flags passed to the corresponding GetFileVersionInfoSizeEx call, which was used to determine the buffer size that is passed in the *dwLen* parameter. Zero or more of the following flags.

Expand table

Value	Meaning
FILE_VER_GET_LOCALISED 0x01	Loads the entire version resource (both strings and binary version information) from the corresponding MUI file, if available.
FILE_VER_GET_NEUTRAL 0x02	Loads the version resource strings from the corresponding MUI file, if available, and loads the binary version information (**VS_FIXEDFILEINFO**) from the corresponding language-neutral file, if available.

FILE_VER_GET_PREFETCHED 0x04	Indicates a preference for version.dll to attempt to preload the image outside of the loader lock to avoid contention. This flag does not change the behavior or semantics of the function.

`[in] lpwstrFilename`

Type: **LPCTSTR**

The name of the file. If a full path is not specified, the function uses the search sequence specified by the LoadLibrary function.

`dwHandle`

Type: **DWORD**

This parameter is reserved, and expected to be zero (0).

`[in] dwLen`

Type: **DWORD**

The size, in bytes, of the buffer pointed to by the *lpData* parameter.

Call the GetFileVersionInfoSizeEx function first to determine the size, in bytes, of a file's version information. The *dwLen* parameter should be equal to or greater than that value.

If the buffer pointed to by *lpData* is not large enough, the function truncates the file's version information to the size of the buffer.

`[out] lpData`

Type: **LPVOID**

When this function returns, contains a pointer to a buffer that contains the file-version information.

You can use this value in a subsequent call to the VerQueryValue function to retrieve data from the buffer.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

Call the GetFileVersionInfoSizeEx function before calling the **GetFileVersionInfoEx** function. To retrieve information from the file-version information buffer, use the VerQueryValue function.

> **Note**
>
> The winver.h header defines GetFileVersionInfoEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	winver.h (include Windows.h)
Library	Version.lib
DLL	Api-ms-win-core-version-l1-1-0.dll

See also

Conceptual

GetFileVersionInfo

GetFileVersionInfoSizeEx

Reference

VS_VERSIONINFO

VerQueryValue

Version Information

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetFileVersionInfoSizeA function (winver.h)

Article02/09/2023

Determines whether the operating system can retrieve version information for a specified file. If version information is available, **GetFileVersionInfoSize** returns the size, in bytes, of that information.

Syntax

```
C++

DWORD GetFileVersionInfoSizeA(
  [in]            LPCSTR  lptstrFilename,
  [out, optional] LPDWORD lpdwHandle
);
```

Parameters

`[in] lptstrFilename`

Type: **LPCTSTR**

The name of the file of interest. The function uses the search sequence specified by the LoadLibrary function.

`[out, optional] lpdwHandle`

Type: **LPDWORD**

A pointer to a variable that the function sets to zero.

Return value

Type: **DWORD**

If the function succeeds, the return value is the size, in bytes, of the file's version information.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

Call the **GetFileVersionInfoSize** function before calling the GetFileVersionInfo function. The size returned by **GetFileVersionInfoSize** indicates the buffer size required for the version information returned by **GetFileVersionInfo**.

> **Note**
>
> The winver.h header defines GetFileVersionInfoSize as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winver.h (include Windows.h)
Library	Version.lib
DLL	Api-ms-win-core-version-l1-1-0.dll

See also

Conceptual

GetFileVersionInfo

Reference

VS_VERSIONINFO

VerQueryValue

Version Information

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

GetFileVersionInfoSizeExA function (winver.h)

Article02/09/2023

Determines whether the operating system can retrieve version information for a specified file. If version information is available, **GetFileVersionInfoSizeEx** returns the size, in bytes, of that information.

Syntax

```
C++

DWORD GetFileVersionInfoSizeExA(
  [in]            DWORD   dwFlags,
  [in]            LPCSTR  lpwstrFilename,
  [out, optional] LPDWORD lpdwHandle
);
```

Parameters

`[in] dwFlags`

Type: **DWORD**

Controls which MUI DLLs (if any) from which the version resource is extracted. Zero or more of the following flags.

Expand table

Value	Meaning
FILE_VER_GET_LOCALISED 0x01	Loads the entire version resource (both strings and binary version information) from the corresponding MUI file, if available.
FILE_VER_GET_NEUTRAL 0x002	Loads the version resource strings from the corresponding MUI file, if available, and loads the binary version information (**VS_FIXEDFILEINFO**) from the corresponding language-neutral file, if available.

`[in] lpwstrFilename`

Type: **LPCTSTR**

The name of the file of interest. The function uses the search sequence specified by the LoadLibrary function.

`[out, optional] lpdwHandle`

Type: **LPDWORD**

When this function returns, contains a pointer to a variable that is set to zero because this function sets it to zero. This parameter exists for historical reasons.

Return value

Type: **DWORD**

If the function succeeds, the return value is the size, in bytes, of the file's version information.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

Call the **GetFileVersionInfoSizeEx** function before calling the GetFileVersionInfoEx function. The size returned by **GetFileVersionInfoSizeEx** indicates the buffer size required for the version information returned by **GetFileVersionInfoEx**.

> **Note**
>
> The winver.h header defines GetFileVersionInfoSizeEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	winver.h (include Windows.h)
Library	Version.lib
DLL	Api-ms-win-core-version-l1-1-0.dll

See also

Conceptual

GetFileVersionInfoEx

GetFileVersionInfoSize

Reference

VS_VERSIONINFO

VerQueryValue

Version Information

Feedback

VerFindFileA function (winver.h)

Article02/09/2023

Determines where to install a file based on whether it locates another version of the file in the system. The values **VerFindFile** returns in the specified buffers are used in a subsequent call to the VerInstallFile function.

Syntax

C++

```
DWORD VerFindFileA(
  [in]           DWORD  uFlags,
  [in]           LPCSTR szFileName,
  [in, optional] LPCSTR szWinDir,
  [in]           LPCSTR szAppDir,
  [out]          LPSTR  szCurDir,
  [in, out]      PUINT  puCurDirLen,
  [out]          LPSTR  szDestDir,
  [in, out]      PUINT  puDestDirLen
);
```

Parameters

`[in] uFlags`

Type: **DWORD**

This parameter can be the following value. All other bits are reserved.

Expand table

Value	Meaning
VFFF_ISSHAREDFILE 0x0001	The source file can be shared by multiple applications. An application can use this information to determine where the file should be copied.

`[in] szFileName`

Type: **LPCTSTR**

The name of the file to be installed. Include only the file name and extension, not a path.

`[in, optional] szWinDir`

Type: **LPCTSTR**

The directory in which Windows is running or will be run. This string is returned by the GetWindowsDirectory function.

`[in] szAppDir`

Type: **LPCTSTR**

The directory where the installation program is installing a set of related files. If the installation program is installing an application, this is the directory where the application will reside. This parameter also points to the application's current directory unless otherwise specified.

`[out] szCurDir`

Type: **LPWSTR**

A buffer that receives the path to a current version of the file being installed. The path is a zero-terminated string. If a current version is not installed, the buffer will contain a zero-length string. The buffer should be at least **_MAX_PATH** characters long, although this is not required.

`[in, out] puCurDirLen`

Type: **PUINT**

The length of the *szCurDir* buffer. This pointer must not be **NULL**.

When the function returns, *lpuCurDirLen* contains the size, in characters, of the data returned in *szCurDir*, including the terminating null character. If the buffer is too small to contain all the data, *lpuCurDirLen* will be the size of the buffer required to hold the path.

`[out] szDestDir`

Type: **LPTSTR**

A buffer that receives the path to the installation location recommended by **VerFindFile**. The path is a zero-terminated string. The buffer should be at least **_MAX_PATH** characters long, although this is not required.

`[in, out] puDestDirLen`

Type: **PUINT**

A pointer to a variable that specifies the length of the *szDestDir* buffer. This pointer must not be **NULL**.

When the function returns, *lpuDestDirLen* contains the size, in characters, of the data returned in *szDestDir*, including the terminating null character. If the buffer is too small to contain all the data, *lpuDestDirLen* will be the size of the buffer needed to hold the path.

Return value

Type: **DWORD**

The return value is a bitmask that indicates the status of the file. It can be one or more of the following values. All other values are reserved.

Expand table

Return code/value	Description
VFF_CURNEDEST 0x0001	The currently installed version of the file is not in the recommended destination.
VFF_FILEINUSE 0x0002	The system is using the currently installed version of the file; therefore, the file cannot be overwritten or deleted.
VFF_BUFFTOOSMALL 0x0004	At least one of the buffers was too small to contain the corresponding string. An application should check the output buffers to determine which buffer was too small.

Remarks

This function works on 16-, 32-, and 64-bit file images.

VerFindFile searches for a copy of the specified file by using the OpenFile function. However, it determines the system directory from the specified Windows directory, or searches the path.

If the *dwFlags* parameter indicates that the file is private to this application (not **VFFF_ISSHAREDFILE**), **VerFindFile** recommends installing the file in the application's directory. Otherwise, if the system is running a shared copy of the system, the function recommends installing the file in the Windows directory. If the system is running a private copy of the system, the function recommends installing the file in the system directory.

> **Note**
>
> The winver.h header defines VerFindFile as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winver.h (include Windows.h)
Library	Version.lib
DLL	Api-ms-win-core-version-l1-1-0.dll

See also

Conceptual

GetWindowsDirectory

OpenFile

Other Resources

Reference

VerInstallFile

Version Information

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

VerInstallFileA function (winver.h)

Article02/09/2023

Installs the specified file based on information returned from the VerFindFile function. **VerInstallFile** decompresses the file, if necessary, assigns a unique filename, and checks for errors, such as outdated files.

Syntax

C++

```
DWORD VerInstallFileA(
  [in]      DWORD  uFlags,
  [in]      LPCSTR szSrcFileName,
  [in]      LPCSTR szDestFileName,
  [in]      LPCSTR szSrcDir,
  [in]      LPCSTR szDestDir,
  [in]      LPCSTR szCurDir,
  [out]     LPSTR  szTmpFile,
  [in, out] PUINT  puTmpFileLen
);
```

Parameters

`[in] uFlags`

Type: **DWORD**

This parameter can be one of the following values. All other bits are reserved.

Expand table

Value	Meaning
VIFF_FORCEINSTALL 0x0001	Installs the file regardless of mismatched version numbers. The function checks only for physical errors during installation.
VIFF_DONTDELETEOLD 0x0002	Installs the file without deleting the previously installed file, if the previously installed file is not in the destination directory.

`[in] szSrcFileName`

Type: **LPCTSTR**

The name of the file to be installed. This is the filename in the directory pointed to by the *szSrcDir* parameter; the filename can include only the filename and extension, not a path.

`[in] szDestFileName`

Type: **LPCTSTR**

The name **VerInstallFile** will give the new file upon installation. This file name may be different from the filename in the *szSrcFileName* directory. The new name should include only the file name and extension, not a path.

`[in] szSrcDir`

Type: **LPCTSTR**

The name of the directory where the file can be found.

`[in] szDestDir`

Type: **LPCTSTR**

The name of the directory where the file should be installed. VerFindFile returns this value in its *szDestDir* parameter.

`[in] szCurDir`

Type: **LPCTSTR**

The name of the directory where a preexisting version of this file can be found. VerFindFile returns this value in its *szCurDir* parameter.

`[out] szTmpFile`

Type: **LPTSTR**

The name of a temporary copy of the source file. The buffer should be at least **_MAX_PATH** characters long, although this is not required, and should be empty on input.

`[in, out] puTmpFileLen`

Type: **PUINT**

The length of the *szTmpFile* buffer. This pointer must not be **NULL**.

When the function returns, *lpuTmpFileLen* receives the size, in characters, of the data returned in *szTmpFile*, including the terminating null character. If the buffer is too small to contain all the data, *lpuTmpFileLen* will be the size of the buffer required to hold the data.

Return value

Type: **DWORD**

The return value is a bitmask that indicates exceptions. It can be one or more of the following values. All other values are reserved.

Expand table

Return code/value	Description
VIF_ACCESSVIOLATION 0x00000200L	A read, create, delete, or rename operation failed due to an access violation.
VIF_BUFFTOOSMALL 0x00040000L	The *szTmpFile* buffer was too small to contain the name of the temporary source file. When the function returns, *lpuTmpFileLen* contains the size of the buffer required to hold the filename.
VIF_CANNOTCREATE 0x00000800L	The function cannot create the temporary file. The specific error may be described by another flag.
VIF_CANNOTDELETE 0x00001000L	The function cannot delete the destination file, or cannot delete the existing version of the file located in another directory. If the **VIF_TEMPFILE** bit is set, the installation failed, and the destination file probably cannot be deleted.
VIF_CANNOTDELETECUR 0x00004000L	The existing version of the file could not be deleted and **VIFF_DONTDELETEOLD** was not specified.
VIF_CANNOTLOADCABINET 0x00100000L	The function cannot load the cabinet file.
VIF_CANNOTLOADLZ32 0x00080000L	The function cannot load the compressed file.
VIF_CANNOTREADDST 0x00020000L	The function cannot read the destination (existing) files. This prevents the function from examining the file's attributes.
VIF_CANNOTREADSRC 0x00010000L	The function cannot read the source file. This could mean that the path was not specified properly.

VIF_CANNOTRENAME 0x00002000L	The function cannot rename the temporary file, but already deleted the destination file.
VIF_DIFFCODEPG 0x00000010L	The new file requires a code page that cannot be displayed by the version of the system currently running. This error can be overridden by calling VerInstallFile with the **VIFF_FORCEINSTALL** flag set.
VIF_DIFFLANG 0x00000008L	The new and preexisting files have different language or code-page values. This error can be overridden by calling VerInstallFile again with the **VIFF_FORCEINSTALL** flag set.
VIF_DIFFTYPE 0x00000020L	The new file has a different type, subtype, or operating system from the preexisting file. This error can be overridden by calling VerInstallFile again with the **VIFF_FORCEINSTALL** flag set.
VIF_FILEINUSE 0x00000080L	The preexisting file is in use by the system and cannot be deleted.
VIF_MISMATCH 0x00000002L	The new and preexisting files differ in one or more attributes. This error can be overridden by calling VerInstallFile again with the **VIFF_FORCEINSTALL** flag set.
VIF_OUTOFMEMORY 0x00008000L	The function cannot complete the requested operation due to insufficient memory. Generally, this means the application ran out of memory attempting to expand a compressed file.
VIF_OUTOFSPACE 0x00000100L	The function cannot create the temporary file due to insufficient disk space on the destination drive.
VIF_SHARINGVIOLATION 0x00000400L	A read, create, delete, or rename operation failed due to a sharing violation.
VIF_SRCOLD 0x00000004L	The file to install is older than the preexisting file. This error can be overridden by calling VerInstallFile again with the **VIFF_FORCEINSTALL** flag set.
VIF_TEMPFILE 0x00000001L	The temporary copy of the new file is in the destination directory. The cause of failure is reflected in other flags.
VIF_WRITEPROT 0x00000040L	The preexisting file is write-protected. This error can be overridden by calling VerInstallFile again with the **VIFF_FORCEINSTALL** flag set.

Remarks

This function works on 16-, 32-, and 64-bit file images.

VerInstallFile copies the file from the source directory to the destination directory. If *szCurDir* indicates that a previous version of the file exists on the system, **VerInstallFile** compares the files' version stamp information. If the previously installed version of the file is more recent than the new version, or if the files' attributes are significantly different, for example, if they are in different languages, then **VerInstallFile** returns with one or more recoverable error codes.

VerInstallFile leaves the temporary file in the destination directory. The application can either override the error or delete the temporary file. If the application overrides the error, **VerInstallFile** deletes the previously installed version and renames the temporary file with the original filename.

> **Note**
>
> The winver.h header defines VerInstallFile as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winver.h (include Windows.h)
Library	Version.lib
DLL	Api-ms-win-core-version-l1-1-0.dll

See also

Conceptual

Reference

VerFindFile

Version Information

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

VerLanguageNameA function (winver.h)

Article02/22/2024

Retrieves a description string for the language associated with a specified binary Microsoft language identifier.

Syntax

```
C++
```

```
DWORD VerLanguageNameA(
  [in]  DWORD wLang,
  [out] LPSTR szLang,
  [in]  DWORD cchLang
);
```

Parameters

`[in] wLang`

Type: **DWORD**

The binary language identifier. For a complete list of the language identifiers, see Language Identifiers.

For example, the description string associated with the language identifier 0x040A is "Spanish (Traditional Sort)". If the identifier is unknown, the *szLang* parameter points to a default string ("Language Neutral").

`[out] szLang`

Type: **LPTSTR**

The language specified by the *wLang* parameter.

`[in] cchLang`

Type: **DWORD**

The size, in characters, of the buffer pointed to by *szLang*.

Return value

Type: **DWORD**

The return value is the size, in characters, of the string returned in the buffer. This value does not include the terminating null character. If the description string is smaller than or equal to the buffer, the entire description string is in the buffer. If the description string is larger than the buffer, the description string is truncated to the length of the buffer.

If an error occurs, the return value is zero. Unknown language identifiers do not produce errors.

Remarks

This function works on 16-, 32-, and 64-bit file images.

Typically, an installation program uses this function to translate a language identifier returned by the VerQueryValue function. The text string may be used in a dialog box that asks the user how to proceed in the event of a language conflict.

> **Note**
>
> The winver.h header defines VerLanguageName as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winver.h (include Windows.h)
Library	Version.lib

Requirement	Value
DLL	Api-ms-win-core-localization-l1-2-1.dll

See also

Version Information Overview

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

VerQueryValueA function (winver.h)

Article02/09/2023

Retrieves specified version information from the specified version-information resource. To retrieve the appropriate resource, before you call **VerQueryValue**, you must first call the GetFileVersionInfoSize function, and then the GetFileVersionInfo function.

Syntax

C++

```
BOOL VerQueryValueA(
  [in]  LPCVOID pBlock,
  [in]  LPCSTR  lpSubBlock,
  [out] LPVOID  *lplpBuffer,
  [out] PUINT   puLen
);
```

Parameters

`[in] pBlock`

Type: **LPCVOID**

The version-information resource returned by the GetFileVersionInfo function.

`[in] lpSubBlock`

Type: **LPCTSTR**

The version-information value to be retrieved. The string must consist of names separated by backslashes (\) and it must have one of the following forms.

\

The root block. The function retrieves a pointer to the VS_FIXEDFILEINFO structure for the version-information resource.

\VarFileInfo\Translation

The translation array in a Var variable information structure—the **Value** member of this structure. The function retrieves a pointer to this array of language and code page identifiers. An application can use these identifiers to access a language-specific StringTable structure (using the **szKey** member) in the version-information resource.

\StringFileInfo\lang-codepage\string-name

A value in a language-specific StringTable structure. The *lang-codepage* name is a concatenation of a language and code page identifier pair found as a **DWORD** in the translation array for the resource. Here the *lang-codepage* name must be specified as a hexadecimal string. The *string-name* name must be one of the predefined strings described in the following Remarks section. The function retrieves a string value specific to the language and code page indicated.

`[out] lplpBuffer`

Type: **LPVOID***

When this method returns, contains the address of a pointer to the requested version information in the buffer pointed to by *pBlock*. The memory pointed to by *lplpBuffer* is freed when the associated *pBlock* memory is freed.

`[out] puLen`

Type: **PUINT**

When this method returns, contains a pointer to the size of the requested data pointed to by *lplpBuffer*: for version information values, the length in characters of the string stored at *lplpBuffer*; for translation array values, the size in bytes of the array stored at *lplpBuffer*; and for root block, the size in bytes of the structure.

Return value

Type: **BOOL**

If the specified version-information structure exists, and version information is available, the return value is nonzero. If the address of the length buffer is zero, no value is available for the specified version-information name.

If the specified name does not exist or the specified resource is not valid, the return value is zero.

Remarks

This function works on 16-, 32-, and 64-bit file images.

The following are predefined version information Unicode strings.

Expand table

Comments	InternalName	ProductName
CompanyName	LegalCopyright	ProductVersion
FileDescription	LegalTrademarks	PrivateBuild
FileVersion	OriginalFilename	SpecialBuild

Examples

The following example shows how to enumerate the available version languages and retrieve the FileDescription string-value for each language.

Be sure to call the GetFileVersionInfoSize and GetFileVersionInfo functions before calling **VerQueryValue** to properly initialize the *pBlock* buffer.

C++

```
// Structure used to store enumerated languages and code pages.

HRESULT hr;

struct LANGANDCODEPAGE {
  WORD wLanguage;
  WORD wCodePage;
} *lpTranslate;

// Read the list of languages and code pages.

VerQueryValue(pBlock,
              TEXT("\\VarFileInfo\\Translation"),
              (LPVOID*)&lpTranslate,
              &cbTranslate);

// Read the file description for each language and code page.

for( i=0; i < (cbTranslate/sizeof(struct LANGANDCODEPAGE)); i++ )
{
  hr = StringCchPrintf(SubBlock, 50,
            TEXT("\\StringFileInfo\\%04x%04x\\FileDescription"),
```

```
            lpTranslate[i].wLanguage,
            lpTranslate[i].wCodePage);
    if (FAILED(hr))
    {
    // TODO: write error handler.
    }

  // Retrieve file description for language and code page "i".
  VerQueryValue(pBlock,
                SubBlock,
                &lpBuffer,
                &dwBytes);
}
```

> **Note**
>
> The winver.h header defines VerQueryValue as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winver.h (include Windows.h)
Library	Version.lib
DLL	Api-ms-win-core-version-l1-1-0.dll

See also

Conceptual

GetFileVersionInfo

GetFileVersionInfoSize

Reference

String

StringFileInfo

StringTable

VS_FIXEDFILEINFO

VS_VERSIONINFO

Var

VarFileInfo

Version Information

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Version Information Structures

Article • 04/27/2021

In This Section

- String
- StringFileInfo
- StringTable
- Var
- VarFileInfo
- VS_FIXEDFILEINFO
- VS_VERSIONINFO

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

String structure

Article • 12/11/2020

Represents the organization of data in a file-version resource. It contains a string that describes a specific aspect of a file, for example, a file's version, its copyright notices, or its trademarks.

Syntax

C++

```
typedef struct {
  WORD  wLength;
  WORD  wValueLength;
  WORD  wType;
  WCHAR szKey;
  WORD  Padding;
  WORD  Value;
} String;
```

Members

wLength

Type: **WORD**

The length, in bytes, of this **String** structure.

wValueLength

Type: **WORD**

The size, in words, of the **Value** member.

wType

Type: **WORD**

The type of data in the version resource. This member is 1 if the version resource contains text data and 0 if the version resource contains binary data.

szKey

Type: **WCHAR**

An arbitrary Unicode string. The **szKey** member can be one or more of the following values. These values are guidelines only.

Comments

The **Value** member contains any additional information that should be displayed for diagnostic purposes. This string can be an arbitrary length.

CompanyName

The **Value** member identifies the company that produced the file. For example, "Microsoft Corporation" or "Standard Microsystems Corporation, Inc."

FileDescription

The **Value** member describes the file in such a way that it can be presented to users. This string may be presented in a list box when the user is choosing files to install. For example, "Keyboard driver for AT-style keyboards" or "Microsoft Word for Windows".

FileVersion

The **Value** member identifies the version of this file. For example, **Value** could be "3.00A" or "5.00.RC2".

InternalName

The **Value** member identifies the file's internal name, if one exists. For example, this string could contain the module name for a DLL, a virtual device name for a Windows virtual device, or a device name for a MS-DOS device driver.

LegalCopyright

The **Value** member describes all copyright notices, trademarks, and registered trademarks that apply to the file. This should include the full text of all notices, legal symbols, copyright dates, trademark numbers, and so on. In English, this string should be in the format "Copyright Microsoft Corp. 1990 1994".

LegalTrademarks

The **Value** member describes all trademarks and registered trademarks that apply to the file. This should include the full text of all notices, legal symbols, trademark numbers, and so on. In English, this string should be in the format "Windows is a trademark of Microsoft Corporation".

OriginalFilename

The **Value** member identifies the original name of the file, not including a path. This enables an application to determine whether a file has been renamed by a user. This name may not be MS-DOS 8.3-format if the file is specific to a non-FAT file system.

PrivateBuild

The **Value** member describes by whom, where, and why this private version of the file was built. This string should only be present if the **VS_FF_PRIVATEBUILD** flag is set in the **dwFileFlags** member of the VS_FIXEDFILEINFO structure. For example, **Value** could be "Built by OSCAR on \OSCAR2".

ProductName

The **Value** member identifies the name of the product with which this file is distributed. For example, this string could be "Microsoft Windows".

ProductVersion

The **Value** member identifies the version of the product with which this file is distributed. For example, **Value** could be "3.00A" or "5.00.RC2".

SpecialBuild

The **Value** member describes how this version of the file differs from the normal version. This entry should only be present if the **VS_FF_SPECIALBUILD** flag is set in the **dwFileFlags** member of the VS_FIXEDFILEINFO structure. For example, **Value** could be "Private build for Olivetti solving mouse problems on M250 and M250E computers".

Padding

Type: **WORD**

As many zero words as necessary to align the **Value** member on a 32-bit boundary.

Value

Type: **WORD**

A zero-terminated string. See the **szKey** member description for more information.

Remarks

This structure is not a true C-language structure because it contains variable-length members. This structure was created solely to depict the organization of data in a

version resource and does not appear in any of the header files shipped with the Windows Software Development Kit (SDK).

A **String** structure may have an **szKey** value of, for example, "CompanyName" and a **Value** of "Microsoft Corporation". Another **String** structure with the same **szKey** value could contain a **Value** of "Microsoft GmbH". This might occur if the second **String** structure were associated with a **StringTable** structure whose **szKey** value is 040704b0 that is, German/Unicode.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

StringTable

VS_FIXEDFILEINFO

StringFileInfo

VS_VERSIONINFO

Conceptual

Version Information

Feedback

StringFileInfo structure

Article • 12/11/2020

Represents the organization of data in a file-version resource. It contains version information that can be displayed for a particular language and code page.

Syntax

```
C++
```

```
typedef struct {
  WORD        wLength;
  WORD        wValueLength;
  WORD        wType;
  WCHAR       szKey;
  WORD        Padding;
  StringTable Children;
} StringFileInfo;
```

Members

wLength

Type: **WORD**

The length, in bytes, of the entire **StringFileInfo** block, including all structures indicated by the **Children** member.

wValueLength

Type: **WORD**

This member is always equal to zero.

wType

Type: **WORD**

The type of data in the version resource. This member is 1 if the version resource contains text data and 0 if the version resource contains binary data.

szKey

Type: **WCHAR**

The Unicode string L"StringFileInfo".

Padding

Type: **WORD**

As many zero words as necessary to align the **Children** member on a 32-bit boundary.

Children

Type: StringTable

An array of one or more StringTable structures. Each **StringTable** structure's **szKey** member indicates the appropriate language and code page for displaying the text in that **StringTable** structure.

Remarks

This structure is not a true C-language structure because it contains variable-length members. This structure was created solely to depict the organization of data in a version resource and does not appear in any of the header files shipped with the Windows Software Development Kit (SDK).

The **Children** member of the VS_VERSIONINFO structure may contain zero or more **StringFileInfo** structures.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

StringTable

String

VS_VERSIONINFO

Conceptual

Version Information

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

StringTable structure

Article • 12/11/2020

Represents the organization of data in a file-version resource. It contains language and code page formatting information for the strings specified by the **Children** member. A code page is an ordered character set.

Syntax

```
C++
```

```
typedef struct {
  WORD   wLength;
  WORD   wValueLength;
  WORD   wType;
  WCHAR  szKey;
  WORD   Padding;
  String Children;
} StringTable;
```

Members

wLength

Type: **WORD**

The length, in bytes, of this **StringTable** structure, including all structures indicated by the **Children** member.

wValueLength

Type: **WORD**

This member is always equal to zero.

wType

Type: **WORD**

The type of data in the version resource. This member is 1 if the version resource contains text data and 0 if the version resource contains binary data.

szKey

Type: **WCHAR**

An 8-digit hexadecimal number stored as a Unicode string. The four most significant digits represent the language identifier. The four least significant digits represent the code page for which the data is formatted. Each Microsoft Standard Language identifier contains two parts: the low-order 10 bits specify the major language, and the high-order 6 bits specify the sublanguage. For a table of valid identifiers see .

Padding

Type: **WORD**

As many zero words as necessary to align the **Children** member on a 32-bit boundary.

Children

Type: **String**

An array of one or more **String** structures.

Remarks

This structure is not a true C-language structure because it contains variable-length members. This structure was created solely to depict the organization of data in a version resource and does not appear in any of the header files shipped with the Windows Software Development Kit (SDK).

The **Children** member of the **StringFileInfo** structure contains at least one **StringTable** structure.

Set the code page portion of the **szKey** member to the hexadecimal value 0x04b0 to indicate the Unicode code page, or to the hexadecimal value of the code page that is appropriate for the language component. After you choose the value for the code page you should continue to use the same value in later revisions to the file.

An executable file or DLL that supports multiple languages should have a version resource for each language, rather than a single version resource that contains strings in several languages. However, if you use the **Var** structure to list the languages that your application supports, the number of **StringTable** structures in the version resource is directly related to the number of language/code page identifier pairs in the **Value** member of the **Var** structure.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

String

StringFileInfo

Var

VarFileInfo

VS_VERSIONINFO

Conceptual

Version Information

Feedback

Var structure

Article • 12/11/2020

Represents the organization of data in a file-version resource. It typically contains a list of language and code page identifier pairs that the version of the application or DLL supports.

Syntax

C++

```
typedef struct {
  WORD  wLength;
  WORD  wValueLength;
  WORD  wType;
  WCHAR szKey;
  WORD  Padding;
  DWORD Value;
} Var;
```

Members

wLength

Type: **WORD**

The length, in bytes, of the **Var** structure.

wValueLength

Type: **WORD**

The length, in bytes, of the **Value** member.

wType

Type: **WORD**

The type of data in the version resource. This member is 1 if the version resource contains text data and 0 if the version resource contains binary data.

szKey

Type: **WCHAR**

The Unicode string L"Translation".

Padding

Type: **WORD**

As many zero words as necessary to align the **Value** member on a 32-bit boundary.

Value

Type: **DWORD**

An array of one or more values that are language and code page identifier pairs. For additional information, see the following Remarks section.

Remarks

This structure is not a true C-language structure because it contains variable-length members. This structure was created solely to depict the organization of data in a version resource and does not appear in any of the header files shipped with the Windows Software Development Kit (SDK).

If you use the **Var** structure to list the languages your application or DLL supports instead of using multiple version resources, use the **Value** member to contain an array of **DWORD** values indicating the language and code page combinations supported by this file. The low-order word of each **DWORD** must contain a Microsoft language identifier, and the high-order word must contain the IBM code page number. Either high-order or low-order word can be zero, indicating that the file is language or code page independent. If the **Var** structure is omitted, the file will be interpreted as both language and code page independent.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

VarFileInfo

StringFileInfo

StringTable

VS_VERSIONINFO

Conceptual

Version Information

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

VarFileInfo structure

Article • 12/11/2020

Represents the organization of data in a file-version resource. It contains version information not dependent on a particular language and code page combination.

Syntax

```
C++
```

```
typedef struct {
  WORD  wLength;
  WORD  wValueLength;
  WORD  wType;
  WCHAR szKey;
  WORD  Padding;
  Var   Children;
} VarFileInfo;
```

Members

wLength

Type: **WORD**

The length, in bytes, of the entire **VarFileInfo** block, including all structures indicated by the **Children** member.

wValueLength

Type: **WORD**

This member is always equal to zero.

wType

Type: **WORD**

The type of data in the version resource. This member is 1 if the version resource contains text data and 0 if the version resource contains binary data.

szKey

Type: **WCHAR**

The Unicode string L"VarFileInfo".

Padding

Type: **WORD**

As many zero words as necessary to align the **Children** member on a 32-bit boundary.

Children

Type: Var

Typically contains a list of languages that the application or DLL supports.

Remarks

This structure is not a true C-language structure because it contains variable-length members. This structure was created solely to depict the organization of data in a version resource and does not appear in any of the header files shipped with the Windows Software Development Kit (SDK).

The **Children** member of the VS_VERSIONINFO structure may contain zero or one **VarFileInfo** structures.

Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

Var

VS_VERSIONINFO

Conceptual

Version Information

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

VS_FIXEDFILEINFO structure (verrsrc.h)

Article04/02/2021

Contains version information for a file. This information is language and code page independent.

Syntax

C++

```
typedef struct tagVS_FIXEDFILEINFO {
  DWORD dwSignature;
  DWORD dwStrucVersion;
  DWORD dwFileVersionMS;
  DWORD dwFileVersionLS;
  DWORD dwProductVersionMS;
  DWORD dwProductVersionLS;
  DWORD dwFileFlagsMask;
  DWORD dwFileFlags;
  DWORD dwFileOS;
  DWORD dwFileType;
  DWORD dwFileSubtype;
  DWORD dwFileDateMS;
  DWORD dwFileDateLS;
} VS_FIXEDFILEINFO;
```

Members

`dwSignature`

Type: **DWORD**

Contains the value 0xFEEF04BD. This is used with the **szKey** member of the VS_VERSIONINFO structure when searching a file for the **VS_FIXEDFILEINFO** structure.

`dwStrucVersion`

Type: **DWORD**

The binary version number of this structure. The high-order word of this member contains the major version number, and the low-order word contains the minor version number.

`dwFileVersionMS`

Type: **DWORD**

The most significant 32 bits of the file's binary version number. This member is used with **dwFileVersionLS** to form a 64-bit value used for numeric comparisons.

`dwFileVersionLS`

Type: **DWORD**

The least significant 32 bits of the file's binary version number. This member is used with **dwFileVersionMS** to form a 64-bit value used for numeric comparisons.

`dwProductVersionMS`

Type: **DWORD**

The most significant 32 bits of the binary version number of the product with which this file was distributed. This member is used with **dwProductVersionLS** to form a 64-bit value used for numeric comparisons.

`dwProductVersionLS`

Type: **DWORD**

The least significant 32 bits of the binary version number of the product with which this file was distributed. This member is used with **dwProductVersionMS** to form a 64-bit value used for numeric comparisons.

`dwFileFlagsMask`

Type: **DWORD**

Contains a bitmask that specifies the valid bits in **dwFileFlags**. A bit is valid only if it was defined when the file was created.

`dwFileFlags`

Type: **DWORD**

Contains a bitmask that specifies the Boolean attributes of the file. This member can include one or more of the following values.

Expand table

Value	Meaning

VS_FF_DEBUG 0x00000001L	The file contains debugging information or is compiled with debugging features enabled.
VS_FF_INFOINFERRED 0x00000010L	The file's version structure was created dynamically; therefore, some of the members in this structure may be empty or incorrect. This flag should never be set in a file's VS_VERSIONINFO data.
VS_FF_PATCHED 0x00000004L	The file has been modified and is not identical to the original shipping file of the same version number.
VS_FF_PRERELEASE 0x00000002L	The file is a development version, not a commercially released product.
VS_FF_PRIVATEBUILD 0x00000008L	The file was not built using standard release procedures. If this flag is set, the StringFileInfo structure should contain a PrivateBuild entry.
VS_FF_SPECIALBUILD 0x00000020L	The file was built by the original company using standard release procedures but is a variation of the normal file of the same version number. If this flag is set, the StringFileInfo structure should contain a SpecialBuild entry.

`dwFileOS`

Type: **DWORD**

The operating system for which this file was designed. This member can be one of the following values.

Expand table

Value	Meaning
VOS_DOS 0x00010000L	The file was designed for MS-DOS.
VOS_NT 0x00040000L	The file was designed for Windows NT.
VOS__WINDOWS16 0x00000001L	The file was designed for 16-bit Windows.
VOS__WINDOWS32 0x00000004L	The file was designed for 32-bit Windows.
VOS_OS216 0x00020000L	The file was designed for 16-bit OS/2.

VOS_OS232 0x00030000L	The file was designed for 32-bit OS/2.
VOS__PM16 0x00000002L	The file was designed for 16-bit Presentation Manager.
VOS__PM32 0x00000003L	The file was designed for 32-bit Presentation Manager.
VOS_UNKNOWN 0x00000000L	The operating system for which the file was designed is unknown to the system.

An application can combine these values to indicate that the file was designed for one operating system running on another. The following **dwFileOS** values are examples of this, but are not a complete list.

Expand table

Value	Meaning
VOS_DOS_WINDOWS16 0x00010001L	The file was designed for 16-bit Windows running on MS-DOS.
VOS_DOS_WINDOWS32 0x00010004L	The file was designed for 32-bit Windows running on MS-DOS.
VOS_NT_WINDOWS32 0x00040004L	The file was designed for Windows NT.
VOS_OS216_PM16 0x00020002L	The file was designed for 16-bit Presentation Manager running on 16-bit OS/2.
VOS_OS232_PM32 0x00030003L	The file was designed for 32-bit Presentation Manager running on 32-bit OS/2.

`dwFileType`

Type: **DWORD**

The general type of file. This member can be one of the following values. All other values are reserved.

Expand table

Value	Meaning
VFT_APP	The file contains an application.

0x00000001L	
VFT_DLL 0x00000002L	The file contains a DLL.
VFT_DRV 0x00000003L	The file contains a device driver. If **dwFileType** is **VFT_DRV**, **dwFileSubtype** contains a more specific description of the driver.
VFT_FONT 0x00000004L	The file contains a font. If **dwFileType** is **VFT_FONT**, **dwFileSubtype** contains a more specific description of the font file.
VFT_STATIC_LIB 0x00000007L	The file contains a static-link library.
VFT_UNKNOWN 0x00000000L	The file type is unknown to the system.
VFT_VXD 0x00000005L	The file contains a virtual device.

`dwFileSubtype`

Type: **DWORD**

The function of the file. The possible values depend on the value of **dwFileType**. For all values of **dwFileType** not described in the following list, **dwFileSubtype** is zero.

If **dwFileType** is **VFT_DRV**, **dwFileSubtype** can be one of the following values.

Expand table

Value	Meaning
VFT2_DRV_COMM 0x0000000AL	The file contains a communications driver.
VFT2_DRV_DISPLAY 0x00000004L	The file contains a display driver.
VFT2_DRV_INSTALLABLE 0x00000008L	The file contains an installable driver.
VFT2_DRV_KEYBOARD 0x00000002L	The file contains a keyboard driver.
VFT2_DRV_LANGUAGE 0x00000003L	The file contains a language driver.
VFT2_DRV_MOUSE	The file contains a mouse driver.

0x00000005L	
VFT2_DRV_NETWORK 0x00000006L	The file contains a network driver.
VFT2_DRV_PRINTER 0x00000001L	The file contains a printer driver.
VFT2_DRV_SOUND 0x00000009L	The file contains a sound driver.
VFT2_DRV_SYSTEM 0x00000007L	The file contains a system driver.
VFT2_DRV_VERSIONED_PRINTER 0x0000000CL	The file contains a versioned printer driver.
VFT2_UNKNOWN 0x00000000L	The driver type is unknown by the system.

If **dwFileType** is **VFT_FONT**, **dwFileSubtype** can be one of the following values.

Expand table

Value	Meaning
VFT2_FONT_RASTER 0x00000001L	The file contains a raster font.
VFT2_FONT_TRUETYPE 0x00000003L	The file contains a TrueType font.
VFT2_FONT_VECTOR 0x00000002L	The file contains a vector font.
VFT2_UNKNOWN 0x00000000L	The font type is unknown by the system.

If **dwFileType** is **VFT_VXD**, **dwFileSubtype** contains the virtual device identifier included in the virtual device control block.

All **dwFileSubtype** values not listed here are reserved.

`dwFileDateMS`

Type: **DWORD**

The most significant 32 bits of the file's 64-bit binary creation date and time stamp.

`dwFileDateLS`

Type: **DWORD**

The least significant 32 bits of the file's 64-bit binary creation date and time stamp.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	verrsrc.h (include Windows.h)

See also

Conceptual

Reference

String

StringFileInfo

VS_VERSIONINFO

Version Information

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

VS_VERSIONINFO structure

Article • 12/11/2020

Represents the organization of data in a file-version resource. It is the root structure that contains all other file-version information structures.

Syntax

```
C++
```

```
typedef struct {
  WORD             wLength;
  WORD             wValueLength;
  WORD             wType;
  WCHAR            szKey;
  WORD             Padding1;
  VS_FIXEDFILEINFO Value;
  WORD             Padding2;
  WORD             Children;
} VS_VERSIONINFO;
```

Members

wLength

Type: **WORD**

The length, in bytes, of the **VS_VERSIONINFO** structure. This length does not include any padding that aligns any subsequent version resource data on a 32-bit boundary.

wValueLength

Type: **WORD**

The length, in bytes, of the **Value** member. This value is zero if there is no **Value** member associated with the current version structure.

wType

Type: **WORD**

The type of data in the version resource. This member is 1 if the version resource contains text data and 0 if the version resource contains binary data.

szKey

Type: **WCHAR**

The Unicode string L"VS_VERSION_INFO".

Padding1

Type: **WORD**

Contains as many zero words as necessary to align the **Value** member on a 32-bit boundary.

Value

Type: VS_FIXEDFILEINFO

Arbitrary data associated with this **VS_VERSIONINFO** structure. The **wValueLength** member specifies the length of this member; if **wValueLength** is zero, this member does not exist.

Padding2

Type: **WORD**

As many zero words as necessary to align the **Children** member on a 32-bit boundary. These bytes are not included in **wValueLength**. This member is optional.

Children

Type: **WORD**

An array of zero or one StringFileInfo structures, and zero or one VarFileInfo structures that are children of the current **VS_VERSIONINFO** structure.

Remarks

This structure is not a true C-language structure because it contains variable-length members. This structure was created solely to depict the organization of data in a version resource and does not appear in any of the header files shipped with the Windows Software Development Kit (SDK).

Requirements

Requirement	Value

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

See also

Reference

StringFileInfo

VerQueryValue

VarFileInfo

VS_FIXEDFILEINFO

Conceptual

Version Information

Feedback

Resource Compiler

Article • 08/23/2019

The Microsoft Windows Resource Compiler (RC) is a tool used in building Windows-based applications. This overview describes how to create a resource-definition (script) file, how to compile your application's resources, and how to add compiled resources to your application.

This tool is available in Visual Studio and the Microsoft Windows Software Development Kit (SDK).

The following topics describe the resource file syntax and the RC command line:

- About Resource Files
- Using RC (The RC Command Line)

Feedback

About Resource Files

Article • 08/23/2019

To include resources in your Windows-based application with RC, do the following:

1. Create individual files for your cursors, icons, bitmaps, dialog boxes, and fonts.
2. Create a resource-definition script (.rc file) that describes the resources used by your application.
3. Compile the script with RC. For more information, see Using RC (The RC Command Line).
4. Link the compiled resource (.res) file into the application's executable file with your linker.

A resource file is a text file with the extension .rc. The file can use single-byte, double-byte, or Unicode characters. The syntax and semantics for the RC preprocessor are similar to those of the Microsoft C/C++ compiler. However, RC supports a subset of the preprocessor directives, defines, and pragmas in a script.

The script file defines resources. For a resource that exists in a separate file, such as an icon or cursor, the script specifies the resource and the file that contains it. For some resources, such as a menu, the entire definition of the resource exists within the script.

The following topics describe the information a script file can contain:

- Comments, which are notes to be ignored by RC.
- Predefined macros, which take no arguments and cannot be redefined.
- Preprocessor directives, which instruct RC to perform actions on the script before compiling it.
- Preprocessor operators, which are used with the **#define** directive.
- Pragma directives
- Resource-definition statements, which name and describe resources.

Comments (Menus and Other Resources)

Article • 12/10/2020

RC supports C-style syntax for both single-line comments and block comments. Single-line comments begin with two forward slashes (//) and run to the end of the line. The following is an example of a resource statement followed by a single-line comment.

syntax

```
NewCursor  CURSOR  NEW.CUR  // a new cursor for the application.
```

Block comments begin with an opening delimiter (/*) and run to a closing delimiter (*/). Comments do not nest. The following is an example of a block comment at the beginning of a .rc file.

syntax

```
/*
    Resources.Rc

    Contains the resource definitions for the application.
    Control identifiers are defined in Resources.h.
*/

#include "resources.h"
//...
```

Feedback

Predefined Macros

Article • 08/23/2019

RC does not support the ANSI C predefined macros (**__DATE__**, **__FILE__**, **__LINE__**, **__STDC__**, **__TIME__**, **__TIMESTAMP__**). Therefore, you cannot include these macros in header files that you will include in your resource script.

RC does define RC_INVOKED, which enables you conditionally compile portions of your header files, depending on whether the compiler is your C compiler or the RC compiler. This is important because the RC compiler supports only a subset of the statements a C compiler would support.

To conditionally compile your code with the RC compiler, surround code that RC cannot compile with #ifndef RC_INVOKED and **#endif**.

The following example is taken from the SDK samples. It demonstrates how to create a header file that can be compiled conditionally.

syntax

```
#ifndef RC_INVOKED
#pragma message("Including CntrOutl.H from " __FILE__)
#endif
```

Feedback

Preprocessor Directives (Menus and Other Resources)

Article • 12/10/2020

You can use the directives described in the following table as needed in your resource script. They instruct RC to perform actions or to assign values to names.

Directive	Description
#define	Defines a specified name by assigning it a given value.
#elif	Marks an optional clause of a conditional-compilation block.
#else	Marks the last optional clause of a conditional-compilation block.
#endif	Marks the end of a conditional-compilation block.
#if	Conditionally compiles the script if a specified expression is true.
#ifdef	Conditionally compiles the script if a specified name is defined.
#ifndef	Conditionally compiles the script if a specified name is not defined.
#include	Copies the contents of a file into the resource-definition file.
#undef	Removes the definition of the specified name.

To define symbols for your resource identifiers, use the **#define** directive to define them in a header file. Include this header both in the resource script and your application source code. Similarly, you define the values for resource attributes and styles by including Windows.h in the resource script.

RC treats files with the .c and .h extensions in a special manner. It assumes that a file with one of these extensions does not contain resources. If a file has the .c or .h file name extension, RC ignores all lines in the file except the preprocessor directives. Therefore, to include a file that contains resources in another resource script, give the file to be included an extension other than .c or .h.

Related topics

Pragma Directives

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

#define

Article • 08/23/2019

The **#define** directive assigns the given value to the specified name. All subsequent occurrences of the name are replaced by the value.

syntax

```
#define name value
```

name

Name to be defined. This value is any combination of letters, digits, and punctuation that is valid for the C/C++ preprocessor.

value

Integer, character string, or line of text.

Example

This example assigns values to the names NONZERO and USERCLASS:

syntax

```
#define     NONZERO     1
#define     USERCLASS   "MyControlClass"
```

Related topics

Preprocessor Directives

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

#elif

Article • 08/23/2019

The **#elif** directive marks an optional clause of a conditional-compilation block defined by a **#ifdef**, **#ifndef**, or **#if** directive. The directive controls conditional compilation of the resource file by checking the specified constant expression. If the constant expression is nonzero, **#elif** directs the compiler to continue processing statements up to the next **#endif**, **#else**, or **#elif** directive and then skip to the statement after **#endif**. If the constant expression is zero, **#elif** directs the compiler to skip to the next **#endif**, **#else**, or **#elif** directive. You can use any number of **#elif** directives in a conditional block.

syntax

```
#elif constant-expression
```

constant-expression

Expression to be checked. This value is a defined name, an integer constant, or an expression consisting of names, integers, and arithmetic and relational operators.

Example

In this example, **#elif** directs the compiler to process the second **BITMAP** statement only if the value assigned to the name Version is less than 7. The **#elif** directive itself is processed only if Version is greater than or equal to 3.

syntax

```
#if Version < 3
BITMAP 1 errbox.bmp
#elif Version < 7
BITMAP 1 userbox.bmp
#endif
```

Related topics

Preprocessor Directives

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

#else

Article • 08/23/2019

The **#else** directive marks an optional clause of a conditional-compilation block defined by a **#ifdef**, **#ifndef**, or **#if** directive. The **#else** directive must be the last directive before the **#endif** directive.

syntax

```
#else
```

This directive has no parameters.

Example

This example compiles the second **BITMAP** statement only if DEBUG is not defined:

syntax

```
#ifdef DEBUG
    BITMAP 1 errbox.bmp
#else
    BITMAP 1 userbox.bmp
#endif
```

Related topics

Preprocessor Directives

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

#endif

Article • 08/23/2019

The **#endif** directive marks the end of a conditional-compilation block defined by a **#ifdef** directive. One **#endif** is required for each **#if**, **#ifdef**, or **#ifndef** directive.

syntax

```
#endif
```

This directive has no parameters.

Related topics

Preprocessor Directives

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

#if

Article • 08/23/2019

The **#if** directive controls conditional compilation of the resource file by checking the specified constant expression. If the constant expression is nonzero, **#if** directs the compiler to continue processing statements up to the next **#endif**, **#else**, or **#elif** directive and then skip to the statement after the **#endif** directive. If the constant expression is zero, **#if** directs the compiler to skip to the next **#endif**, **#else**, or **#elif** directive.

syntax

```
#if constant-expression
```

constant-expression

Expression to be checked. This value is a defined name, an integer constant, or an expression consisting of names, integers, and arithmetic and relational operators.

Example

This example compiles the **BITMAP** statement only if the value assigned Version is less than 3:

syntax

```
#if Version < 3
BITMAP 1 errbox.bmp
#endif
```

Related topics

Preprocessor Directives

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

#ifdef

Article • 08/23/2019

The **#ifdef** directive controls conditional compilation of the resource file by checking the specified name. If the name has been defined by using a **#define** directive or by using the **/d** command-line option with the resource compiler, **#ifdef** directs the compiler to continue with the statement immediately after the **#ifdef** directive. If the name has not been defined, **#ifdef** directs the compiler to skip all statements up to the next **#endif** directive.

syntax

```
#ifdef name
```

name

Name to be checked by the directive.

Example

This example compiles the **BITMAP** statement only if Debug is defined:

syntax

```
#ifdef Debug
BITMAP 1 errbox.bmp
#endif
```

Related topics

Preprocessor Directives

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

#ifndef

Article • 08/23/2019

The **#ifndef** directive controls conditional compilation of the resource file by checking the specified name. If the name has not been defined or if its definition has been removed by using the **#undef** directive, **#ifndef** directs the compiler to continue processing statements up to the next **#endif**, **#else**, or **#elif** directive and then skip to the statement after the **#endif** directive. If the name is defined, **#ifndef** directs the compiler to skip to the next **#endif**, **#else**, or **#elif** directive.

syntax

```
#ifndef name
```

name

Name to be checked by the directive.

Example

This example compiles the **BITMAP** statement only if Optimize is not defined:

syntax

```
#ifndef Optimize
BITMAP 1 errbox.bmp
#endif
```

Related topics

Preprocessor Directives

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

' include'

Article • 12/10/2020

The **#include** directive causes the resource compiler to process the file specified in the *filename* parameter. This file should be a header file that defines the constants used in the resource-definition file. The file can use single-byte, double-byte, or Unicode characters.

syntax

```
#include filename
```

filename

Name of the file to be included. If the file is in the current directory, the string must be enclosed in double quotation marks; if the file is in the directory specified by the INCLUDE environment variable, the string must be enclosed in less-than and greater-than characters (< >). You must give a full path enclosed in double quotation marks (") if the file is not in the current directory or in the directory specified by the INCLUDE environment variable.

Remarks

Use the following statement in your header file to surround statements that can be compiled by a C compiler but not RC:

syntax

```
#ifndef RC_INVOKED
```

This way, you can use the same include files in your .c and .rc files.

Example

This example processes the header files Windows.h and MyDefs.h while compiling the resource-definition file:

syntax

```
#include <windows.h>
#include "headers\mydefs.h"
```

Related topics

Preprocessor Directives

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

#undef

Article • 08/23/2019

The **#undef** directive removes the current definition of the specified name. All subsequent occurrences of the name are processed without replacement.

syntax

```
#undef name
```

name

Name to be removed. This value is any combination of letters, digits, and punctuation that is valid for the C/C++ preprocessor.

Example

This example removes the definitions for the names nonzero and USERCLASS:

syntax

```
#undef     nonzero
#undef     USERCLASS
```

Related topics

Preprocessor Directives

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Preprocessor Operators

Article • 08/23/2019

RC supports using the standard C preprocessing operators in macro definitions. These operators are described in the following table.

Operator	Description
#	Encloses the argument in quotes.
#@	Encloses the argument in single quotes.
##	Concatenates tokens used as arguments to form other tokens.

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Pragma Directives

Article • 08/19/2020

RC does not support the pragma directives supported by the C/C++ compiler. However, RC does support the following pragma directive for changing the code page:

syntax

```
#pragma code_page( [ DEFAULT | CodePageNum ] )
```

This pragma is not supported in an included resource file (.rc). Therefore, if you have a parent .rc file and it includes .rc files for multiple languages, use this pragma before including another .rc file rather than using it in the included .rc file itself. This is demonstrated in the following example.

syntax

```
#include "English.rc"
#pragma code_page(932)
#include "Japanese.rc"
```

For a list of values for CodePageNum, see Code Page Identifiers.

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Resource-Definition Statements

Article • 06/13/2022

The resource-definition statements define the resources that the resource compiler puts in the resource (.Res) file. After the .Res file is linked to the executable file, the application can load its resources at run time as needed. All resource statements associate an identifying name or number with a given resource.

The resource-definition statements can be divided into the following categories:

- Resources
- Controls
- Statements

The following tables describe the resource-definition statements.

Resources

Resource	Description
ACCELERATORS	Defines menu accelerator keys.
BITMAP	Defines a bitmap by naming it and specifying the name of the file that contains it. (To use a particular bitmap, the application requests it by name.)
CURSOR	Defines a cursor or animated cursor by naming it and specifying the name of the file that contains it. (To use a particular cursor, the application requests it by name.)
DIALOG	Defines a template that an application can use to create dialog boxes.
DIALOGEX	Defines a template that an application can use to create dialog boxes.
FONT	Specifies the name of a file that contains a font.
HTML	Specifies an HTML file.
ICON	Defines an icon or animated icon by naming it and specifying the name of the file that contains it. (To use a particular icon, the application requests it by name.)
MENU	Defines the appearance and function of a menu.
MENUEX	Defines the appearance and function of a menu.
MESSAGETABLE	Defines a message table by naming it and specifying the name of the file that contains it. The file is a binary resource file generated by the message compiler.

Resource	Description
POPUP	Defines a menu item that can contain menu items and submenus.
PLUGPLAY	Obsolete.
RCDATA	Defines data resources. Data resources let you include binary data in the executable file.
STRINGTABLE	Defines string resources. String resources are Unicode or ASCII strings that can be loaded from the executable file.
TEXTINCLUDE	A special resource that is interpreted by Visual C++. For more information, see TN035.
TYPELIB	A special resource that is used with the /TLBID and /TLBOUT linker options.
User-Defined	Defines a resource that contains application-specific data.
VERSIONINFO	Defines a version-information resource. Contains information such as the version number, intended operating system, and so on.
VXD	Obsolete.

For more information about predefined MFC resources, see TN023 and TN024.

Controls

Control	Description
AUTO3STATE	Creates an automatic three-state check box control.
AUTOCHECKBOX	Creates an automatic check box control.
AUTORADIOBUTTON	Creates an automatic radio button control.
CHECKBOX	Creates a check box control.
COMBOBOX	Creates a combo box control.
CONTROL	Creates an application-defined control.
CTEXT	Creates a centered-text control.
DEFPUSHBUTTON	Creates a default pushbutton control.
EDITTEXT	Creates an edit control.

Control	Description
GROUPBOX	Creates a group box control.
ICON	Creates an icon control. This control is an icon displayed in a dialog box.
LISTBOX	Creates a list box control.
LTEXT	Creates a left-aligned text control.
PUSHBOX	Creates a push box control.
PUSHBUTTON	Creates a push button control.
RADIOBUTTON	Creates a radio button control.
RTEXT	Creates a right-aligned control.
SCROLLBAR	Creates a scroll bar control.
STATE3	Creates a three-state check box control.

Statements

Statement	Description
CAPTION	Sets the title for a dialog box.
CHARACTERISTICS	Specifies information about a resource that can be used by tool that can read or write resource-definition files.
CLASS	Sets the class of the dialog box.
EXSTYLE	Sets the extended window style of the dialog box.
FONT	Sets the font with which the system will draw text for the dialog box.
LANGUAGE	Sets the language for all resources up to the next LANGUAGE statement or to the end of the file. When the **LANGUAGE** statement appears before the beginning of the body of an ACCELERATORS, DIALOG, MENU, RCDATA, or STRINGTABLE resource definition, the specified language applies only to that resource.
MENU	Sets the menu for the dialog box.
MENUITEM	Defines a menu item.
STYLE	Sets the window style for the dialog box.

Statement	Description
VERSION	Specifies version information for a resource that can be used by tool that can read or write resource-definition files.

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

ACCELERATORS resource

Article • 08/19/2020

Defines one or more accelerators for an application. An accelerator is a keystroke defined by the application to give the user a quick way to perform a task.

syntax

```
acctablename ACCELERATORS [optional-statements] {event, idvalue, [type]
[options]... }
```

Parameters

acctablename

Unique name or a 16-bit unsigned integer value that identifies the resource.

optional-statements

Zero or more of the following statements.

Statement	Description
CHARACTERISTICS *dword*	User-defined information about a resource that can be used by tools that read and write resource files. For more information, see CHARACTERISTICS.
LANGUAGE *language*, *sublanguage*	Specifies the language for the resource. For more information, see LANGUAGE.
VERSION *dword*	User-defined version number for the resource that can be used by tools that read and write resource files. For more information, see VERSION.

event

Keystroke to be used as an accelerator. It can be any one of the following character types.

Type	Description
"*char*"	A single character enclosed in double quotation marks ("). The character can be preceded by a caret (^), meaning that the character is a control character.

Type	Description
Character	An integer value representing a character. The *type* parameter must be **ASCII**.
virtual-key character	An integer value representing a virtual key. The virtual key for alphanumeric keys can be specified by placing the uppercase letter or number in double quotation marks (for example, "9" or "C"). The *type* parameter must be **VIRTKEY**.

idvalue

a 16-bit unsigned integer value that identifies the accelerator.

type

Required only when the *event* parameter is a *character* or a *virtual-key character*. The *type* parameter specifies either **ASCII** or **VIRTKEY**; the integer value of *event* is interpreted accordingly. When **VIRTKEY** is specified and *event* contains a string, *event* must be uppercase.

options

options that define the accelerator. This parameter can be one or more of the following values.

Option	Description
NOINVERT	Specifies that no top-level menu item is highlighted when the accelerator is used. This is useful when defining accelerators for actions such as scrolling that do not correspond to a menu item. If **NOINVERT** is omitted, a top-level menu item will be highlighted (if possible) when the accelerator is used. This attribute is obsolete and retained only for backward compatibility with resource files designed for 16-bit Windows.
ALT	Causes the accelerator to be activated only if the ALT key is down. Applies only to virtual keys.
SHIFT	Causes the accelerator to be activated only if the SHIFT key is down. Applies only to virtual keys
CONTROL	Defines the character as a control character (the accelerator is only activated if the CONTROL key is down). This has the same effect as using a caret (^) before the accelerator character in the *event* parameter. Applies only to virtual keys

Certain attributes are also supported for backward compatibility. For more information, see Common Resource Attributes.

Remarks

The **TranslateAccelerator** function is used to translate accelerator messages from the application queue into **WM_COMMAND** or **WM_SYSCOMMAND** messages.

Examples

The following example demonstrates the use of accelerator keys.

syntax

```
1 ACCELERATORS
{
  "^C",  IDDCLEAR         ; control C
  "K",   IDDCLEAR         ; shift K
  "k",   IDDELLIPSE, ALT  ; alt k
  98,    IDDRECT, ASCII   ; b
  66,    IDDSTAR, ASCII   ; B (shift b)
  "g",   IDDRECT          ; g
  "G",   IDDSTAR          ; G (shift G)
  VK_F1, IDDCLEAR, VIRTKEY                ; F1
  VK_F1, IDDSTAR, CONTROL, VIRTKEY        ; control F1
  VK_F1, IDDELLIPSE, SHIFT, VIRTKEY       ; shift F1
  VK_F1, IDDRECT, ALT, VIRTKEY            ; alt F1
  VK_F2, IDDCLEAR, ALT, SHIFT, VIRTKEY    ; alt shift F2
  VK_F2, IDDSTAR, CONTROL, SHIFT, VIRTKEY ; ctrl shift F2
  VK_F2, IDDRECT, ALT, CONTROL, VIRTKEY   ; alt control F2
}
```

See also

Using Keyboard Accelerators

TranslateAccelerator

CHARACTERISTICS

DIALOG

LANGUAGE

MENU

RCDATA

STRINGTABLE

VERSION

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

AUTO3STATE control

Article • 08/19/2020

Defines an automatic three-state check box. The control is an open box with the given text positioned to the right of the box. When chosen, the box automatically advances between three states: checked, unchecked, and disabled (grayed). The control sends a message to its parent whenever the user chooses the control.

syntax

```
AUTO3STATE text, id, x, y, width, height [, style [, extended-style]]
```

style

Styles for the control, which can be a combination of the **BS_AUTO3STATE** style and the following styles: **WS_TABSTOP**, **WS_DISABLED**, and **WS_GROUP**.

If you do not specify a style, the default style is `BS_AUTO3STATE | WS_TABSTOP`.

For more information about the general syntax of a control statement, see Common Control Parameters.

See also

AUTOCHECKBOX

Check Boxes

CHECKBOX

CONTROL

STATE3

Window Styles

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

AUTOCHECKBOX control

Article • 08/19/2020

Defines an automatic check box control. The control is a small rectangle (check box) that has the specified text displayed next to it (typically, to the right). When the user chooses the control, the control highlights the rectangle and sends a message to its parent window.

The **AUTOCHECKBOX** statement can only be used in the body of a **DIALOG** or **DIALOGEX** statement.

syntax

```
AUTOCHECKBOX text, id, x, y, width, height [, style [, extended-style]]
```

style

Styles of the control. This value can be a combination of the button class style **BS_AUTOCHECKBOX** and the **WS_TABSTOP** and **WS_GROUP** styles.

If you do not specify a style, the default style is `BS_AUTOCHECKBOX | WS_TABSTOP`.

For more information about the general syntax of a control statement, see Common Control Parameters.

See also

AUTO3STATE

Button Styles

CHECKBOX

CONTROL

STATE3

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

AUTORADIOBUTTON control

Article • 10/24/2019

Defines an automatic radio button control. This control automatically performs mutual exclusion with the other **AUTORADIOBUTTON** controls in the same group. When the button is chosen, the application is notified with **BN_CLICKED**.

syntax

```
AUTORADIOBUTTON text, id, x, y, width, height [, style [, extended-style]]
```

text

Text that will appear next to the radio button.

style

Styles for the automatic radio button, which can be a combination of BUTTON-class styles and the following styles: **WS_TABSTOP**, **WS_DISABLED**, and **WS_GROUP**.

If you do not specify a style, the default style is `BS_AUTORADIOBUTTON | WS_TABSTOP`.

For more information about the general syntax of a control statement, see Common Control Parameters.

See also

CONTROL

Radio Buttons

RADIOBUTTON

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

BITMAP resource

Article • 08/19/2020

Defines a bitmap that an application uses in its screen display or as an item in a menu or control.

syntax

```
nameID BITMAP filename
```

Parameters

nameID

Unique name or a 16-bit unsigned integer value identifying the resource.

filename

Name of the file that contains the resource. The name must be a valid file name; it must be a full path if the file is not in the current working directory. The path should be a quoted string.

Certain attributes are also supported for backward compatibility. For more information, see Common Resource Attributes.

Examples

The following example defines two bitmap resources:

syntax

```
disk1   BITMAP "disk.bmp"
12      BITMAP "diskette.bmp"
```

See also

Using Bitmaps

LoadBitmap

LoadImage

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

CAPTION statement

Article • 10/24/2019

Defines the title for a dialog box. The title appears in the box's caption bar (if it has one).

The default caption is empty.

syntax

```
CAPTION "captiontext"
```

captiontext

A character string enclosed in double quotation marks (").

Examples

The following example demonstrates the use of the **CAPTION** statement:

syntax

```
CAPTION "Error!"
```

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

CHARACTERISTICS statement

Article • 10/24/2019

Defines information about a resource that can be used by tools that read and write resource-definition files. The specified **DWORD** value appears with the resource in the compiled .res file. However, the value is not stored in the executable file and has no significance to the system.

The **CHARACTERISTICS** statement appears before the beginning of the body of an ACCELERATORS, DIALOG, MENU, RCDATA, or STRINGTABLE resource definition. The specified value applies only to that resource.

syntax

```
CHARACTERISTICS dword
```

dword

User-defined **DWORD** value.

See also

ACCELERATORS

DIALOG

LANGUAGE

MENU

RCDATA

STRINGTABLE

Feedback

Get help at Microsoft Q&A

CHECKBOX control

Article • 08/19/2020

Defines a check box control. The control is a small rectangle (check box) that has the specified text displayed next to it (typically, to the right). When the user selects the control, the control highlights the rectangle and sends a message to its parent window.

The **CHECKBOX** statement, which can only be used in a **DIALOGEX** statement, defines the text, identifier, dimensions, and attributes of the control.

syntax

```
CHECKBOX text, id, x, y, width, height [, style [, extended-style]]
```

text

Text that is to be displayed to the right of the control.

style

Control styles. This value can be a combination of the button class style **BS_CHECKBOX** and the **WS_TABSTOP** and **WS_GROUP** styles.

If you do not specify a style, the default style is `BS_CHECKBOX | WS_TABSTOP`.

For more information about the general syntax of a control statement, see Common Control Parameters.

Examples

This example defines a check-box control that is labeled Italic:

syntax

```
CHECKBOX "Italic", 3, 10, 10, 40, 10
```

See also

AUTOCHECKBOX

AUTO3STATE

Check Boxes

GetDialogBaseUnits

STATE3

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

CLASS statement

Article • 08/19/2020

Defines the class of the dialog box.

The **CLASS** statement appears in the optional section before a DIALOG statement's main. If no class is given, the standard dialog class is used.

syntax

```
CLASS class
```

class

A 16-bit unsigned integer or a string, enclosed in double quotation marks ("), that identifies the class of the dialog box. If the window procedure for the class does not process a message sent to it, it must call the DefDlgProc function to ensure that all messages are handled properly for the dialog box. A private class can use **DefDlgProc** as the default window procedure. The class must be registered with the **cbWndExtra** member of the WNDCLASS structure set to **DLGWINDOWEXTRA**.

Remarks

The **CLASS** statement should only be used with special cases, because it overrides the normal processing of a dialog box. The **CLASS** statement converts a dialog box to a window of the specified class; depending on the class, this could give undesirable results. Do not use the redefined control-class names with this statement.

Examples

The following example demonstrates the use of the **CLASS** statement:

syntax

```
CLASS "myclass"
```

See also

DefDlgProc

DIALOG

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

COMBOBOX control

Article • 08/19/2020

Defines a combination box control (a combo box). A combo box consists of either a static text box or an edit box combined with a list box. The list box can be displayed at all times or pulled down by the user. If the combo box contains a static text box, the text box always displays the selection (if any) in the list box portion of the combo box. If it uses an edit box, the user can type in the desired selection; the list box highlights the first item (if any) that matches what the user has entered in the edit box. The user can then select the item highlighted in the list box to complete the choice. In addition, the combo box can be owner-drawn and of fixed or variable height.

syntax

```
COMBOBOX id, x, y, width, height [, style [, extended-style]]
```

style

Control styles. This value can be a combination of the COMBOBOX class styles and any of the following styles: **WS_TABSTOP**, **WS_GROUP**, **WS_VSCROLL**, and **WS_DISABLED**.

If you do not specify a style, the default style is `CBS_SIMPLE | WS_TABSTOP`.

The *height* parameter applies to the height of a combo box with a drop-down list fully expanded.

For more information about the general syntax of a control statement, see Common Control Parameters.

Examples

This example defines a combo-box control with a vertical scroll bar:

syntax

```
COMBOBOX 777, 10, 10, 50, 54, CBS_SIMPLE | WS_VSCROLL | WS_TABSTOP
```

See also

Combo Boxes

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

CONTROL control

Article • 08/19/2020

Defines a user-defined control.

syntax

```
CONTROL text, id, class, style, x, y, width, height [, extended-style]
```

class

Redefined name, character string, or a 16-bit unsigned integer value that defines the class. This can be any one of the control classes; for a list of the control classes, see the first list following this description. If the value is a redefined name supplied by the application, it must be a string enclosed in double quotation marks (").

style

Redefined name or integer value that specifies the style of the given control. The exact meaning of *style* depends on the *class* value. The sections following this description show the control classes and corresponding styles.

For more information about the general syntax of a control statement, see Common Control Parameters.

Remarks

The six possible control classes are described in the following sections.

The Button Control Class

A button control is a small rectangular child window that the user can turn on or off by clicking it with the mouse. Button controls can be used alone or in groups, and can either be labeled or appear without text. Button controls typically change appearance when the user clicks them.

The button styles are described in the following topic: Button Styles.

The Combo Box Control Class

Combo box controls consist of a selection field similar to an edit control plus a list box. The list box may be displayed at all times or may be dropped down when the user selects a "pop box" next to the selection field.

Depending on the style of the combo box, the user can or cannot edit the contents of the selection field. If the list box is visible, typing characters into the selection box will cause the first entry that matches the characters typed to be highlighted. Conversely, selecting an item in the list box displays the selected text in the selection field.

The combo box control styles are described in the following topic: Combo Box Styles.

The Edit Control Class

An edit control is a rectangular child window in which the user can enter text from the keyboard. The user selects the control, and gives it the input focus, by clicking the mouse inside it or pressing the TAB key. The user can enter text when the control displays a flashing insertion point. The mouse can be used to move the cursor and select characters to be replaced, or to position the cursor for inserting characters. The BACKSPACE key can be used to delete characters.

Edit controls use the fixed-pitch font and display Unicode characters. They expand tab characters into as many space characters as are required to move the cursor to the next tab stop. Tab stops are assumed to be at every eighth character position.

The edit control styles are described in the following topic: Edit Control Styles.

The List Box Control Class

List box controls consist of a list of character strings. The control is used whenever an application needs to present a list of names, such as filenames, that the user can view and select. The user can select a string by pointing to the string with the mouse and clicking a mouse button. When a string is selected, it is highlighted and a notification message is passed to the parent window. A scroll bar can be used with a list box control to scroll lists that are too long or too wide for the control window.

The list box control styles are described in the following topic: List Box Styles.

The Scroll-Bar Control Class

A scroll-bar control is a rectangle that contains a scroll thumb and has direction arrows at both ends. The scroll bar sends a notification message to its parent whenever the user clicks the mouse in the control. The parent is responsible for updating the thumb

position, if necessary. Scroll-bar controls have the same appearance and function as the scroll bars used in ordinary windows. But unlike scroll bars, scroll-bar controls can be positioned anywhere within a window and used whenever needed to provide scrolling input for a window.

The scroll bar styles are described in the following topic: Scroll Bar Control Styles.

The Static Control Class

Static controls are simple text fields, boxes, and rectangles that can be used to label, box, or separate other controls. Static controls take no input and provide no output.

The static control styles are described in the following topic: Static Control Styles.

Feedback

CTEXT control

Article • 08/19/2020

Defines a centered-text control. The control is a simple rectangle displaying the given text centered in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line. Words that are longer than the width of the control are truncated.

The **LTEXT** statement, which can be used only in a rep statement, defines the text, identifier, dimensions, and attributes of the control.

syntax

```
CTEXT text, id, x, y, width, height [, style [, extended-style]]
```

text

Text that is to be centered in the rectangular area of the control.

style

Control styles. This value can be any combination of the following styles: **SS_CENTER**, **WS_TABSTOP**, and **WS_GROUP**.

If you do not specify a style, the default style is `SS_CENTER | WS_GROUP`.

For more information about the general syntax of a control statement, see Common Control Parameters.

Examples

This example defines a centered-text control that is labeled Filename:

syntax

```
CTEXT "Filename", 101, 10, 10, 100, 100
```

See also

CONTROL

Edit Controls

LTEXT

RTEXT

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

CURSOR resource

Article • 08/19/2020

Defines a bitmap that defines the shape of the cursor on the display screen or an animated cursor.

syntax

```
nameID CURSOR filename
```

Parameters

nameID

Unique name or a 16-bit unsigned integer identifying the resource.

filename

Name of the file that contains the resource. The name must be a valid file name; it must be a full path if the file is not in the current working directory. The path should be a quoted string.

Certain attributes are also supported for backward compatibility. For more information, see Common Resource Attributes.

Remarks

Icon and cursor resources can contain more than one image.

Examples

The following example defines two cursor resources; one by name (cursor1) and the other by number (2):

syntax

```
cursor1 CURSOR "bullseye.cur"
2       CURSOR "d:\\cursor\\arrow.cur"
```

See also

Using Cursors

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

DEFPUSHBUTTON control

Article • 10/24/2019

Defines a default push-button control. The control is a small rectangle with a bold outline that represents the default response for the user. The given text is displayed inside the button. The control highlights the button in the usual way when the user clicks the mouse in it and sends a message to its parent window.

syntax

```
DEFPUSHBUTTON text, id, x, y, width, height [, style [, extended-style]]
```

text

Text that is to be centered in the rectangular area of the control.

style

Control styles. This value can be a combination of the following styles: **BS_DEFPUSHBUTTON**, **WS_TABSTOP**, **WS_GROUP**, and **WS_DISABLED**.

If you do not specify a style, the default style is `BS_DEFPUSHBUTTON | WS_TABSTOP`.

For more information about the general syntax of a control statement, see Common Control Parameters.

Examples

This example defines a default push-button control that is labeled Cancel:

syntax

```
DEFPUSHBUTTON "Cancel", 101, 10, 10, 24, 50
```

See also

Push Buttons

PUSHBUTTON

RADIOBUTTON

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

DIALOG resource

Article • 03/09/2021

Defines a dialog box. The statement defines the position and dimensions of the dialog box on the screen as well as the dialog box style.

> ⓘ **Note**
>
> **DIALOG** is an obsolete resource ID. New applications should use **DIALOGEX**.

syntax

```
nameID DIALOG x, y, width, height  [optional-statements] {control-statement
. . . }
```

Parameters

nameID

Unique name or a unique 16-bit unsigned integer value that identifies the dialog box.

optional-statements

Options for the dialog box. This can be zero or more of the following statements.

Statement	Description
CAPTION "*text*"	Caption of the dialog box if it has a title bar. For more information, see CAPTION.
CHARACTERISTICS *dword*	User-defined **DWORD** value for use by resource tools. This value is not used by the system. For more information, see CHARACTERISTICS.
CLASS *class*	A 16-bit unsigned integer or a string, enclosed in double quotation marks ("), that identifies the class of the dialog box. For more information, see CLASS.
EXSTYLE= *extended-styles*	Extended window style of the dialog box. For more information, see EXSTYLE.
FONT *pointsize*, *typeface*	Point size and typeface for the font. For more information, see FONT.

Statement	Description
LANGUAGE *language*, *sublanguage*	Language of the dialog box. For more information, see **LANGUAGE**.
MENU *menuname*	Menu to be used. This value is either the name of the menu or its integer identifier.
STYLE *styles*	Styles of the dialog box. For more information, see **STYLE**.
VERSION *dword*	User-defined **DWORD** value. This statement is intended for use by additional resource tools and is not used by the system. For more information, see **VERSION**.

Certain attributes are also supported for backward compatibility. For more information, see Common Resource Attributes.

Remarks

The **GetDialogBaseUnits** function returns the dialog base units in pixels. The exact meaning of the coordinates depends on the style defined by the **STYLE** option statement. For child-style dialog boxes, the coordinates are relative to the origin of the parent window, unless the dialog box has the style **DS_ABSALIGN**; in that case, the coordinates are relative to the origin of the display screen.

Do not use the **WS_CHILD** style with a modal dialog box. The **DialogBox** function always disables the parent/owner of the newly created dialog box. When a parent window is disabled, its child windows are implicitly disabled. Since the parent window of the child-style dialog box is disabled, the child-style dialog box is too.

If a dialog box has the **DS_ABSALIGN** style, the dialog coordinates for its upper-left corner are relative to the screen origin instead of to the upper-left corner of the parent window. You would typically use this style when you wanted the dialog box to start in a specific part of the display no matter where the parent window may be on the screen.

The name **DIALOG** can also be used as the class-name parameter to the **CreateWindow** function to create a window with dialog-box attributes.

Examples

The following demonstrates the usage of the **DIALOG** statement:

syntax

```
#include <windows.h>

ErrorDialog DIALOG  10, 10, 300, 110
STYLE WS_POPUP | WS_BORDER
CAPTION "Error!"
{
    CTEXT "Select One:", 1, 10, 10, 280, 12
    PUSHBUTTON "&Retry", 2, 75, 30, 60, 12
    PUSHBUTTON "&Abort", 3, 75, 50, 60, 12
    PUSHBUTTON "&Ignore", 4, 75, 80, 60, 12
}
```

See also

Using Dialog Boxes

ACCELERATORS

CHARACTERISTICS

CONTROL

CreateDialog

CreateWindow

DialogBox

GetDialogBaseUnits

LANGUAGE

MENU

RCDATA

STRINGTABLE

VERSION

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

DIALOGEX resource

Article • 03/09/2021

Defines a dialog box. The statement defines the position and dimensions of the dialog box on the screen as well as the dialog box style. It also defines the following:

- Help IDs on the dialog itself as well as on controls within the dialog box.
- Use of the **EXSTYLE** statement for the dialog box itself as well as on controls within the dialog box.
- Font weight and italic settings for the font to be used in the dialog box.
- Control-specific data for controls within the dialog box.
- Use of the **BEDIT**, **IEDIT**, and **HEDIT** predefined system class names.

syntax

```
nameID DIALOGEX x, y, width, height [ , helpID] [optional-statements]
{control-statements}
```

Parameters

nameID

Unique name or a unique 16-bit unsigned integer value that identifies the dialog box.

x

Location on the screen of the left side of the dialog box, in dialog units.

y

Location on the screen of the top of the dialog box, in dialog units.

width

Width of the dialog box, in dialog units.

height

Height of the dialog box, in dialog units.

helpID

Numeric expression indicating the ID used to identify the dialog box during **WM_HELP** processing.

optional-statements

Options for the dialog box. This can be zero or more of the following statements.

Statement	Description
CAPTION "*text*"	Caption of the dialog box if it has a title bar. For more information, see **CAPTION Statement**.
CHARACTERISTICS *dword*	User-defined **DWORD** value for use by resource tools. This value is not used by the system. For more information, see **CHARACTERISTICS Statement**.
CLASS*class*	A 16-bit unsigned integer or a string, enclosed in double quotation marks ("), that identifies the class of the dialog box. For more information, see **CLASS Statement**.
EXSTYLE= *extended-styles*	Extended window style of the dialog box. For more information, see **EXSTYLE Statement**.
FONT *pointsize*, "*typeface*", *weight*, *italic*, *charset*	Point size and typeface for the font. For *weight*, use the **FW_*** values defined in WinGDI.h. For *italic*, specify TRUE to use an italic font, FALSE otherwise. For *charset*, use the value defined in the **lfCharSet** member of the **LOGFONT** structure. To get the definitive font for a dialog box, an application should specify *charset* along with other font properties. For more information, see **FONT Statement**.
LANGUAGE *language*, *sublanguage*	Language of the dialog box. For more information, see **LANGUAGE Statement**.
MENU *menuname*	Menu to be used. This value is either the name of the menu or its integer identifier. For more information, see **MENU Statement**.
STYLE *styles*	Styles of the dialog box. For more information, see **STYLE Statement**.
VERSION *dword*	User-defined **DWORD** value. This statement is intended for use by additional resource tools and is not used by the system. For more information, see **VERSION Statement**.

control-statements

Body of the **DIALOGEX** resource is made up of any number of control statements. There are four families of control statements: generic, static, button, and edit. For more information, see Remarks.

Certain attributes are also supported for backward compatibility. For more information, see Common Resource Attributes.

Remarks

The valid operations that may be contained in any of the numeric expressions in the statements of **DIALOGEX** are as follows:

- Add ('+')
- Subtract ('-')
- Unary minus ('-')
- Unary NOT ('~')
- AND ('&')
- OR ('|')

The body of the resource is made up of generic, static, button, and edit control statements. While each of these families of statements uses a different syntax for defining specific features of its controls, they all share a common syntax for defining the position, size, extended styles, help identification number, and control-specific data. For more information, see Common Control Parameters.

Generic Control Statements

syntax

```
CONTROL controlText, id, className, style
```

controlText

Window text for the control. For more information, see Common Control Parameters.

id

Control identifier. For more information, see Common Control Parameters.

className

Name of the class. This may be either a string enclosed in double quotation marks (") or one of the following predefined system classes: **BUTTON**, **STATIC**, **EDIT**, **LISTBOX**, **SCROLLBAR**, or **COMBOBOX**.

style

Window styles (explicit **WS_***, **BS_***, **SS_***, **ES_***, **LBS_***, **SBS_***, and **CBS_*** style values defined in Winuser.H can be used by adding an include to the .rc file: `#include "winuser.h"`). For more information, see Window Styles.

Static Control Statements

syntax

```
staticClass controlText, id
```

staticClass

LTEXT, **RTEXT**, or **CTEXT**.

controlText

Window text for the control. For more information, see Common Control Parameters.

id

Control identifier. For more information, see Common Control Parameters.

Button Control Statements

syntax

```
buttonClass controlText, id
```

buttonClass

AUTO3STATE, **AUTOCHECKBOX**, **AUTORADIOBUTTON**, **CHECKBOX**, **PUSHBOX**, **PUSHBUTTON**, **RADIOBUTTON**, **STATE3**, or **USERBUTTON**.

controlText

Window text for the control. For more information, see Common Control Parameters.

id

Control identifier. For more information, see Common Control Parameters.

Edit Control Statements

syntax

```
editClass id
```

editClass

EDITTEXT, **BEDIT**, **HEDIT**, or **IEDIT**.

id

Control identifier. For more information, see Common Control Parameters.

See also

Using Dialog Boxes

ACCELERATORS

CHARACTERISTICS

CONTROL

CreateDialog

CreateWindow

DialogBox

GetDialogBaseUnits

LANGUAGE

LOGFONT

MENU

RCDATA

STRINGTABLE

VERSION

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

EDITTEXT control

Article • 08/19/2020

Defines an edit control belonging to the EDIT class. It creates a rectangular region in which the user can type and edit text. The control displays a cursor when the user clicks the mouse in it. The user can then use the keyboard to enter text or edit the existing text. Editing keys include the BACKSPACE and DELETE keys. The user can also use the mouse to select characters to be deleted or to select the place to insert new characters.

syntax

```
EDITTEXT id, x, y, width, height [, style [, extended-style]]
```

style

Control styles. This value can be a combination of the edit class styles and the following styles: **WS_TABSTOP**, **WS_GROUP**, **WS_VSCROLL**, **WS_HSCROLL**, and **WS_DISABLED**.

If you do not specify a style, the default style is `ES_LEFT | WS_BORDER | WS_TABSTOP`.

For more information about the general syntax of a control statement, see Common Control Parameters.

Examples

The following example demonstrates the use of the **EDITTEXT** statement:

syntax

```
EDITTEXT  3, 10, 10, 100, 10
```

See also

Edit Controls

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

EXSTYLE statement

Article • 08/19/2020

Defines extended window styles for a dialog box. In a resource definition, the **EXSTYLE** statement is placed with the optional statements before the beginning of the body of the resource definition.

syntax

```
EXSTYLE extended-style
```

extended-style

Extended window style for the dialog box or control. For a list of extended window styles, see **Extended Window Styles**.

Remarks

For controls, extended styles are specified after the *style* option in the control resource-definition statement. For more information, see the resource-definition statement for the individual control.

See also

ACCELERATORS

CONTROL

DIALOG

MENU

POPUP

RCDATA

STRINGTABLE

User-Defined Resource

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

FONT resource

Article • 10/24/2019

Defines a file that contains a font.

syntax

```
nameID FONT filename
```

Parameters

nameID

Unique 16-bit unsigned integer value identifying the resource.

filename

Name of the file that contains the resource. The name must be a valid file name; it must be a full path if the file is not in the current working directory. The path should be a quoted string.

Certain attributes are also supported for backward compatibility. For more information, see Common Resource Attributes.

Examples

The following example defines a single font resource:

syntax

```
5 FONT  "cmroman.fnt"
```

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

FONT statement

Article • 08/19/2020

Defines the font with which the system will draw text in the dialog box. The font must have been previously loaded; for example, by calling the LoadResource function.

syntax

```
FONT pointsize, "typeface", weight, italic, charset
```

pointsize

The size of the font, in points.

typeface

The name of the typeface. This parameter must be enclosed in quotes (").

weight

The weight of the font in the range 0 through 1000. For example, 400 is normal and 700 is bold. If this value is 0, the default weight is used. For a list of predefined values, see the **FW_*** values defined in WinGDI.h.

italic

Indicates an italic font if set to TRUE.

charset

The character set. For a list of values, see the **lfCharSet** member of the LOGFONT structure.

Examples

The following example demonstrates the use of the **FONT** statement:

syntax

```
FONT 12, "MS Shell Dlg"
```

See also

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

GROUPBOX control

Article • 11/12/2019

Defines a group box control. The control is a rectangle that groups other controls together. The controls are grouped by drawing a border around them and displaying the given text in the upper-left corner.

The **GROUPBOX** statement, which you can use only in a **DIALOGEX** statement, defines the text, identifier, dimensions, and attributes of a control window.

syntax

```
GROUPBOX text, id, x, y, width, height [, style [, extended-style]]
```

style

Control styles. This value can be a combination of the button class style **BS_GROUPBOX** and the **WS_TABSTOP** and **WS_DISABLED** styles.

If you do not specify a style, the default style is **BS_GROUPBOX**.

For more information about the general syntax of a control statement, see Common Control Parameters.

Remarks

When the style contains **WS_TABSTOP** or the text specifies an accelerator, tabbing or pressing the accelerator key moves the focus to the first control within the group.

Examples

This example defines a group-box control that is labeled Options:

syntax

```
GROUPBOX "Options", 101, 10, 10, 100, 100
```

See also

Group Boxes

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

HTML resource

Article • 10/24/2019

Defines an HTML file.

syntax

```
nameID HTML filename
```

Parameters

nameID

Unique name or a 16-bit unsigned integer value identifying the resource.

filename

The name of the HTML file. It must be a full or relative path if the file is not in the current working directory. The path should be a quoted string.

Examples

The following example defines an HTML resource:

syntax

```
ID_RESPONSE_ERROR_PAGE  HTML "res\\responseerorpage.htm"
```

Feedback

ICON resource

Article • 08/19/2020

Defines a bitmap that defines the shape of the icon to be used for a given application or an animated icon.

syntax

```
nameID ICON filename
```

Parameters

nameID

Unique name or a 16-bit unsigned integer value identifying the resource.

filename

Name of the file that contains the resource. The name must be a valid file name; it must be a full path if the file is not in the current working directory. The path should be a quoted string.

Certain attributes are also supported for backward compatibility. For more information, see Common Resource Attributes.

Remarks

Icon and cursor resources can contain more than one image.

Examples

The following example defines two icon resources:

syntax

```
desk1   ICON "desk.ico"
11      ICON "custom.ico"
```

See also

Using Icons

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

ICON control

Article • 10/24/2019

Defines an icon control. This control is an icon displayed in a dialog box.

The **ICON** control statement, which you can use only in a **DIALOGEX** statement, defines the icon-resource identifier, icon-control identifier, position, and attributes of a control.

syntax

```
ICON text, id, x, y [, width, height, style [, extended-style]]
```

text

Name of an icon (not a file name) defined elsewhere in the resource file.

width

This value is ignored and should be set to zero.

height

This value is ignored and should be set to zero.

style

Control style. This parameter is optional. The only value that can be specified is the **SS_ICON** style. This is the default style whether this parameter is specified or not.

For more information about the general syntax of a control statement, see Common Control Parameters.

Examples

This example defines an icon control whose icon identifier is 901 and whose name is myicon:

syntax

```
ICON "myicon", 901, 30, 30
```

See also

ICON

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

LANGUAGE statement

Article • 08/19/2020

Defines the language for all resources up to the next **LANGUAGE** statement or to the end of the file.

When the **LANGUAGE** statement appears before the beginning of the body of an **ACCELERATORS**, **DIALOGEX**, **MENU**, **RCDATA**, or **STRINGTABLE** resource definition, the specified language applies only to that resource.

syntax

```
LANGUAGE language, sublanguage
```

language

Language identifier.

sublanguage

Sublanguage identifier.

For more information, see Language Identifier Constants and Strings.

See also

ACCELERATORS

CHARACTERISTICS

MENU

RCDATA

STRINGTABLE

VERSION

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

LISTBOX control

Article • 08/19/2020

Defines commonly used controls for a dialog box or window. The control is a rectangle containing a list of strings (such as filenames) from which the user can select.

The **LISTBOX** statement, which can only be used in a **DIALOGEX** or **WINDOW** statement, defines the identifier, dimensions, and attributes of a control window.

syntax

```
LISTBOX id, x, y, width, height [, style [, extended-style]]
```

style

Control styles. This value can be a combination of the list-box class styles and any of the following styles: **WS_BORDER** and **WS_VSCROLL**.

If you do not specify a style, the default style is `LBS_NOTIFY | WS_BORDER`.

For more information about the general syntax of a control statement, see Common Control Parameters.

Examples

This example defines a list-box control whose identifier is 101:

syntax

```
LISTBOX 101, 10, 10, 100, 100
```

See also

COMBOBOX

List Boxes

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

LTEXT control

Article • 08/19/2020

Defines a left-aligned text control. The control is a simple rectangle displaying the given text left-aligned in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line. Words that are longer than the width of the control are truncated.

The **LTEXT** statement, which can be used only in a **DIALOGEX** statement, defines the text, identifier, dimensions, and attributes of the control.

syntax

```
LTEXT text, id, x, y, width, height [, style [, extended-style]]
```

style

Control styles. This value can be any combination of the **BS_RADIOBUTTON** style and the following styles: **SS_LEFT**, **WS_TABSTOP**, and **WS_GROUP**.

If you do not specify a style, the default style is `SS_LEFT | WS_GROUP`.

For more information about the general syntax of a control statement, see Common Control Parameters.

Examples

This example defines a left-aligned text control that is labeled Filename:

syntax

```
LTEXT "Filename", 101, 10, 10, 100, 100
```

See also

CONTROL

CTEXT

Edit Controls

RTEXT

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

MENU resource

Article • 03/09/2021

Defines the contents of a menu resource. A menu resource is a collection of information that defines the appearance and function of an application menu. A menu is a special input tool that lets a user select commands and open submenus from a list of menu items.

syntax

```
menuID MENU  [optional-statements]  {item-definitions ... }
```

Parameters

menuID

Number that identifies the menu. This value is either a unique string or a unique 16-bit unsigned integer value in the range of 1 to 65,535.

optional-statements

This parameter can be zero of more of the following statements.

Statement	Description
CHARACTERISTICS *dword*	User-defined information about a resource that can be used by tools that read and write resource files. For more information, see CHARACTERISTICS.
LANGUAGE *language*, *sublanguage*	Language for the resource. For more information, see LANGUAGE.
VERSION *dword*	User-defined version number for the resource that can be used by tools that read and write resource files. For more information, see VERSION.

Certain attributes are also supported for backward compatibility. For more information, see Common Resource Attributes.

Examples

The following is an example of a complete **MENU** statement:

syntax

```
sample MENU
{
     MENUITEM "&Soup", 100
     MENUITEM "S&alad", 101
     POPUP "&Entree"
     {
          MENUITEM "&Fish", 200
          MENUITEM "&Chicken", 201, CHECKED
          POPUP "&Beef"
          {
               MENUITEM "&Steak", 301
               MENUITEM "&Prime Rib", 302
          }
     }
     MENUITEM "&Dessert", 103
}
```

See also

Using Menus

ACCELERATORS

CHARACTERISTICS

LANGUAGE

MENUEX

MENUITEM

POPUP

RCDATA

STRINGTABLE

VERSION

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

MENU statement

Article • 10/24/2019

Defines the menu for a dialog box. If no statement is given, the dialog box has no menu.

syntax

```
MENU menuname
```

menuname

The name or identifier of the menu to be used.

Examples

The following example demonstrates the use of the **MENU** dialog statement:

syntax

```
MENU errmenu
```

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

MENUEX resource

Article • 03/09/2021

Defines the contents of a menu resource. A menu resource is a collection of information that defines the appearance and function of an application menu. A menu is a special input tool that lets a user select commands and open submenus from a list of menu items. It also defines the following:

- Help identifiers on menus.
- Identifiers on menus.
- Use of the **MFT_*** type flags and **MFS_*** state flags. For more information on these flags, see the MENUITEMINFO structure.

syntax

```
menuID MENUEX{ [{[MENUITEM itemText [,[id][, [type][, state]]]] |
    POPUP itemText [,[id][, [type][, [state][, helpID]]]] { popupBody } } .
. .]}
```

Parameters

MENUITEM

Defines a menu item.

itemText

String containing the text for the menu item. For more information, see MENUITEM.

id

Numeric expression indicating the identifier of the menu item.

type

Numeric expression indicating the type of the menu item To use the predefined MFT_* type values, include the following statement in your .rc file: `#include "winuser.h"`

state

Numeric expression indicating the state of the menu item To use the predefined MFS_* state values, include the following statement in your .RC file: `#include "winuser.h"`

POPUP

Defines a menu item that has a submenu associated with it.

itemText

String containing the text for the menu item.

id

Numeric expression indicating the identifier of the menu item.

type

Numeric expression indicating the type of the menu item To use the predefined MFT_* type values, include the following statement in your .RC file: `#include "winuser.h"`

state

Numeric expression indicating the state of the menu item To use the predefined MFS_* state values, include the following statement in your .rc file: `#include "winuser.h"`

helpID

Numeric expression indicating the identifier used to identify the menu during **WM_HELP** processing.

popupBody

Contains any combination of the **MENUITEM** and **POPUP** statements.

Certain attributes are also supported for backward compatibility. For more information, see Common Resource Attributes.

Remarks

The valid arithmetic and Boolean operations that can be contained in any of the numeric expressions in the statements of **MENUEX** are as follows:

- Add ('+')
- Subtract ('-')
- Unary minus ('-')
- Unary NOT ('~')
- AND ('&')
- OR ('|')

See also

Using Menus

ACCELERATORS

CHARACTERISTICS

DIALOG

LANGUAGE

MENU

MENUITEM

POPUP

RCDATA

STRINGTABLE

VERSION

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

MENUITEM statement

Article • 03/28/2022

Defines a menu item.

syntax

```
MENUITEM text, result, [optionlist]
MENUITEM SEPARATOR
```

text

The name of the menu item.

The string can contain the escape characters **\t** and **\a**. The **\t** character inserts a tab in the string and is used to align text in columns. Tab characters should be used only in menus, not in menu bars. (For information on menus, see **POPUP Resource**.) The **\a** character aligns all text that follows it flush right to the menu bar or pop-up menu.

result

A number that specifies the result generated when the user selects the menu item. This parameter takes an integer value. Menu-item results are always integers; when the user clicks the menu-item name, the result is sent to the window that owns the menu.

optionlist

The appearance of the menu item. This optional parameter takes one or more of the following redefined menu options, separated by commas or spaces.

Option	Description
CHECKED	Menu item has a check mark next to it.
GRAYED	Menu item is initially inactive and appears on the menu in gray or a lightened shade of the menu-text color. This option cannot be used with the **INACTIVE** option.
HELP	Identifies a help item. This option has no effect on the appearance of the menu item.
INACTIVE	Menu item is displayed but it cannot be selected. This option cannot be used with the **GRAYED** option.
MENUBARBREAK	Same as **MENUBREAK** except that for pop-up menus, it separates the new column from the old column with a vertical line.

Option	Description
MENUBREAK	Places the menu item on a new line for static menu-bar items. For menus, it places the menu item in a new column with no dividing line between the columns.

MENUITEM SEPARATOR

The **MENUITEM SEPARATOR** form of the **MENUITEM** statement creates an inactive menu item that serves as a dividing bar between two active menu items on a menu. Note that this form works inside a **MENU** block, whereas **MENUITEM**'s inside a **MENUEX** instead require the form `MENUITEM "", -1, MFT_SEPARATOR`.

Examples

The following example demonstrates the use of the **MENUITEM** and **MENUITEM SEPARATOR** statements:

syntax

```
MENUITEM "&Roman", 206, CHECKED, GRAYED
MENUITEM SEPARATOR
MENUITEM "&Blackletter", 301
```

See also

MENU

POPUP

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

MESSAGETABLE resource

Article • 08/19/2020

Defines the ID and file of an application's message table resource. Message tables are special string resources used in event logging and with the **FormatMessage** function. The file contains a binary message table generated by the message compiler, MC.EXE.

The message compiler also generates a resource script file that contains the **MESSAGETABLE** statements you need to include the message table resources in the compiled resource file. Use the **#include** directive to include this resource script into your main resource script.

syntax

```
nameID MESSAGETABLE filename
```

Parameters

nameID

Unique name or a 16-bit unsigned integer value identifying the resource.

filename

Name of the file that contains the resource. The name must be a valid file name; it must be a full path if the file is not in the current working directory.

Certain attributes are also supported for backward compatibility. For more information, see Common Resource Attributes.

Examples

The following example defines a **MESSAGETABLE** resource:

syntax

```
1  MESSAGETABLE MSG00409.bin
```

See also

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

POPUP resource

Article • 10/24/2019

Defines a menu item that can contain menu items and submenus.

syntax

```
POPUP text, [optionlist] {item-definitions ...}
```

Parameters

text

String that contains the name of the menu. This string must be enclosed in double quotation marks (").

optionlist

This parameter specifies redefined menu options that specify the appearance of the menu item. This optional parameter can be one or more of the following.

Option	Description
CHECKED	Menu item has a check mark next to it. This option is not valid for a top-level menu.
GRAYED	Menu item is initially inactive and appears on the menu in gray or a lightened shade of the menu-text color. This option cannot be used with the **INACTIVE** option.
HELP	Identifies a help item. The menu item is place at the right-most position on the menu bar.
INACTIVE	Menu item is displayed but it cannot be selected. This option cannot be used with the **GRAYED** option.
MENUBARBREAK	Same as **MENUBREAK** except that for pop-up menus, it separates the new column from the old column with a vertical line.
MENUBREAK	Places the menu item on a new line for static menu-bar items. For menus, it places the menu item in a new column with no dividing line between the columns.

Certain attributes are also supported for backward compatibility. For more information, see Common Resource Attributes.

Examples

The following example demonstrates the use of the **POPUP** statement:

syntax

```
chem MENU
{
    POPUP "&Elements"
    {
         MENUITEM "&Oxygen", 200
         MENUITEM "&Carbon", 201, CHECKED
         MENUITEM "&Hydrogen", 202
         MENUITEM SEPARATOR
         MENUITEM "&Sulfur", 203
         MENUITEM "Ch&lorine", 204
    }
    POPUP "&Compounds"
    {
         POPUP "&Sugars"
         {
            MENUITEM "&Glucose", 301
            MENUITEM "&Sucrose", 302, CHECKED
            MENUITEM "&Lactose", 303, MENUBREAK
            MENUITEM "&Fructose", 304
         }
         POPUP "&Acids"
         {
              "&Hydrochloric", 401
              "&Sulfuric", 402
         }
    }
}
```

See also

MENU

MENUITEM

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

PUSHBOX control

Article • 10/24/2019

Defines a push-box control, which is identical to a **PUSHBUTTON**, except that it does not display a button face or frame; only the text appears.

syntax

```
PUSHBOX text, id, x, y, width, height [, style [, extended-style]]
```

style

Styles for the pushbox, which can be a combination of the **BS_PUSHBOX** style and the following styles: **WS_TABSTOP**, **WS_DISABLED**, and **WS_GROUP**.

If you do not specify a style, the default style is `BS_PUSHBOX | WS_TABSTOP`.

For more information about the general syntax of a control statement, see Common Control Parameters.

See also

Push Buttons

PUSHBUTTON

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

PUSHBUTTON control

Article • 08/19/2020

Defines a push-button control. The control is a round-cornered rectangle containing the given text. The text is centered in the control. The control sends a message to its parent whenever the user chooses the control.

syntax

```
PUSHBUTTON text, id, x, y, width, height [, style [, extended-style]]
```

style

Styles for the pushbutton, which can be a combination of the **BS_PUSHBUTTON** style and the following styles: **WS_TABSTOP**, **WS_DISABLED**, and **WS_GROUP**.

If you do not specify a style, the default style is `BS_PUSHBUTTON | WS_TABSTOP`.

For more information about the general syntax of a control statement, see Common Control Parameters.

Examples

The following example demonstrates the use of the **PUSHBUTTON** statement:

syntax

```
PUSHBUTTON "ON", 7, 10, 10, 20, 10
```

See also

GetDialogBaseUnits

Push Buttons

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

RADIOBUTTON control

Article • 08/19/2020

Defines a radio-button control. The control is a small circle that has the given text displayed next to it, typically to its right. The control highlights the circle and sends a message to its parent window when the user selects the button. The control removes the highlight and sends a message when the button is next selected.

syntax

```
RADIOBUTTON text, id, x, y, width, height [, style [, extended-style]]
```

style

Styles for the radio button, which can be a combination of BUTTON-class styles and the following styles: **WS_TABSTOP**, **WS_DISABLED**, and **WS_GROUP**.

If you do not specify a style, the default style is `BS_RADIOBUTTON | WS_TABSTOP`.

For more information about the general syntax of a control statement, see Common Control Parameters.

Examples

The following example demonstrates the use of the **RADIOBUTTON** statement:

syntax

```
RADIOBUTTON "Italic", 100, 10, 10, 40, 10
```

See also

GetDialogBaseUnits

Radio Buttons

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

RCDATA resource

Article • 10/24/2019

Defines a raw data resource for an application. Raw data resources permit the inclusion of binary data directly in the executable file.

syntax

```
nameID RCDATA  [optional-statements] {raw-data  ...}
```

Parameters

nameID

Unique name or a 16-bit unsigned integer value that identifies the resource.

optional-statements

This parameter can be zero or more of the following statements.

Statement	Description
CHARACTERISTICS *dword*	User-defined information about a resource that can be used by tools that read and write resource files. For more information, see CHARACTERISTICS.
LANGUAGE *language*, *sublanguage*	Language for the resource. For more information, see LANGUAGE.
VERSION *dword*	User-defined version number for the resource that can be used by tools that read and write resource files. For more information, see VERSION.

raw-data

Raw data consisting of one or more integers or strings of characters. Integers can be specified in decimal, octal, or hexadecimal format. To be compatible with 16-bit Windows, integers are stored as **WORD** values. You can store an integer as a **DWORD** value by qualifying the integer with the "L" suffix.

Strings are enclosed in quotation marks. RC does not automatically append a terminating null character to a string. Each string is a sequence of the specified ANSI characters, unless you qualify it as a wide-character string with the L prefix.

The block of data begins on a **DWORD** boundary and RC performs no padding or alignment of data within the *raw-data* block. It is your responsibility to ensure the proper alignment of data within the block.

Certain attributes are also supported for backward compatibility. For more information, see Common Resource Attributes.

Examples

The following example demonstrates the use of the **RCDATA** statement:

syntax

```
resname RCDATA
{
   "Here is an ANSI string\0",    // explicitly null-terminated
   L"Here is a Unicode string\0", // explicitly null-terminated
   1024,                          // integer, stored as WORD
   7L,                            // integer, stored as DWORD
   0x029a,                        // hex integer
   0o733,                         // octal integer
}
```

See also

ACCELERATORS

CHARACTERISTICS

LANGUAGE

MENU

STRINGTABLE

VERSION

Feedback

RTEXT control

Article • 08/19/2020

Defines a right-aligned text control. The control is a simple rectangle displaying the given text right-aligned in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line. Words that are longer than the width of the control are truncated.

The **RTEXT** statement, which can be used only in a **DIALOGEX** statement, defines the text, identifier, dimensions, and attributes of the control.

syntax

```
RTEXT text, id, x, y, width, height [, style [, extended-style]]
```

style

Styles for the text control, which can be any combination of the following: **WS_TABSTOP** and **WS_GROUP**.

If you do not specify a style, the default style is `SS_RIGHT | WS_GROUP`.

For more information about the general syntax of a control statement, see Common Control Parameters.

Examples

The following example demonstrates the use of the **RTEXT** statement:

syntax

```
RTEXT "Number of Messages", 4, 30, 50, 100, 10
```

See also

CONTROL

CTEXT

Edit Controls

LTEXT

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

SCROLLBAR control

Article • 08/19/2020

Defines a scroll-bar control. The control is a rectangle that contains a scroll box and has direction arrows at both ends. The scroll-bar control sends a notification message to its parent whenever the user clicks the mouse in the control. The parent is responsible for updating the scroll-box position. Scroll-bar controls can be positioned anywhere in a window and used whenever needed to provide scrolling input.

syntax

```
SCROLLBAR id, x, y, width, height [, style [, extended-style]]
```

style

Zero or a combination of the following styles: **WS_TABSTOP**, **WS_GROUP**, and **WS_DISABLED**.

In addition to these styles, the *style* parameter may contain a combination (or none) of the SCROLLBAR-class styles.

If you do not specify a style, the default style is **SBS_HORZ**.

For more information about the general syntax of a control statement, see Common Control Parameters. Note that you cannot specify text for this control.

Examples

The following example demonstrates the use of the **SCROLLBAR** statement:

syntax

```
#define IDC_SCROLLBARV                  1010

SCROLLBAR IDC_SCROLLBARV, 7, 55, 187, 44, SBS_VERT
```

See also

Scroll Bars

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

STATE3 control

Article • 10/24/2019

Defines a three-state check box control. The control is identical to a **CHECKBOX**, except that it has three states: checked, unchecked, and disabled (grayed).

syntax

```
STATE3 text, id, x, y, width, height [, style [, extended-style]]
```

text

Text that is to be displayed to the right of the control.

style

Control styles. This value can be a combination of the button class style **BS_3STATE** and the **WS_TABSTOP** and **WS_GROUP** styles.

If you do not specify a style, the default style is `BS_3STATE | WS_TABSTOP`.

For more information about the general syntax of a control statement, see Common Control Parameters.

See also

AUTOCHECKBOX

Check Boxes

CHECKBOX

CONTROL

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

STRINGTABLE resource

Article • 05/09/2023

Defines one or more string resources for an application. String resources are simply null-terminated Unicode or ASCII strings that can be loaded when needed from the executable file, using the LoadString function.

There are two ways to format a **STRINGTABLE** statement:

syntax

```
STRINGTABLE  [optional-statements] {stringID string  ...}
```

- or -

syntax

```
STRINGTABLE
  [optional-statements]
BEGIN
stringID string
. . .
END
```

Parameters

optional-statements

This parameter can be zero or more of the following statements.

Statement	Description
CHARACTERISTICS *dword*	User-defined information about a resource that can be used by tools that read and write resource files. For more information, see CHARACTERISTICS.
LANGUAGE *language*, *sublanguage*	Specifies the language for the resource. For more information, see LANGUAGE.
VERSION *dword*	User-defined version number for the resource that can be used by tools that read and write resource files. For more information, see VERSION.

stringID

Unsigned 16-bit integer that identifies the resource.

string

One or more strings, enclosed in quotation marks. The string must be no longer than 4097 characters and must occupy a single line in the source file (unless a '\' is used as a line continuation). To add a carriage return to the string, use this character sequence: \012. For example, "Line one\012Line two" defines a string that is displayed as follows:

```
syntax

Line one
Line two
```

To embed quotes in the string, use the following sequence: "". For example, """Line three""" defines a string that is displayed as follows:

```
syntax

"Line three"
```

To encode Unicode characters, use an "L" followed by the Unicode characters enclosed by quotes. See the Examples section for an example.

The resource compiler also supports line continuations in *string*. See the Examples section for an example.

Certain attributes are also supported for backward compatibility. For more information, see Common Resource Attributes.

Remarks

RC allocates 16 strings per section and uses the identifier value to determine which section is to contain the string. Strings whose identifiers differ only in the bottom 4 bits are placed in the same section.

Examples

The following example demonstrates the use of the **STRINGTABLE** statement to display ASCII strings:

```
```

syntax

```
#define IDS_HELLO    1
#define IDS_GOODBYE  2

STRINGTABLE
{
    IDS_HELLO,   "Hello"
    IDS_GOODBYE, "Goodbye"
}
```

The following example shows how to encode Unicode characters:

syntax

```
STRINGTABLE
BEGIN
IDS_CHINESESTRING L"\x5e2e\x52a9"
IDS_RUSSIANSTRING L"\x0421\x043f\x0440\x0430\x0432\x043a\x0430"
IDS_ARABICSTRING L"\x062a\x0639\x0644\x064a\x0645\x0627\x062a"
END
```

The following example shows strings with both ASCII and Unicode. Note that strings without the initial "L" use the 2-digit escape format:

syntax

```
STRINGTABLE
BEGIN
IDS_1 L"5\x00BC-Inch Floppy Disk"
IDS_1a "5\xBC-Inch Floppy Disk"
IDS_2 L"Don't confuse \x2229 (intersection) with \x222A (union)"
IDS_3 "Copyright \xA92001"
IDS_3a L"Copyright \x00a92001"
END
```

The following example shows how line continuations can be used:

syntax

```
STRINGTABLE
BEGIN
IDS_VERYLONGSTRING "blah blah blah blah blah blah \
blah blah blah blah blah blah \
blah blah blah blah blah blah \
blah blah blah blah blah blah"
END
```

See also

LoadString

ACCELERATORS

CHARACTERISTICS

LANGUAGE

MENU

RCDATA

VERSION

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

STYLE statement

Article • 08/19/2020

Defines the window style of the dialog box. The window style specifies whether the box is a pop-up or a child window.

syntax

```
STYLE style
```

style

The window style. This parameter can be an integer value or a combination of window style values (such as **WS_CAPTION** and **WS_SYSMENU**) and dialog box style values (such as **DS_CENTER**).

For a list of window styles, see Window Styles. For a list of dialog box styles, see Dialog Box Template Styles. If you use the window or dialog box style values, you must include Windows.h.

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

User-Defined Resource

Article • 08/23/2019

A user-defined resource-definition statement defines a resource that contains application-specific data. The data can have any format and can be defined either as the content of a given file (if the *filename* parameter is given) or as a series of numbers and strings (if the *raw-data* block is specified).

syntax

```
nameID typeID filename
```

The *filename* specifies the name of a file containing the binary data of the resource. The contents of the file are included as the resource. RC does not interpret the binary data in any way. It is the programmer's responsibility to ensure that the data is properly aligned for the target computer architecture.

A user-defined resource can also be defined completely in the resource script using the following syntax:

syntax

```
nameID typeID  {  raw-data  }
```

Parameters

nameID

Unique name or a 16-bit unsigned integer that identifies the resource.

typeID

Unique name or a 16-bit unsigned integer that identifies the resource type. If a number is given, it must be greater than 255. The numbers 1 through 255 are reserved for existing and future redefined resource types.

filename

Name of the file that contains the resource data. The parameter must be a valid file name; it must be a full path if the file is not in the current working directory.

raw-data

Raw data consisting of one or more integers or strings of characters. Integers can be specified in decimal, octal, or hexadecimal format. To be compatible with 16-bit Windows, integers are stored as WORD values. You can store an integer as a DWORD value by qualifying the integer with the "L" suffix.

Strings are enclosed in quotation marks. RC does not automatically append a terminating null character to a string. Each string is a sequence of the specified ANSI characters, unless you qualify it as a wide-character string with the "L" prefix.

The block of data begins on a **DWORD** boundary and RC performs no padding or alignment of data within the *raw-data* block. It is the programmer's responsibility to ensure the proper alignment of data within the block.

Example

The following example shows several user-defined statements:

syntax

```
array   MYRES   data.res
14      300     custom.res
18 MYRES2
{
   "Here is an ANSI string\0",    // explicitly null-terminated
   L"Here is a Unicode string\0", // explicitly null-terminated
   1024,                          // integer, stored as WORD
   7L,                            // integer, stored as DWORD
   0x029a,                        // hex integer
   0o733,                         // octal integer
}
```

Feedback

VERSION statement

Article • 10/24/2019

Defines version information about a resource that can be used by tools that read and write resource files. The specified *dword* value appears with the resource in the compiled .RES file. However, the value is not stored in the executable file and has no significance to the system.

The **VERSION** statement appears before the beginning of the body of an ACCELERATORS, DIALOGEX, MENU, RCDATA, or STRINGTABLE resource definition. The specified value applies only to that resource.

syntax

```
VERSION dword
```

dword

A user-defined **DWORD** value.

See also

ACCELERATORS

CHARACTERISTICS

LANGUAGE

MENU

RCDATA

STRINGTABLE

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

VERSIONINFO resource

Article • 01/18/2022

Defines a version-information resource. The resource contains such information about the file as its version number, its intended operating system, and its original filename. The resource is intended to be used with the Version Information functions.

There are two ways to format a **VERSIONINFO** statement:

```
C

versionID VERSIONINFO fixed-info  { block-statement . . . }
```

- or -

```
C

versionID VERSIONINFO
fixed-info
BEGIN
block-statement
. . .
END
```

Parameters

versionID

Version-information resource identifier. This value must be 1.

fixed-info

Version information, such as the file version and the intended operating system. This parameter consists of the following statements.

Statement	Description
FILEVERSION *version*	Binary version number for the file. The *version* consists of two 32-bit integers, defined by four 16-bit integers. For example, "FILEVERSION 3,10,0,61" is translated into two doublewords: 0x0003000a and 0x0000003d, in that order. Therefore, if *version* is defined by the **DWORD** values *dw1* and *dw2*, they need to appear in the **FILEVERSION** statement as follows: `HIWORD(dw1)`, `LOWORD(dw1)`, `HIWORD(dw2)`, `LOWORD(dw2)`.

Statement	Description
PRODUCTVERSION *version*	Binary version number for the product with which the file is distributed. The *version* parameter is two 32-bit integers, defined by four 16-bit integers. For more information about *version*, see the **FILEVERSION** description.
FILEFLAGSMASK *fileflagsmask*	Indicates which bits in the **FILEFLAGS** statement are valid. For 16-bit Windows, this value is 0x3f.
FILEFLAGS *fileflags*	Attributes of the file.
FILEOS *fileos*	Operating system for which this file was designed. The *fileos* parameter can be one of the operating system values given in the Remarks section.
FILETYPE *filetype*	General type of file. The *filetype* parameter can be one of the file type values listed in the Remarks section.
FILESUBTYPE *subtype*	Function of the file. The *subtype* parameter is zero unless the *filetype* parameter in the **FILETYPE** statement is VFT_DRV, VFT_FONT, or VFT_VXD. For a list of file subtype values, see the Remarks section.

block-statement

Specifies one or more version-information blocks. A block can contain string information or variable information. For more information, see StringFileInfo Block or VarFileInfo Block.

Remarks

To use the constants specified with the **VERSIONINFO** statement, you must include the Winver.h or Windows.h header file in the resource-definition file.

The following list describes the parameters used in the **VERSIONINFO** statement:

fileflags

A combination of the following values.

Value	Description
VS_FF_DEBUG	File contains debugging information or is compiled with debugging features enabled.
VS_FF_PATCHED	File has been modified and is not identical to the original shipping file of the same version number.

Value	Description
VS_FF_PRERELEASE	File is a development version, not a commercially released product.
VS_FF_PRIVATEBUILD	File was not built using standard release procedures. If this value is given, the StringFileInfo block must contain a **PrivateBuild** string.
VS_FF_SPECIALBUILD	File was built by the original company using standard release procedures but is a variation of the standard file of the same version number. If this value is given, the StringFileInfo block block must contain a **SpecialBuild** string.
VS_FFI_FILEFLAGSMASK	A combination of all the preceding values.

fileos

One of the following values.

Value	Description
VOS_UNKNOWN	The operating system for which the file was designed is unknown.
VOS_DOS	File was designed for MS-DOS.
VOS_NT	File was designed for 32-bit Windows.
VOS__WINDOWS16	File was designed for 16-bit Windows.
VOS__WINDOWS32	File was designed for 32-bit Windows.
VOS_DOS_WINDOWS16	File was designed for 16-bit Windows running with MS-DOS.
VOS_DOS_WINDOWS32	File was designed for 32-bit Windows running with MS-DOS.
VOS_NT_WINDOWS32	File was designed for 32-bit Windows.

The values 0x00002L, 0x00003L, 0x20000L and 0x30000L are reserved.

filetype

One of the following values.

Value	Description
VFT_UNKNOWN	File type is unknown.

Value	Description
VFT_APP	File contains an application.
VFT_DLL	File contains a dynamic-link library (DLL).
VFT_DRV	File contains a device driver. If *filetype* is **VFT_DRV**, *subtype* contains a more specific description of the driver.
VFT_FONT	File contains a font. If *filetype* is VFT_FONT, *subtype* contains a more specific description of the font.
VFT_VXD	File contains a virtual device.
VFT_STATIC_LIB	File contains a static-link library.

All other values are reserved for use by Microsoft.

subtype

Additional information about the file type.

If *filetype* specifies **VFT_DRV**, this parameter can be one of the following values.

Value	Description
VFT2_UNKNOWN	Driver type is unknown.
VFT2_DRV_COMM	File contains a communications driver.
VFT2_DRV_PRINTER	File contains a printer driver.
VFT2_DRV_KEYBOARD	File contains a keyboard driver.
VFT2_DRV_LANGUAGE	File contains a language driver.
VFT2_DRV_DISPLAY	File contains a display driver.
VFT2_DRV_MOUSE	File contains a mouse driver.
VFT2_DRV_NETWORK	File contains a network driver.
VFT2_DRV_SYSTEM	File contains a system driver.
VFT2_DRV_INSTALLABLE	File contains an installable driver.
VFT2_DRV_SOUND	File contains a sound driver.
VFT2_DRV_VERSIONED_PRINTER	File contains a versioned printer driver.

If *filetype* specifies **VFT_FONT**, this parameter can be one of the following values.

Value	Description
VFT2_UNKNOWN	Font type is unknown.
VFT2_FONT_RASTER	File contains a raster font.
VFT2_FONT_VECTOR	File contains a vector font.
VFT2_FONT_TRUETYPE	File contains a TrueType font.

If *filetype* specifies VFT_VXD, this parameter must be the virtual-device identifier included in the virtual-device control block.

All *subtype* values not listed here are reserved for use by Microsoft.

langID

One of the following language codes.

Code	Language	Code	Language
0x0401	Arabic	0x0415	Polish
0x0402	Bulgarian	0x0416	Portuguese (Brazil)
0x0403	Catalan	0x0417	Rhaeto-Romanic
0x0404	Traditional Chinese	0x0418	Romanian
0x0405	Czech	0x0419	Russian
0x0406	Danish	0x041A	Croato-Serbian (Latin)
0x0407	German	0x041B	Slovak
0x0408	Greek	0x041C	Albanian
0x0409	U.S. English	0x041D	Swedish
0x040A	Castilian Spanish	0x041E	Thai
0x040B	Finnish	0x041F	Turkish
0x040C	French	0x0420	Urdu

Code	Language	Code	Language
0x040D	Hebrew	0x0421	Bahasa
0x040E	Hungarian	0x0804	Simplified Chinese
0x040F	Icelandic	0x0807	Swiss German
0x0410	Italian	0x0809	U.K. English
0x0411	Japanese	0x080A	Spanish (Mexico)
0x0412	Korean	0x080C	Belgian French
0x0413	Dutch	0x0C0C	Canadian French
0x0414	Norwegian ? Bokmal	0x100C	Swiss French
0x0810	Swiss Italian	0x0816	Portuguese (Portugal)
0x0813	Belgian Dutch	0x081A	Serbo-Croatian (Cyrillic)
0x0814	Norwegian ? Nynorsk		

charsetID

One of the following character-set identifiers.

Decimal	Hexadecimal	Character Set
0	0000	7-bit ASCII
932	03A4	Japan (Shift ? JIS X-0208)
949	03B5	Korea (Shift ? KSC 5601)
950	03B6	Taiwan (Big5)
1200	04B0	Unicode
1250	04E2	Latin-2 (Eastern European)
1251	04E3	Cyrillic
1252	04E4	Multilingual
1253	04E5	Greek
1254	04E6	Turkish

Decimal	Hexadecimal	Character Set
1255	04E7	Hebrew
1256	04E8	Arabic

string-name

One of the following predefined names.

Name	Description
Comments	Additional information that should be displayed for diagnostic purposes.
CompanyName	Company that produced the file—for example, `Microsoft Corporation` or `Standard Microsystems Corporation, Inc.` This string is required.
FileDescription	File description to be presented to users. This string may be displayed in a list box when the user is choosing files to install—for example, `Keyboard Driver for AT-Style Keyboards`. This string is required.
FileVersion	Version number of the file—for example, `3.10` or `5.00.RC2`. This string is required.
InternalName	Internal name of the file, if one exists—for example, a module name if the file is a dynamic-link library. If the file has no internal name, this string should be the original filename, without extension. This string is required.
LegalCopyright	Copyright notices that apply to the file. This should include the full text of all notices, legal symbols, copyright dates, and so on. This string is optional.
LegalTrademarks	Trademarks and registered trademarks that apply to the file. This should include the full text of all notices, legal symbols, trademark numbers, and so on. This string is optional.
OriginalFilename	Original name of the file, not including a path. This information enables an application to determine whether a file has been renamed by a user. The format of the name depends on the file system for which the file was created. This string is required.
PrivateBuild	Information about a private version of the file—for example, `Built by TESTER1 on \\TESTBED`. This string should be present only if **VS_FF_PRIVATEBUILD** is specified in the *fileflags* parameter of the root block.
ProductName	Name of the product with which the file is distributed. This string is required.
ProductVersion	Version of the product with which the file is distributed—for example, `3.10` or `5.00.RC2`. This string is required.

Name	Description
SpecialBuild	Text that specifies how this version of the file differs from the standard version—for example, `Private build for TESTER1 solving mouse problems on M250 and M250E computers`. This string should be present only if **VS_FF_SPECIALBUILD** is specified in the *fileflags* parameter of the root block.

Certain attributes are also supported for backward compatibility. For more information, see Common Resource Attributes.

Examples

The following example defines a **VERSIONINFO** resource:

C

```
#define VER_FILEVERSION             3,10,349,0
#define VER_FILEVERSION_STR         "3.10.349.0\0"

#define VER_PRODUCTVERSION          3,10,0,0
#define VER_PRODUCTVERSION_STR      "3.10\0"

#ifndef DEBUG
#define VER_DEBUG                   0
#else
#define VER_DEBUG                   VS_FF_DEBUG
#endif

VS_VERSION_INFO VERSIONINFO
FILEVERSION     VER_FILEVERSION
PRODUCTVERSION  VER_PRODUCTVERSION
FILEFLAGSMASK   VS_FFI_FILEFLAGSMASK
FILEFLAGS       (VER_PRIVATEBUILD|VER_PRERELEASE|VER_DEBUG)
FILEOS          VOS__WINDOWS32
FILETYPE        VFT_DLL
FILESUBTYPE     VFT2_UNKNOWN
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904E4"
        BEGIN
            VALUE "CompanyName",      VER_COMPANYNAME_STR
            VALUE "FileDescription",  VER_FILEDESCRIPTION_STR
            VALUE "FileVersion",      VER_FILEVERSION_STR
            VALUE "InternalName",     VER_INTERNALNAME_STR
            VALUE "LegalCopyright",   VER_LEGALCOPYRIGHT_STR
            VALUE "LegalTrademarks1", VER_LEGALTRADEMARKS1_STR
            VALUE "LegalTrademarks2", VER_LEGALTRADEMARKS2_STR
```

```
            VALUE "OriginalFilename", VER_ORIGINALFILENAME_STR
            VALUE "ProductName",      VER_PRODUCTNAME_STR
            VALUE "ProductVersion",   VER_PRODUCTVERSION_STR
        END
    END

    BLOCK "VarFileInfo"
    BEGIN
        /* The following line should only be modified for localized
versions.     */
        /* It consists of any number of WORD,WORD pairs, with each pair
*/
        /* describing a language,codepage combination supported by the file.
*/
        /*
*/
        /* For example, a file might have values "0x409,1252" indicating
that it  */
        /* supports English language (0x409) in the Windows ANSI codepage
(1252). */

        VALUE "Translation", 0x409, 1252

    END
END
```

See also

Using Version Information

Version Information

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

StringFileInfo BLOCK statement

Article • 10/24/2019

Defines a string information block.

syntax

```
BLOCK "StringFileInfo" { BLOCK "lang-charset" {VALUE "string-name", "value"
. . . }}
```

Parameters

lang-charset

Language and character-set identifier pair. It is a hexadecimal string consisting of the concatenation of the language and character-set identifiers specified in the Remarks section.

string-name

Name of a value in the block and can be one of the predefined names specified in the Remarks section.

value

Character string that specifies the value of the corresponding string name. More than one **VALUE** statement can be given.

Remarks

The *lang-charset* parameter specifies one of the following language codes.

Code	Language	Code	Language
0x0401	Arabic	0x0415	Polish
0x0402	Bulgarian	0x0416	Portuguese (Brazil)
0x0403	Catalan	0x0417	Rhaeto-Romanic
0x0404	Traditional Chinese	0x0418	Romanian
0x0405	Czech	0x0419	Russian

Code	Language	Code	Language
0x0406	Danish	0x041A	Croato-Serbian (Latin)
0x0407	German	0x041B	Slovak
0x0408	Greek	0x041C	Albanian
0x0409	U.S. English	0x041D	Swedish
0x040A	Castilian Spanish	0x041E	Thai
0x040B	Finnish	0x041F	Turkish
0x040C	French	0x0420	Urdu
0x040D	Hebrew	0x0421	Bahasa
0x040E	Hungarian	0x0804	Simplified Chinese
0x040F	Icelandic	0x0807	Swiss German
0x0410	Italian	0x0809	U.K. English
0x0411	Japanese	0x080A	Spanish (Mexico)
0x0412	Korean	0x080C	Belgian French
0x0413	Dutch	0x0C0C	Canadian French
0x0414	Norwegian – Bokmal	0x100C	Swiss French
0x0810	Swiss Italian	0x0816	Portuguese (Portugal)
0x0813	Belgian Dutch	0x081A	Serbo-Croatian (Cyrillic)
0x0814	Norwegian – Nynorsk	?	?

The *lang-charset* parameter also specifies one of the following character-set identifiers.

Identifier	Character Set
0	7-bit ASCII
932	Japan (Shift – JIS X-0208)
949	Korea (Shift – KSC 5601)
950	Taiwan (Big5)

Identifier	Character Set
1200	Unicode
1250	Latin-2 (Eastern European)
1251	Cyrillic
1252	Multilingual
1253	Greek
1254	Turkish
1255	Hebrew
1256	Arabic

The *string-name* parameter specifies one of the following predefined names.

Name	Description
Comments	Additional information that should be displayed for diagnostic purposes.
CompanyName	Company that produced the file—for example, "Microsoft Corporation" or "Standard Microsystems Corporation, Inc." This string is required.
FileDescription	File description to be presented to users. This string may be displayed in a list box when the user is choosing files to install—for example, "Keyboard Driver for AT-Style Keyboards". This string is required.
FileVersion	Version number of the file—for example, "3.10" or "5.00.RC2". This string is required.
InternalName	Internal name of the file, if one exists—for example, a module name if the file is a dynamic-link library. If the file has no internal name, this string should be the original filename, without extension. This string is required.
LegalCopyright	Copyright notices that apply to the file. This should include the full text of all notices, legal symbols, copyright dates, and so on. This string is optional.
LegalTrademarks	Trademarks and registered trademarks that apply to the file. This should include the full text of all notices, legal symbols, trademark numbers, and so on. This string is optional.
OriginalFilename	Original name of the file, not including a path. This information enables an application to determine whether a file has been renamed by a user. The format of the name depends on the file system for which the file was created. This string is required.

Name	Description
PrivateBuild	Information about a private version of the file—for example, "Built by TESTER1 on \TESTBED". This string should be present only if **VS_FF_PRIVATEBUILD** is specified in the *fileflags* parameter of the root block.
ProductName	Name of the product with which the file is distributed. This string is required.
ProductVersion	Version of the product with which the file is distributed—for example, "3.10" or "5.00.RC2". This string is required.
SpecialBuild	Text that specifies how this version of the file differs from the standard version —for example, "Private build for TESTER1 solving mouse problems on M250 and M250E computers". This string should be present only if **VS_FF_SPECIALBUILD** is specified in the *fileflags* parameter of the root block.

VarFileInfo BLOCK statement

Article • 10/24/2019

Defines a variable information block.

syntax

```
BLOCK "VarFileInfo" { VALUE "Translation", langID, charsetID . . . }
```

Parameters

langID

One of the language identifiers specified in the Remarks section.

charsetID

One of the character-set identifiers specified in the Remarks section.

Remarks

More than one identifier pair can be given, but each pair must be separated from the preceding pair with a comma.

The *langID* parameter specifies one of the following language codes.

Code	Language	Code	Language
0x0401	Arabic	0x0415	Polish
0x0402	Bulgarian	0x0416	Portuguese (Brazil)
0x0403	Catalan	0x0417	Rhaeto-Romanic
0x0404	Traditional Chinese	0x0418	Romanian
0x0405	Czech	0x0419	Russian
0x0406	Danish	0x041A	Croato-Serbian (Latin)
0x0407	German	0x041B	Slovak
0x0408	Greek	0x041C	Albanian
0x0409	U.S. English	0x041D	Swedish

Code	Language	Code	Language
0x040A	Castilian Spanish	0x041E	Thai
0x040B	Finnish	0x041F	Turkish
0x040C	French	0x0420	Urdu
0x040D	Hebrew	0x0421	Bahasa
0x040E	Hungarian	0x0804	Simplified Chinese
0x040F	Icelandic	0x0807	Swiss German
0x0410	Italian	0x0809	U.K. English
0x0411	Japanese	0x080A	Spanish (Mexico)
0x0412	Korean	0x080C	Belgian French
0x0413	Dutch	0x0C0C	Canadian French
0x0414	Norwegian – Bokmal	0x100C	Swiss French
0x0810	Swiss Italian	0x0816	Portuguese (Portugal)
0x0813	Belgian Dutch	0x081A	Serbo-Croatian (Cyrillic)
0x0814	Norwegian – Nynorsk	?	?

The *charsetID* parameter specifies one of the following character-set identifiers:

Identifier	Character Set
0	7-bit ASCII
932	Japan (Shift – JIS X-0208)
949	Korea (Shift – KSC 5601)
950	Taiwan (Big5)
1200	Unicode
1250	Latin-2 (Eastern European)
1251	Cyrillic
1252	Multilingual

Identifier	Character Set
1253	Greek
1254	Turkish
1255	Hebrew
1256	Arabic

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Common Control Parameters

Article • 08/19/2020

The following describes the general syntax for a control resource-definition statement. The meaning of each parameter is given below. Occasionally, a statement will use a parameter differently, or may ignore a parameter. The statement-specific variation is described in the documentation for the statement.

syntax

```
control [[text,]] id, x, y, width, height[[, style[[, extended-style]]]][,
helpId]
[{ data-element-1 [, data-element-2 [,  . . . ]]}]
```

text

Text that is to be displayed with the control. The text is positioned within the control or adjacent to the control.

This parameter must contain zero or more characters enclosed in double quotation marks ("). Strings are automatically null-terminated and converted to Unicode in the resulting resource file.

By default, the characters listed between the double quotation marks are ANSI characters, and escape sequences are interpreted as byte escape sequences. If the string is preceded by the "L" prefix, the string is a wide-character string and escape sequences are interpreted as 2-byte escape sequences that specify Unicode characters. If a double quotation mark is required in the text, you must include the double quotation mark twice.

An ampersand (&) character in the text indicates that the following character is used as a mnemonic character for the control. When the control is displayed, the ampersand is not shown, but the mnemonic character is underlined. The user can choose the control by pressing the key corresponding to the underlined mnemonic character. To use the ampersand as a character in a string, insert two ampersands (&&).

id

Control identifier. This value must be a 16-bit unsigned integer in the range 0 through 65,535 or a simple arithmetic expression that evaluates to a value in that range.

x

X-coordinate of the left side of the control relative to the left side of the dialog box. This value must be a 16-bit unsigned integer in the range 0 through 65,535. The coordinate is in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.

y

Y-coordinate of the top side of the control relative to the top of the dialog box. This value must be a 16-bit unsigned integer in the range 0 through 65,535. The coordinate is in dialog units relative to the origin of the dialog box, window, or control containing the specified control.

width

Width of the control. This value must be a 16-bit unsigned integer in the range 1 through 65,535. The width is in 1/4-character units.

height

Height of the control. This value must be a 16-bit unsigned integer in the range 1 through 65,535. The height is in 1/8-character units.

style

Control styles. Use the bitwise OR (|) operator to combine styles. For more information, see Window Styles.

extended-style

Extended window styles. You must specify *style* to specify *extended-style*. For more information, see **EXSTYLE**.

helpId

Numeric expression indicating the ID used to identify the control during **WM_HELP** processing.

controlData

Control-specific data for the control. When a dialog is created, and a control in that dialog which has control-specific data is created, a pointer to that data is passed into the control's window procedure through the *lParam* of the **WM_CREATE** message for that control.

Remarks

Horizontal dialog units are 1/4 of the dialog base width unit. Vertical units are 1/8 of the dialog base height unit. The current dialog base units are computed from the height and width of the current system font. The **GetDialogBaseUnits** function returns the dialog base units in pixels. The coordinates are relative to the origin of the dialog box.

Feedback

Common Resource Attributes

Article • 08/23/2019

The resource-definition statements supported on 16-bit Windows include a *load-mem* option that specifies the loading and memory characteristics of the resource. These attributes are permitted in resource scripts for backward compatibility, but they are ignored. Windows resources are loaded when the corresponding module is loaded, and are freed when the module is unloaded.

Load Attributes

The load attributes specify when the resource is to be loaded. The load parameter must be one of the following attributes.

Attribute	Description
PRELOAD	Ignored. In 16-bit Windows, the resource is loaded with the executable file.
LOADONCALL	Ignored. In 16-bit Windows, the resource is loaded when called.

Memory Attributes

The memory attributes specify whether the resource is fixed or movable, whether it is discardable, and whether it is pure. The memory parameter can be one or more of the following attributes.

Attribute	Description
FIXED	Ignored. In 16-bit Windows, the resource remains at a fixed memory location.
MOVEABLE	Ignored. In 16-bit Windows, the resource can be moved if necessary to compact memory.
DISCARDABLE	Ignored. In 16-bit Windows, the resource can be discarded if no longer needed.
PURE	Ignored. Accepted for compatibility with existing resource scripts.
IMPURE	Ignored. Accepted for compatibility with existing resource scripts.
SHARED	Ignored. In 16-bit Windows, SHARED is ignored for regular modules. For a resource from a ROM Windows module, the memory is shared.

Attribute	Description
NONSHARED	Ignored. In 16-bit Windows, NONSHARED is ignored for regular modules. For a resource from a ROM Windows module, the memory is not shared.

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Sample Resource-Definition File

Article • 08/23/2019

The following example shows a script file that defines the resources for an application named Shapes:

syntax

```
#include "shapes.h"

ShapesCursor  CURSOR  SHAPES.CUR
ShapesIcon    ICON    SHAPES.ICO

ShapesMenu MENU
{
    POPUP "&Shape"
    {
        MENUITEM "&Clear", ID_CLEAR
        MENUITEM "&Rectangle", ID_RECT
        MENUITEM "&Triangle", ID_TRIANGLE
        MENUITEM "&Star", ID_STAR
        MENUITEM "&Ellipse", ID_ELLIPSE
    }
}
```

The **CURSOR** statement names the application's cursor resource ShapesCursor and specifies the cursor file SHAPES.CUR, which contains the image for that cursor.

The **ICON** statement names the application's icon resource ShapesIcon and specifies the icon file SHAPES.ICO, which contains the image for that icon.

The **MENU** statement defines an application menu named **ShapesMenu**, a pop-up menu with five menu items.

The body of the menu definition, enclosed by the curly braces, or the **BEGIN** and **END** keywords, specifies each menu item and the menu identifier that is returned when the user selects that item. For example, the first item on the menu, **Clear**, returns the menu identifier ID_CLEAR when the user selects it. The menu identifiers are defined in the application header file, SHAPES.H.

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Using RC (The RC Command Line)

Article • 08/19/2020

To start RC, use the following command.

RC [*options*] *script-file*

The *script-file* parameter specifies the name of the resource-definition file that contains the names, types, filenames, and descriptions of the resources to be compiled.

RC can generate separate resource files for applications that have both language-neutral and language-specific resources. Developers can use a resource configuration file or set command-line options to select which resource types and items are non-localizable resources of the language-neutral (LN) file and which are localizable resources of language-specific MUI files. For more information, see the Multilingual User Interface.

The *options* parameter can be one or more of the following command-line options.

Options

/?

Displays a list of command-line options.

/c

Defines a code page used by NLS conversion.

/d

Defines a symbol for the preprocessor that you can test with the **#ifdef** directive.

/fm *mresname*

RC creates one language-neutral .RES file and one language-dependent (MUI) .RES file using *script-file*. This option must be used together with the **/fo** *resname* option. RC names the language-neutral .RES file *resname.res* and names the language-dependent (MUI) .RES file *mresname.res*.

Windows Server 2003 and Windows XP/2000: This option is not available without also using the **LoadMUILibrary** and **FreeMUILibrary** functions on an updated system.

/fo *resname*

RC creates a .RES file named *resname* using *script-file*.

If the **/fm** *mresname* option is also set, RC creates one language-neutral .RES file and one language-dependent (MUI) .RES file.

Windows Server 2003 and Windows XP/2000: This option is not available without also using the **LoadMUILibrary** and **FreeMUILibrary** functions on an updated system.

/g1

If /g1, is set, RC generates a MUI file if the only localizable resource being included in the MUI file is a version resource. If /g1 is not set, RC will not generate a MUI file if the only localizable resource being included in the MUI file is a version resource.

/h

Displays the list of command-line options.

/I

Searches the specified directory before searching the directories specified by the INCLUDE environment variable.

/j *loctype*

Localizable resource types RC places into the language-dependent (MUI) .RES file. If the **/q** option is also set, this option is ignored, and the information in the RC Configuration file takes precedence.

Windows Server 2003 and Windows XP/2000: This option is not available without also using the **LoadMUILibrary** and **FreeMUILibrary** functions on an updated system.

/k *overtype*

Overlapping resource types that RC places into both the language-neutral .RES and the language-dependent (MUI).RES files. The resource types that are specified by the **/k** option must be a subset of those that are specified by the **/j** option. For example, ?J2 ?J3 ?K3 specifies that RC places resource type 3 in both the language-neutral and language-dependent (MUI) files. If the **/q** option is also set, this option is ignored, and the information in the RC Configuration file takes precedence.

Windows Server 2003 and Windows XP/2000: This option is not available without also using the **LoadMUILibrary** and **FreeMUILibrary** functions on an updated system.

/l *langid*

Specifies the default language for compilation. For example, -l409 is equivalent to including the following statement at the top of the resource script file: `LANGUAGE LANG_ENGLISH,SUBLANG_ENGLISH_US`

For more information, see Language Identifiers.

/n

Null terminates all strings in the string table.

/q *Mui.RCConfig*

An RC configuration file that follows the RC Configuration File format. The RC Configuration File format enables components to self-describe resource information such as resource versioning, MUI file path, resource types and items. This file specifies which resources go into the language-neutral .RES file and which resources go into the language-dependent (MUI) .RES file. This option, and the information provided in the RC Configuration file, override the command line options **/j** and **/k**.

Windows Server 2003 and Windows XP/2000: This option is not available without also using the **LoadMUILibrary** and **FreeMUILibrary** functions on an updated system.

/r

Ignored. Provided for compatibility with existing makefiles.

/u

Undefines a symbol for the preprocessor.

/v

Displays messages that report on the progress of the compiler.

/x

Prevents RC from checking the INCLUDE environment variable when searching for header files or resource files.

Remarks

Options are not case sensitive, and a hyphen (-) can be used in place of a slash mark (/). You can combine single-letter options if they do not require any additional parameters.

RC will not generate a MUI file in the following cases.

- No localizable resources exist in the .rc file.
- The only resource language id specified in the .rc file is neutral (0x0).
- The .rc file has resources that are specified in more than one language. The exception is if the .rc file contains two languages, and one language is neutral (0x0), RC generates a MUI file.

For more information, see the following topics:

- Defining Names for the Preprocessor
- Renaming the Compiled Resource File
- Searching for Files
- Displaying Progress Messages
- RC Diagnostic Messages

Related topics

Multilingual User Interface

Feedback

Defining Names for the Preprocessor

Article • 08/23/2019

You can specify conditional compilation in a script, based on whether a name is defined on the RC command line with the **/d** option, or in the file or an include file with the **#define** directive.

For example, suppose your application has a pop-up menu that should appear only with debugging versions of the application. When you compile the application for normal use, the menu is not included. The following example shows the statements that can be added to the resource-definition file to define a Debug menu:

syntax

```
#include <windows.h>

MainMenu MENU
{
    //. . .
#ifdef DEBUG
    POPUP "&Debug"
    {
        MENUITEM "&Memory usage", ID_MEMORY
        MENUITEM "&Walk data heap", ID_WALK_HEAP
    }
#endif
}
```

When compiling resources for a debugging version of the application, you could include the Debug menu by using the following command:

rc -d DEBUG myapp.rc

To compile resources for a normal version of the application?one that does not include the Debug menu?you could use the following command:

rc myapp.rc

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Renaming the Compiled Resource File

Article • 08/23/2019

By default, when compiling resources, RC names the compiled resource (.res) file with the base name of the .rc file and places it in the same directory as the .rc file. CVTRES must then be invoked to convert the .res file to a binary resource (.rbj) format which can be understood by the linker. The following example compiles MyApp.rc and creates a compiled resource file named MyApp.res in the same directory as MyApp.rc:

rc myapp.rc

The **/fo** option gives the resulting .res file a name that differs from the name of the corresponding .rc file. For example, to name the resulting .res file NewFile.res, use the following command:

rc -fo newfile.res myapp.rc

The **/fo** option can also place the .res file in a different directory. For example, the following command places the compiled resource file MyApp.res in the directory C:\Source\Resource:

rc -fo c:\source\resource\myapp.res myapp.rc

Feedback

Searching for Files

Article • 08/23/2019

By default, RC searches for header files and resource files (such as icon and cursor files) first in the current directory and then in the directories specified by the INCLUDE environment variable. (The PATH environment variable has no effect on which directories RC searches.)

Adding a Directory to Search

You can use the **/i** option to add a directory to the list of directories RC searches. The compiler then searches the directories in the following order:

1. The current directory
2. The directory or directories you specify by using the **/i** option, in the order in which they appear on the RC command line
3. The list of directories specified by the INCLUDE environment variable, in the order in which the variable lists them, unless you specify the **/x** option

The following example compiles the resource-definition file MyApp.rc:

rc /i c:\source\stuff /i d:\resources myapp.rc

When compiling the script MyApp.rc, RC searches for header files and resource files first in the current directory, then in C:\Source\Stuff and D:\Resources, and then in the directories specified by the INCLUDE environment variable.

Ignoring the INCLUDE Environment Variable

You can prevent RC from using the INCLUDE environment variable when determining the directories to search. To do so, use the **/x** option. The compiler then searches for files only in the current directory and in any directories you specify by using the **/i** option.

The following command compiles the script file MyApp.rc:

rc /x /i c:\source\stuff myapp.rc

When compiling the script MyApp.rc, RC searches for header files and resource files first in the current directory and then in C:\Source\Stuff. It does not search the directories specified by the INCLUDE environment variable.

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Displaying Progress Messages

Article • 08/23/2019

By default, RC compiles without displaying progress messages. However, you can specify that RC is to display progress messages. To do so, use the **/v** option.

The following example causes RC to report its progress as it compiles the resource-definition file Sample.rc and creates the compiled resource file Sample.res:

rc /v sample.rc

Feedback

RC Diagnostic Messages

Article • 08/23/2019

This section describes of diagnostic messages produced by RC. Many of these messages appear when RC is not able to compile resources properly. The descriptions clarify the cause of each message.

A capital V in parentheses (V) at the beginning of a message description indicates that the message is displayed only if RC is run with the **/v** (verbose) option. These messages are generally informative and do not necessarily indicate errors.

The messages are listed in alphabetic order in the following topics:

Feedback

A (Menus and Other Resources)

Article • 12/10/2020

Accelerator Type required (ASCII or VIRTKEY)

The *type* parameter in the **ACCELERATORS** statement must contain one of the following values: ASCII or VIRTKEY.

Feedback

B (Menus and Other Resources)

Article • 12/10/2020

BEGIN expected in Accelerator Table

An **ACCELERATORS** statement was not followed by the **BEGIN** keyword.

BEGIN expected in Dialog

A **DIALOG** statement was not followed by the **BEGIN** keyword.

BEGIN expected in menu

A **MENU** statement was not followed by the **BEGIN** keyword.

BEGIN expected in RCData

An **RCDATA** statement was not followed by the **BEGIN** keyword.

BEGIN expected in String Table

A **STRINGTABLE** statement was not followed by the **BEGIN** keyword.

BEGIN expected in VERSIONINFO resource

A **VERSIONINFO** statement was not followed by the **BEGIN** keyword.

Feedback

C (Menus and Other Resources)

Article • 12/10/2020

Cannot Re-use String Constants

You are using the same value twice in a **STRINGTABLE** statement. Make sure that you have not mixed overlapping decimal and hexadecimal values.

Control Character out of range [A?Z]

A control character in the **ACCELERATORS** statement is invalid. The character following the caret (^) must be in the range A through Z.

Could not openin-file-name

Microsoft Windows Resource Compiler (RC) could not open the specified file. Make sure that the file exists and that you typed the filename correctly.

Couldn't openresource-name

Microsoft Windows Resource Compiler (RC) could not open the specified file. Make sure that the file exists and that you typed the filename correctly.

Creatingresource-name

(V) Microsoft Windows Resource Compiler (RC) is creating a new binary resource (.res) file.

Feedback

E (Menus and Other Resources)

Article • 12/10/2020

Empty menus not allowed

An **END** keyword appears before any menu items are defined in the **MENU** statement. Empty menus are not permitted by Microsoft Windows 32 Resource Compiler (RC). Make sure that you do not have any opening quotation marks within the **MENU** statement.

END expected in Dialog

The **END** keyword must appear at the end of a **DIALOG** statement. Make sure that there are no opening quotation marks left from the preceding statement.

END expected in menu

The **END** keyword must appear at the end of a **MENU** statement. Make sure that there are no mismatched **BEGIN** and **END** statements.

Error Creatingresource-name

Microsoft Windows 32 Resource Compiler (RC) could not create the specified binary resource (.RES) file. Make sure that it is not being created on a read-only drive. Use the **/v** option to find out whether the file is being created.

Expected Comma in Accelerator Table

Microsoft Windows 32 Resource Compiler (RC) requires a comma between the *event* and *idvalue* parameters in the **ACCELERATORS** statement.

Expected control class name

The *class* parameter of a **CONTROL** statement in the **DIALOG** statement must be one of the following control types: **BUTTON**, **COMBOBOX**, **EDIT**, **LISTBOX**, **SCROLLBAR**, **STATIC**, or user-defined. Make sure that the class is spelled correctly.

Expected font face name

The *typeface* parameter of the **FONT** statement in the **DIALOG** statement must be a character string enclosed in double quotation marks ("). This parameter specifies the name of a font.

Expected ID value for Menuitem

The MENU statement must contain a MENUITEM statement, which has either an integer or a symbolic constant in the *MenuID* parameter.

Expected Menu String

Each MENUITEM and POPUP statement must contain a *text* parameter. This parameter is a string enclosed in double quotation marks (") that specifies the name of the menu item or pop-up menu. A **MENUITEM SEPARATOR** statement requires no quoted string.

Expected numeric command value

Microsoft Windows Resource Compiler (RC) was expecting a numeric *idvalue* parameter in the ACCELERATORS statement. Make sure that you have used a #define statement to specify the value and that the constant used is spelled correctly.

Expected numeric constant in string table

A numeric constant, defined in a #define statement, must immediately follow the **BEGIN** keyword in a STRINGTABLE statement.

Expected numeric point size

The *pointsize* parameter of the FONT statement in the DIALOG statement must be an integer point-size value.

Expected Numerical Dialog constant

A DIALOG statement requires integer values for the *x*, *y*, *width*, and *height* parameters. Make sure that these values, which are included after the **DIALOG** keyword, are not negative.

Expected String in STRINGTABLE

A string is expected after each numeric *stringid* parameter in a STRINGTABLE statement.

Expected String or Constant Accelerator command

Microsoft Windows Resource Compiler (RC) was not able to determine which key was being set up for the accelerator. The *event* parameter in the ACCELERATORS statement might be invalid.

Expected VALUE, BLOCK, or END keyword.

The VERSIONINFO structure requires a **VALUE**, **BLOCK**, or **END** keyword.

Expecting number for ID

A number is expected for the *id* parameter of a **CONTROL** statement in the **DIALOG** statement. Make sure that you have a number or a **#define** statement for the control identifier.

Expecting quoted string for key

The key string following the **BLOCK** or **VALUE** keyword should be enclosed in double quotation marks.

Expecting quoted string in dialog class

The *class* parameter of the **CLASS** statement in the **DIALOG** statement must be an integer or a string enclosed in double quotation marks (").

Expecting quoted string in dialog title

The *captiontext* parameter of the **CAPTION** statement in the **DIALOG** statement must be a character string, enclosed in double quotation marks (").

Feedback

F (Menus and Other Resources)

Article • 12/10/2020

File not found:filename

The file specified in the RC command was not found. Make sure that the file has not been moved to another directory and that the filename or path is typed correctly.

Font names must be ordinals

The *pointsize* parameter in the **FONT** statement must be an integer, not a string.

Feedback

I (Menus and Other Resources)

Article • 12/10/2020

Invalid Accelerator

An *event* parameter in the **ACCELERATORS** statement was not recognized or was more than two characters long.

Invalid Accelerator Type (ASCII or VIRTKEY)

The *type* parameter in the **ACCELERATORS** statement must contain either the **ASCII** or **VIRTKEY** value.

Invalid control character

A control character in the **ACCELERATORS** statement is invalid. A valid control character consists of a caret (^) followed by a single letter.

Invalid Control type

The control statement in a **DIALOG** statement must be one of the following: **AUTO3STATE**, **AUTOCHECKBOX**, **AUTORADIOBUTTON**, **CHECKBOX**, **COMBOBOX**, **CONTROL**, **CTEXT**, **DEFPUSHBUTTON**, **EDITTEXT**, **GROUPBOX**, **ICON**, **LISTBOX**, **LTEXT**, **PUSHBOX**, **PUSHBUTTON**, **RADIOBUTTON**, **RTEXT**, **SCROLLBAR**, **STATE3**.

Invalid directive in preprocessed RC file

The specified filename has an embedded newline character.

Invalid .EXE file

The executable (.exe) file is invalid. Make sure that the linker created it correctly and that the file exists.

Invalid switch,option

An option used was invalid. For a list of the command-line options, use the **rc /?** command.

Invalid switch, too many -D switches

Too many **-d** options were specified. To define a large number of symbols, use the **#include** directive to include a file of definitions.

Invalid type

The resource type was not among the types defined in the include file.

Invalid usage. Use RC -? for Help

Make sure that you have at least one filename to work with. For a list of the command-line options, use the **rc -?** command.

I/O error reading file

Read failed. Since this is a generic routine, no specific filename is supplied.

I/O error seeking in file

Seeking in file failed. Since this is a generic routine, no specific filename is supplied.

I/O error writing file

Write failed. Since this is a generic routine, no specific filename is supplied.

Feedback

N (Menus and Other Resources)

Article • 12/10/2020

No resource binary filename specified.

The **/fo** option was used, but no binary resource (.res) file was specified.

Feedback

Was this page helpful? Yes No

O (Menus and Other Resources)

Article • 12/10/2020

Old DIB inresource-name. Pass it through IMAGEDIT.

The resource file specified is not compatible.

Out of heap memory

There was not enough memory.

Out of memory, neededbytes

Microsoft Windows Resource Compiler (RC) was not able to allocate the specified amount of memory.

Feedback

R (Menus and Other Resources)

Article • 12/10/2020

Redefinition of Button Type

The specified button styles conflict with or modify the style specified by the control keyword. This is a warning only and may be ignored.

The following common method of declaring an automatic radio button generates this warning:

Feedback

T (Menus and Other Resources)

Article • 12/10/2020

Text string or ordinal expected in Control

The *text* parameter of a **CONTROL** statement in the **DIALOG** statement must be either a text string or an ordinal reference to the type of control that is expected. If using an ordinal, make sure that you have a **#define** statement for the control.

Feedback

U (Menus and Other Resources)

Article • 11/19/2022

Unable to createdestination

Microsoft Windows Resource Compiler (RC) was not able to create the destination file. Make sure that there is enough disk space.

Unbalanced Parentheses

Make sure that you have closed every opening parenthesis in the **DIALOG** statement.

Unexpected value in RCData

The values for the *raw-data* parameter in the **RCDATA** statement must be integers or strings, separated by commas. Make sure that you did not leave out a comma or a quotation mark around a string.

Unexpected value in value data

A statement contained information with a format or size different from the expected value for that parameter.

Unknown DIB header format

The device-independent bitmap (DIB) header is not a **BITMAPCOREHEADER** or **BITMAPINFOHEADER** structure.

Unknown Menu SubType

The *item-definitions* parameter of the **MENU** statement can contain only **MENUITEM** and **POPUP** statements.

Unrecognized VERSIONINFO field; BEGIN or comma expected

The format of the information following a **VERSIONINFO** statement is incorrect.

Feedback

Was this page helpful? Yes No

V (Menus and Other Resources)

Article • 12/10/2020

Version WORDs separated by commas expected

Values in an information block for a **VERSIONINFO** statement should be separated by commas.

Feedback

W (Menus and Other Resources)

Article • 12/10/2020

Warning: ASCII character not equivalent to virtual key code

An invalid virtual-key code exists in the **ACCELERATORS** statement. The ASCII values for some characters such as *, ^, or & are not equivalent to the virtual-key codes for the corresponding keys. In the case of the asterisk (*), the virtual-key code is equivalent to the ASCII value for 8, the numeric character on the same key.

Warning: SHIFT or CONTROL used without VIRTKEY

The **ALT**, **SHIFT**, and **CONTROL** options apply only to virtual keys in the **ACCELERATORS** statement. Make sure that the **VIRTKEY** option is used with one of these other options.

Writing resourcetypeimagelang:languagesize:size

(V) Microsoft Windows Resource Compiler (RC) is writing the specified resource. The *type* is the resource type. It may be a number or a string. The *image* can be a number or a string. The *language* is the national language of the resource. The *size* is the decimal size of the resource in bytes.

Feedback

Package resource indexing (PRI) reference

Article • 09/17/2024

A set of APIs for creating PRI files via a resource indexer. PRI files are used by both packaged and unpackaged apps to locate resources (such as strings or image files) at runtime.

These APIs are used to create build tools that perform similar functions to the Windows SDK **MakePri** tool and to Visual Studio MSIX projects. They are not intended to be used by applications themselves; at runtime apps should use either **Windows.ApplicationModel.Resources** (for UWP apps) or **Microsoft.Windows.ApplicationModel.Resources** (for WinApp SDK apps) to resolve resources.

For more info, and scenario-based walkthroughs of how to use these APIs, see Package resource indexing (PRI) APIs and custom build systems.

In this section

Conceptual

- Qualifiers in MRM
- Resource names in MRM
- File resources in MRM

Functions

- **MrmCreateConfig** function
- **MrmCreateConfigInMemory** function
- **MrmCreateResourceFile** function
- **MrmCreateResourceFileInMemory** function
- **MrmCreateResourceIndexer** function
- **MrmCreateResourceIndexerFromPreviousPriData** function
- **MrmCreateResourceIndexerFromPreviousPriFile** function
- **MrmCreateResourceIndexerFromPreviousSchemaData** function
- **MrmCreateResourceIndexerFromPreviousSchemaFile** function
- **MrmDestroyIndexerAndMessages** function
- **MrmDumpPriFile** function
- **MrmDumpPriFileInMemory** function

- MrmDumpPriDataInMemory function
- MrmFreeMemory function
- MrmIndexEmbeddedData function
- MrmIndexFile function
- MrmIndexFileAutoQualifiers function
- MrmIndexResourceContainerAutoQualifiers function
- MrmIndexString function
- MrmPeekResourceIndexerMessages function

Structures

- MrmResourceIndexerHandle structure
- MrmResourceIndexerMessage structure

Enumerations

- MrmDumpType enumeration
- MrmPackagingMode enumeration
- MrmPackagingOptions enumeration
- MrmPlatformVersion enumeration
- MrmResourceIndexerMessageSeverity enumeration

Feedback

MrmCreateConfig function

Article • 09/17/2024

Creates a new, initialized PRI config file for use with the Windows SDK **MakePri** tool. None of the other MRM functions work with config files.

This function performs the equivalent of the `makepri createconfig` command.

COM must be initialized (e.g. by calling **CoInitializeEx**) before using this function.

Syntax

```
C++

HRESULT HRESULT MrmCreateConfig(
  _In_     MrmPlatformVersion platformVersion,
  _In_opt_ PCWSTR             defaultQualifiers,
  _In_     PCWSTR             outputXmlFile
);
```

Parameters

platformVersion [in]

Type: **MrmPlatformVersion**

The platform version (*targetOsVersion*) to use for the generated configuration file. Most callers should just use **MrmPlatformVersion_Windows10_0_0_5**

defaultQualifiers [in, optional]

Type: **PCWSTR**

A list of default resource qualifiers. For example, "language-en-US_scale-100". For more information about qualifiers, see Qualifiers in MRM.

outputXmlFile [in]

Type: **PCWSTR**

The path of the configuration file to create. The file will be overwritten if it already exists. The directory must already exist.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise an error value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

Package resource indexing (PRI) APIs and custom build systems

Feedback

MrmCreateConfigInMemory function

Article • 09/17/2024

Creates a new, initialized PRI config file in memory (i.e. not on disk) for use with the Windows SDK **MakePri** tool. None of the other MRM functions work with config files.

This function performs the equivalent of the `makepri createconfig` command, but in memory.

COM must be initialized (e.g. by calling **CoInitializeEx**) before using this function.

Syntax

```
C++

HRESULT HRESULT MrmCreateConfigInMemory(
  _In_     MrmPlatformVersion platformVersion,
  _In_opt_ PCWSTR             defaultQualifiers,
  _Out_    BYTE               **outputXmlData,
  _Out_    ULONG              *outputXmlSize
);
```

Parameters

platformVersion [in]

Type: **MrmPlatformVersion**

The platform version (*targetOsVersion*) to use for the generated configuration file. Most callers should just use **MrmPlatformVersion_Windows10_0_0_5**

defaultQualifiers [in, optional]

Type: **PCWSTR**

A list of default resource qualifiers. For example, "language-en-US_scale-100". For more information about qualifiers, see Qualifiers in MRM.

outputXmlData [out]

Type: **BYTE****

The address of a **BYTE** pointer. On successful return, contains a pointer to the buffer allocated by the function that contains the generated config file. You must free the memory by calling **MrmFreeMemory** when you are done with it.

outputXmlSize [out]

Type: **ULONG***

The address of a **ULONG**. On successful return, contains the size of the allocated memory buffer pointed to by *outputXmlData*.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise an error value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

Package resource indexing (PRI) APIs and custom build systems

Feedback

Was this page helpful? Yes No

MrmCreateResourceFile function

Article • 09/17/2024

Creates one or more PRI files on disk in the specified directory. This is the last stage of creating a PRI file, after creating the indexer and adding resources.

Syntax

```
C++

HRESULT HRESULT MrmCreateResourceFile(
  _In_ MrmResourceIndexerHandle indexer,
  _In_ MrmPackagingMode         packagingMode,
  _In_ MrmPackagingOptions      packagingOptions,
       PCWSTR                   outputDirectory
);
```

Parameters

indexer [in]

Type: **MrmResourceIndexerHandle**

A handle identifying the resource indexer from which to create the PRI file. This handle is returned via a call to **MrmCreateResourceIndexer** or one of the related **MrmCreateResourceIndexer...*** functions.

packagingMode [in]

Type: **MrmPackagingMode**

Specifies what file(s) the function should create. To create a PRI file for an unpackaged desktop app, you *must* use **MrmPackagingModeStandaloneFile**. See **MrmPackagingMode** for more info.

packagingOptions [in]

Type: **MrmPackagingOptions**

Specifies additional options about the PRI file. Most callers should specify **MrmPackagingOptionsNone**. See the documentation for **MrmPackagingOptions** for more info.

outputDirectory

Type: **PCWSTR**

The directory in which to save the PRI file(s), which may be absolute or relative. The directory must already exist.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise an error value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Some errors you may encounter include:

- **0x80070057: E_INVALIDARG One or more arguments are invalid.** This may indicate an attempt to create a Resource Pack PRI without specifying a main PRI. See MrmPackagingMode for more info.
- **0x80073B0F: HRESULT_FROM_WIN32(ERROR_MRM_DUPLICATE_ENTRY) : Duplicate Entry.** This indicates a resource was added twice with the same name and qualifiers but different values. There is no way to determine which resource caused the error.
- **0x80073B08: HRESULT_FROM_WIN32(ERROR_MRM_INVALID_FILE_TYPE) : Invalid file type.** This can occur when using the packaging mode **MrmPackagingModeResourcePack** and there are candidates that don't match the default qualifiers (e.g. the default language is German but a resource was added for French) *or* if a resource is added that doesn't exist in the main PRI file. There is no way to determine which resource caused the error.
- **0x80073B39: An entity was defined as both resource and scope, which is not allowed**. This indicates that a resource name inside a resource container (e.g. a `resw` file) had embedded separators (slashes or dots) *and* the resource container was in a subdirectory such as `\language-en\`. There is no way to determine which resource caused the error.
- **0x8007000B HRESULT_FROM_WIN32(ERROR_BAD_FORMAT) : An attempt was made to load a program with an incorrect format.** This indicates that a file was added via **MrmIndexFileAutoQualifiers** but it was not a relative path. There is no way to determine which resource caused the error.

Remarks

If this function succeeds, it creates one or more files in the *outputDirectory* depending on the value of the *packagingMode* parameter:

- If **packagingMode** is set to **MrmPackagingModeStandaloneFile**, a single file named **resources.pri** will be created.
- If **packagingMode** is set to **MrmPackagingModeResourcePack**, a single file named **resources.pri** will be created.
- If **packagingMode** is set to **MrmPackagingModeAutoSplit**, one or more files will be created based on the different language(s) and / or scale(s) used by resources added to the indexer.

See **MrmPackagingMode** for more info on the different *packagingMode* options, including limitations.

There is no way to specify the name of the output file(s). For packaged apps, you cannot change the names of the files or else they will not work correctly. For unpackaged apps, you can rename **resources.pri** as long as you pass the correct filename to the **ResourceLoader(String, String)** constructor.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

MrmCreateResourceFileInMemory

Package resource indexing (PRI) APIs and custom build systems

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MrmCreateResourceFileInMemory function

Article • 09/17/2024

Creates PRI file in memory, not as a file on disk. This is the last stage of creating a PRI file, after creating the indexer and adding resources.

This function can only create a single PRI file; if you want to create split PRI files you must use the **MrmCreateResourceFile** function.

Syntax

```
C++

HRESULT HRESULT MrmCreateResourceFileInMemory(
  _In_  MrmResourceIndexerHandle indexer,
  _In_  MrmPackagingMode         packagingMode,
  _In_  MrmPackagingOptions      packagingOptions,
  _Out_ BYTE                     **outputPriData,
  _Out_ ULONG                    *outputPriSize
);
```

Parameters

indexer [in]

Type: **MrmResourceIndexerHandle**

A handle identifying the resource indexer from which to create the PRI file. This handle is returned via a call to **MrmCreateResourceIndexer** or one of the related **MrmCreateResourceIndexer...*** functions.

packagingMode [in]

Type: **MrmPackagingMode**

Specifies what kind of PRI file the function should create. You cannot use **MrmPackagingModeAutoSplit** with this function. See the documentation for **MrmPackagingMode** and the **Remarks** section of **MrmCreateResourceFile** for more info.

packagingOptions [in]

Type: MrmPackagingOptions

Specifies additional options about the PRI file. Most callers should specify **MrmPackagingOptionsNone**. See the documentation for MrmPackagingOptions for more info.

outputPriData [out]

Type: **BYTE****

The address of a **BYTE** pointer. On successful return, contains a pointer to the buffer allocated by the function that contains the generated PRI file. You must free the memory by calling MrmFreeMemory when you are done with it.

outputPriSize [out]

Type: **ULONG***

The address of a **ULONG**. On successful return, contains the size of the allocated memory buffer pointed to by *outputPriData*.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise some other value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Remarks

If you intend to pass *outputPriData* to MrmCreateResourceIndexerFromPreviousPriData, don't free the memory until after you've finished using the resource indexer.

For more information about the **packagingMode** parameter, see the **Remarks** section of MrmCreateResourceFile. In particular, if you intend to save the data to a file yourself, be sure to follow the correct naming conventions.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

MrmCreateResourceFile

Package resource indexing (PRI) APIs and custom build systems

Feedback

MrmCreateResourceIndexer function

Article • 09/17/2024

Creates a resource indexer, used to generate package resource index (PRI) files for both packaged and unpackaged desktop apps.

COM must be initialized (e.g. by calling **CoInitializeEx**) before using this function.

Syntax

C++

```
HRESULT HRESULT MrmCreateResourceIndexer(
  _In_     PCWSTR                   packageFamilyName,
  _In_     PCWSTR                   projectRoot,
  _In_     MrmPlatformVersion       platformVersion,
  _In_opt_ PCWSTR                   defaultQualifiers,
  _Inout_  MrmResourceIndexerHandle *indexer
);
```

Parameters

packageFamilyName [in]

Type: **PCWSTR**

The package family name (PFN) of the app for which you will be generating PRI files. If you are building a PRI file for a packaged app, this *must* match the PFN of the app (as listed in the AppxManifest). If you are building a PRI file for an unpackaged app, this *must* be the string "**Application**".

projectRoot [in]

Type: **PCWSTR**

The root directory from which some file paths will be computed. Typically this will be the root directory of your source project, but may differ. See File resources in MRM for more information.

platformVersion [in]

Type: **MrmPlatformVersion**

The platform version (*targetOsVersion*) to use for the generated configuration file. Most callers should just use **MrmPlatformVersion_Windows10_0_0_5**

defaultQualifiers [in, optional]

Type: **PCWSTR**

A list of default resource qualifiers. For example, "language-en-US_scale-100". For more information about qualifiers, see Qualifiers in MRM.

indexer [in, out]

Type: **MrmResourceIndexerHandle***

A pointer to a resource indexer handle. On successful return, this will contain a handle to a resource indexer. You must free the indexer via **MrmDestroyIndexerAndMessages** after using it.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise some other value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Remarks

Most of the MRM APIs require an indexer handle to operate. The handle is created by this function or one of the related functions listed below under **See Also**.

Use **MrmDestroyIndexerAndMessages** to release the resources associated with the indexer after using it.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h

Requirement	Value
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

MrmCreateResourceIndexerFromPreviousPriData

MrmCreateResourceIndexerFromPreviousPriFile

MrmCreateResourceIndexerFromPreviousSchemaData

MrmCreateResourceIndexerFromPreviousSchemaFile

Package resource indexing (PRI) APIs and custom build systems

Feedback

MrmCreateResourceIndexerFromPreviousPriData function

Article • 09/17/2024

Creates a resource indexer used to create PRI files for use in resource packages.

This function is not needed if you created the original PRI file(s) with the **MrmPackagingModeStandaloneFile** or **MrmPackagingModeAutoSplit** packaging mode. If you are building resources for an unpackaged desktop app, you *cannot* use this function since only stand-alone PRI files are supported for unpackaged apps.

COM must be initialized (e.g. by calling **CoInitializeEx**) before using this function.

Syntax

C++

```
HRESULT HRESULT MrmCreateResourceIndexerFromPreviousPriData (
  _In_     PCWSTR                   projectRoot,
  _In_     MrmPlatformVersion       platformVersion,
  _In_opt_ PCWSTR                   defaultQualifiers,
  _In_     BYTE                     *priData,
  _In_     ULONG                    priSize,
  _Inout_  MrmResourceIndexerHandle *indexer
);
```

Parameters

projectRoot [in]

Type: **PCWSTR**

The root directory from which some file paths will be computed. Typically this will be the root directory of your source project, but may differ. See File resources in MRM for more information.

platformVersion [in]

Type: **MrmPlatformVersion**

The platform version (*targetOsVersion*) to use for the generated configuration file. Most callers should just use **MrmPlatformVersion_Windows10_0_0_5**

defaultQualifiers [in, optional]

Type: **PCWSTR**

A list of default resource qualifiers. For example, "language-en-US_scale-100". For more information about qualifiers, see Qualifiers in MRM.

priData [in]

Type: **BYTE***

A pointer to an in-memory PRI file. You can obtain an in-memory PRI file either by manually loading an existing PRI file from disk, or by creating it in-memory with **MrmCreateResourceFileInMemory**.

priSize [in]

Type: **ULONG**

The size of the data pointed to by *priData*.

indexer [in, out]

Type: **MrmResourceIndexerHandle***

A pointer to a resource indexer handle. On successful return, this will contain a handle to a resource indexer. You must free the indexer via **MrmDestroyIndexerAndMessages** after using it.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise some other value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Remarks

See the **Remarks** section of **MrmCreateResourceIndexerFromPreviousPriFile** for more info, as this function is essentially the same (except it uses in-memory PRI rather than an on-disk file).

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

MrmCreateResourceIndexer

MrmCreateResourceIndexerFromPreviousPriFile

MrmCreateResourceIndexerFromPreviousSchemaData

MrmCreateResourceIndexerFromPreviousSchemaFile

Package resource indexing (PRI) APIs and custom build systems

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MrmCreateResourceIndexerFromPreviousPriFile function

Article • 09/17/2024

Creates a resource indexer used to create PRI files for use in resource packages.

This function is not needed if you created the original PRI file(s) with the **MrmPackagingModeStandaloneFile** or **MrmPackagingModeAutoSplit** packaging mode. If you are building resources for an unpackaged desktop app, you *cannot* use this function since only stand-alone PRI files are supported for unpackaged apps.

COM must be initialized (e.g. by calling **CoInitializeEx**) before using this function.

Syntax

```
C++
```

```
HRESULT HRESULT MrmCreateResourceIndexerFromPreviousPriFile(
  _In_     PCWSTR                   projectRoot,
  _In_     MrmPlatformVersion       platformVersion,
  _In_opt_ PCWSTR                   defaultQualifiers,
  _In_     PCWSTR                   priFile,
  _Inout_  MrmResourceIndexerHandle *indexer
);
```

Parameters

projectRoot [in]

Type: **PCWSTR**

The root directory from which some file paths will be computed. Typically this will be the root directory of your source project, but may differ. See File resources in MRM for more information.

platformVersion [in]

Type: **MrmPlatformVersion**

The platform version (*targetOsVersion*) to use for the generated configuration file. Most callers should just use **MrmPlatformVersion_Windows10_0_0_5**

defaultQualifiers [in, optional]

Type: **PCWSTR**

A list of default resource qualifiers. For example, "language-en-US_scale-100". For more information about qualifiers, see Qualifiers in MRM.

priFile [in]

Type: **PCWSTR**

A fully-qualified path to an existing PRI file.

indexer [in, out]

Type: **MrmResourceIndexerHandle***

A pointer to a resource indexer handle. On successful return, this will contain a handle to a resource indexer. You must free the indexer via **MrmDestroyIndexerAndMessages** after using it.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise some other value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Remarks

See general remarks about creating a resource indexer in the **Remarks** section of **MrmCreateResourceIndexer**.

This function is used to create language- or scale-specific PRI files that are used in addition to a primary PRI file (the one passed as *priFile*). This is only useful if you are building a packaged MSIX app and intend to build separate resource packages for the target language(s) / scale(s). In all other cases, this function is not needed.

For example, assume your default resources are in English and you created a PRI file for them. Now you have localized those resources into German and wish to create a German-language resource pack for your app.

1. First you would create your English-language PRI file, saving it to disk.

2. Then you would call this function, passing in the path to the English-language PRI file as the *priFile* parameter and making sure "language-de" was one of the *defaultQualifiers*.
3. Next you would add all the German-language resource candidates to the indexer. Note that you do not need to have a German candidate for every resource; for those where the English resource is applicable (e.g. non-localized text or images) the German candidate can be omitted.
4. Finally, you would save the German-languaeg PRI file using the **MrmPackagingModeResourcePack** packaging mode, placing it in a different directory than your English-language PRI file.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

MrmCreateResourceIndexer

MrmCreateResourceIndexerFromPreviousPriData

MrmCreateResourceIndexerFromPreviousSchemaData

MrmCreateResourceIndexerFromPreviousSchemaFile

Package resource indexing (PRI) APIs and custom build systems

Feedback

Was this page helpful?

Yes No

Provide product feedback | Get help at Microsoft Q&A

MrmCreateResourceIndexerFromPreviousSchemaData function

Article • 09/17/2024

Creates a resource indexer that can create PRI files that are compatible with existing PRI files. This function is only needed in fairly limited scenarios; see the **Remarks** section of MrmCreateResourceIndexerFromPreviousSchemaFile for more info.

COM must be initialized (e.g. by calling CoInitializeEx) before using this function.

Syntax

C++

```
HRESULT HRESULT MrmCreateResourceIndexerFromPreviousSchemaData(
  _In_     PCWSTR                   projectRoot,
  _In_     MrmPlatformVersion       platformVersion,
  _In_opt_ PCWSTR                   defaultQualifiers,
  _In_     BYTE                     *schemaXmlData,
  _In_     ULONG                    schemaXmlSize,
  _Inout_  MrmResourceIndexerHandle *indexer
);
```

Parameters

projectRoot [in]

Type: **PCWSTR**

The root directory from which some file paths will be computed. Typically this will be the root directory of your source project, but may differ. See File resources in MRM for more information.

platformVersion [in]

Type: MrmPlatformVersion

The platform version (*targetOsVersion*) to use for the generated configuration file. Most callers should just use **MrmPlatformVersion_Windows10_0_0_5**

defaultQualifiers [in, optional]

Type: **PCWSTR**

A list of default resource qualifiers. For example, "language-en-US_scale-100". For more information about qualifiers, see Qualifiers in MRM.

schemaXmlData [in]

Type: **BYTE***

A pointer to an in-memory PRI file or in-memory Schema XML dump. You can obtain an in-memory PRI file either by manually loading an existing PRI file from disk, or by creating it in-memory with **MrmCreateResourceFileInMemory**. You can obtain an in-memory Schema XML dump either by manually loading an existing file from disk, or by using one of the **MrmDump...** functions.

schemaXmlSize [in]

Type: **ULONG**

The size of the data pointed to by *schemaXmlData*.

indexer [in, out]

Type: **MrmResourceIndexerHandle***

A pointer to a resource indexer handle. On successful return, this will contain a handle to a resource indexer. You must free the indexer via **MrmDestroyIndexerAndMessages** after using it.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise some other value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Remarks

See the **Remarks** section of **MrmCreateResourceIndexerFromPreviousSchemaFile** for more info, as this function is essentially the same (except it uses in-memory reference PRI rather than an on-disk file).

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

MrmCreateResourceIndexer

MrmCreateResourceIndexerFromPreviousPriData

MrmCreateResourceIndexerFromPreviousPriFile

MrmCreateResourceIndexerFromPreviousSchemaFile

Package resource indexing (PRI) APIs and custom build systems

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MrmCreateResourceIndexerFromPreviousSchemaFile function

Article • 09/17/2024

Creates a resource indexer that can create PRI files that are compatible with existing PRI files. This function is only needed in fairly limited scenarios; see **Remarks** for more info.

COM must be initialized (e.g. by calling **CoInitializeEx**) before using this function.

Syntax

```
C++

HRESULT HRESULT MrmCreateResourceIndexerFromPreviousSchemaFile(
  _In_     PCWSTR                   projectRoot,
  _In_     MrmPlatformVersion       platformVersion,
  _In_opt_ PCWSTR                   defaultQualifiers,
  _In_     PCWSTR                   schemaFile,
  _Inout_  MrmResourceIndexerHandle *indexer
);
```

Parameters

projectRoot [in]

Type: **PCWSTR**

The root directory from which some file paths will be computed. Typically this will be the root directory of your source project, but may differ. See File resources in MRM for more information.

platformVersion [in]

Type: **MrmPlatformVersion**

The platform version (*targetOsVersion*) to use for the generated configuration file. Most callers should just use **MrmPlatformVersion_Windows10_0_0_5**

defaultQualifiers [in, optional]

Type: **PCWSTR**

A list of default resource qualifiers. For example, "language-en-US_scale-100". For more information about qualifiers, see Qualifiers in MRM.

schemaFile [in]

Type: **PCWSTR**

The path to an existing PRI file whose schema you need to match, or the path to an XML dump containing the schema of the PRI file to match. See **MrmDumpPriFile** for info on creating an XML dump.

indexer [in, out]

Type: **MrmResourceIndexerHandle***

A pointer to a resource indexer handle. On successful return, this will contain a handle to a resource indexer. You must free the indexer via **MrmDestroyIndexerAndMessages** after using it.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise some other value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Remarks

This function is equivalent to **MrmCreateResourceIndexer** except that the indexer will create PRI files that are guaranteed to include all the resource names and indices of the provided *schemaFile* in addition to any new resources added to the indexer. This ensures that the new PRI file is "schema-compatible" with the original *schemaFile*, even if some resources have been omitted.

For most common scenarios, this function is not needed. If either of the following apply, you do not need to worry about building schema-compatible PRI files:

- You use a single, stand-alone PRI file (which is the only supported option for unpackaged apps); **or**
- You always rebuild your resource pack PRIs when rebuilding your main PRI.

This function is only needed in a fairly limited scenario, namely:

1. You have a main PRI file and one or more resource-pack PRI files; **and**

2. You are rebuilding the main PRI file (but not the resource-pack PRI files); **and**
3. You are (or might be) removing one or more resources from the main PRI file.

In this situation, setting the *schemaFile* reference to the original main PRI file will ensure that the new main PRI file is compatible with the existing resource packs, by ensuring there are placeholder index values for the removed resources.

Note: The new PRI file will contain all the resource names (and indices) from the original file but will only contain values for the resources explicitly added to the indexer. The new file *does not* inherit (copy) the values from the original PRI file.

Example

Assume the PRI file "original.pri" contains three resources for English:

Expand table

Name	Index	Value
ms-resource:///strings/test	0	"TestValue"
ms-resource:///strings/save	1	"Save"
ms-resource:///strings/delete	2	"Delete"

Now consider the following code snippet (error checking omitted):

C++

```
// Create an indexer with the existing schema
MrmResourceIndexerHandle indexer{};
MrmCreateResourceIndexerFromPreviousSchemaFile(L"C:\\MyProject",
MrmPlatformVersion_Windows10_0_0_5, L"language-en", L"original.pri",
&indexer);

// Add values for "save" and "delete".
// Note that there is no "test" resource added, since we don't need it
anymore.
MrmIndexString(indexer, L"ms-resource:///strings/save", L"Save", L"language-
en");
MrmIndexString(indexer, L"ms-resource:///strings/delete", L"Delete",
L"language-en");

// Save the file
MrmCreateResourceFile(indexer, MrmPackagingModeStandaloneFile,
MrmPackagingOptionsNone, L"C:\\");
```

The new resource file will contain the following for English:

Expand table

Name	Index	Value
ms-resource:///strings/test	0	<none>
ms-resource:///strings/save	1	"Save"
ms-resource:///strings/delete	2	"Delete"

This ensures that the resource indices for `save` and `delete` remain the same (1 and 2, respectively), so that resource packs with localized versions of "Save" and "Delete" will match. Note that if the application code still tried to load the `test` resource, it would fail at runtime (at least in English) since there are no candidates defined.

To illustrate why this is important, let's see what happens if the new PRI file had not been created with the base schema. In that case the new PRI file would contain the following:

Expand table

Name	Index	Value
ms-resource:///strings/save	0	"Save"
ms-resource:///strings/delete	1	"Delete"

Now resoure lookups for `delete` (index 1) would retrieve the correct string "Delete" in English, but in other languages it would retrieve the localized version of "Save" (since index 1 used to be the index for the `save` resource). This would lead to users seeing UX labelled "Save" (in their local language) that actually performed a delete operation - a serious bug.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]

Requirement	Value
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

MrmCreateResourceIndexer

MrmCreateResourceIndexerFromPreviousPriData

MrmCreateResourceIndexerFromPreviousPriFile

MrmCreateResourceIndexerFromPreviousSchemaData

Package resource indexing (PRI) APIs and custom build systems

Feedback

MrmDestroyIndexerAndMessages function

Article • 09/17/2024

Releases resources associated with a resource indexer. Destroys the handle, frees the indexer, and deletes any messages retrieved via **MrmPeekResourceIndexerMessages**.

Note this *does not* free memory allocated via functions such as **MrmCreateConfigInMemory** or **MrmCreateResourceFileInMemory** that produce outputs; that memory must be freed by using **MrmFreeMemory**, as outlined in the specific API topics.

Syntax

```
C++

HRESULT HRESULT MrmDestroyIndexerAndMessages(
  _In_ MrmResourceIndexerHandle indexer
);
```

Parameters

indexer [in]

Type: **MrmResourceIndexerHandle**

A handle identifying the resource indexer to destroy.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise some other value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

MrmFreeMemory

Package resource indexing (PRI) APIs and custom build systems

Feedback

MrmDumpPriFile function

Article • 09/17/2024

Dumps a PRI file to its XML equivalent. PRI files are stored in an undocumented binary format, so in order to view the contents of a file for debugging etc. it must be saved ("dumped") as an XML file.

This function performs the equivalent of the `makepri dump` command.

COM must be initialized (e.g. by calling **CoInitializeEx**) before using this function.

Syntax

```
C++

HRESULT HRESULT MrmDumpPriFile(
  _In_     PCWSTR      indexFileName,
  _In_opt_ PCWSTR      schemaPriFile,
  _In_     MrmDumpType dumpType,
  _In_     PCWSTR      outputXmlFile
);
```

Parameters

indexFileName [in]

Type: **PCWSTR**

The path to the PRI file to dump. If this file does not have an embedded schema, then *schemaPriFile* is required.

schemaPriFile [in, optional]

Type: **PCWSTR**

The path to the PRI file that provides the scheme for the *indexFileName* if needed, otherwise **NULL**. See **Remarks** for more info.

dumpType [in]

Type: **MrmDumpType**

Specified the kind of dump to create. For most use-cases, **MrmDumpTypeBasic** is sufficient. See the **MrmDumpType** reference for more info.

outputXmlFile [in]

Type: **PCWSTR**

The path of the XML file to create. If the file already exists, it will be overwritten.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise some other value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Remarks

You can create XML dumps from a PRI file, but there is no equivalent function to build a PRI file from an XML dump - that must be done manually by using the other Resource Indexer APIs.

When to use a *schemaPriFile*

In a PRI file, resources are identified by both a name and an index (known as the "Schema"). When a resource-pack PRI file is created, the resource names can be omitted to save space. The resource-pack PRI file will contain only the resource indexes, not the names. In order to produce the dump file (which includes resource names), the original base PRI file is required.

A schema-free resource pack PRI file is created by using the **MrmPackagingOptionsOmitSchemaFromResourcePacks** option when creating a resource file via one of the **MrmCreateResourceFile...** functions. When dumping such a resource-pack PRI file, pass the original (main) PRI file as the *schemaPriFile* argument.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

MrmDumpPriDataInMemory

MrmDumpPriFileInMemory

Package resource indexing (PRI) APIs and custom build systems

Feedback

MrmDumpPriFileInMemory function

Article • 09/17/2024

Dumps a PRI file to its XML equivalent, storing the result in memory. PRI files are stored in an undocumented binary format, so in order to view the contents of a file for debugging etc. it must be saved ("dumped") as XML.

This function performs the equivalent of the `makepri dump` command, with the results being stored in memory.

COM must be initialized (e.g. by calling **CoInitializeEx**) before using this function.

Syntax

C++

```
HRESULT HRESULT MrmDumpPriFileInMemory(
  _In_     PCWSTR      indexFileName,
  _In_opt_ PCWSTR      schemaPriFile,
  _In_     MrmDumpType dumpType,
  _Out_    BYTE        **outputXmlData,
  _Out_    ULONG       *outputXmlSize
);
```

Parameters

indexFileName [in]

Type: **PCWSTR**

The path to the PRI file to dump. If this file does not have an embedded schema, then *schemaPriFile* is required.

schemaPriFile [in, optional]

Type: **PCWSTR**

The path to the PRI file that provides the schema for the *indexFileName* if needed, otherwise **NULL**. See the **Remarks** section of **MrmDumpPriFile** for more info.

dumpType [in]

Type: **MrmDumpType**

Specified the kind of dump to create. For most use-cases, **MrmDumpTypeBasic** is sufficient. See the **MrmDumpType** reference for more info.

outputXmlData [out]

Type: **BYTE****

The address of a **BYTE** pointer. On successful return, contains a pointer to the buffer allocated by the function that contains the generated XML content. You must free the memory by calling **MrmFreeMemory** when you are done with it.

outputXmlSize [out]

Type: **ULONG***

The address of a **ULONG**. On successful return, contains the size of the allocated memory buffer pointed to by *outputXmlData*.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise some other value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Remarks

See the **Remarks** section of **MrmDumpPriFile** for more info, as this function is essentially the same (except it dumps the XML to memory instead of to a disk file).

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib

Requirement	Value
DLL	Mrmsupport.dll

See also

MrmDumpPriDataInMemory

MrmDumpPriFile

Package resource indexing (PRI) APIs and custom build systems

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MrmDumpPriDataInMemory function

Article • 09/17/2024

Dumps an in-memory PRI file to its XML equivalent, storing the result in memory. PRI files are stored in an undocumented binary format, so in order to view the contents of a file for debugging etc. it must be saved ("dumped") XML.

This function performs the equivalent of the `makepri dump` command, but entirely in memory.

COM must be initialized (e.g. by calling **CoInitializeEx**) before using this function.

Syntax

C++

```
HRESULT HRESULT MrmDumpPriDataInMemory(
  _In_     BYTE        *inputPriData,
  _In_     ULONG       inputPriSize,
  _In_opt_ BYTE        *schemaPriData,
  _In_     ULONG       schemaPriSize,
  _In_     MrmDumpType dumpType,
  _Out_    BYTE        **outputXmlData,
  _Out_    ULONG       *outputXmlSize
);
```

Parameters

inputPriData [in]

Type: **BYTE***

A pointer to an in-memory PRI file. If this in-memory file does not have an embedded schema, then *schemaPriData* is required.

inputPriSize [in]

Type: **ULONG**

The size of the data pointed to by *inputPriData*.

schemaPriData [in, optional]

Type: **BYTE***

A pointer to an in-memory PRI file that provides the schema for *inputPriData* if needed, otherwise **NULL**. See the **Remarks** section of **MrmDumpPriFile** for more info.

schemaPriSize [in]

Type: **ULONG**

The size of the data pointed to by *schemaPriData*.

dumpType [in]

Type: **MrmDumpType**

Specified the kind of dump to create. For most debugging use-cases, **MrmDumpTypeBasic** is sufficient. See the **MrmDumpType** reference for more info.

outputXmlData [out]

Type: **BYTE****

The address of a **BYTE** pointer. On successful return, contains a pointer to the buffer allocated by the function that contains the generated XML content. You must free the memory by calling **MrmFreeMemory** when you are done with it.

outputXmlSize [out]

Type: **ULONG***

The address of a **ULONG**. On successful return, contains the size of the allocated memory buffer pointed to by *outputXmlData*.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise some other value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Remarks

See the **Remarks** section of **MrmDumpPriFile** for more info, as this function is essentially the same (except it works with in-memory data instead of files).

You can obtain an in-memory PRI file either by manually loading an existing PRI file from disk, or by creating it in-memory with **MrmCreateResourceFileInMemory**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

MrmDumpPriFile

MrmDumpPriFileInMemory

Package resource indexing (PRI) APIs and custom build systems

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MrmFreeMemory function

Article • 09/17/2024

Frees memory allocated by MrmCreateConfigInMemory, MrmCreateResourceFileInMemory, MrmDumpPriFileInMemory, and MrmDumpPriDataInMemory.

To free the indexer handle, use MrmDestroyIndexerAndMessages.

Syntax

C++

```
HRESULT HRESULT MrmFreeMemory(
  _In_ BYTE *data
);
```

Parameters

data [in]

Type: **BYTE***

A pointer to memory allocated and returned by MrmCreateConfigInMemory, MrmCreateResourceFileInMemory, MrmDumpPriFileInMemory, or MrmDumpPriDataInMemory.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise some other value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

MrmDestroyIndexerAndMessages

Package resource indexing (PRI) APIs and custom build systems

Feedback

MrmIndexEmbeddedData function

Article • 09/17/2024

Adds an embeded data resource (BLOB) to a Resource Indexer. Note that it is the resource name that is indexed, not the data. In other words, this function does not search the data looking for things to index.

Embedded data resources are more efficient than file resources, but are less useful during app development. See Files resources in MRM for more info.

Syntax

C++

```
HRESULT HRESULT MrmIndexEmbeddedData(
  _In_           MrmResourceIndexerHandle indexer,
  _In_           PCWSTR                   resourceUri,
  _In_     const BYTE                     *embeddedData,
  _In_           ULONG                    embeddedDataSize,
  _In_opt_       PCWSTR                   qualifiers
);
```

Parameters

indexer [in]

Type: **MrmResourceIndexerHandle**

A handle identifying the resource indexer to add the resource to. This handle is returned via a call to **MrmCreateResourceIndexer** or one of the related **MrmCreateResourceIndexer...*** functions.

resourceUri [in]

Type: **PCWSTR**

The URI (name) to assign to the resource. See Resource names in MRM for more info.

embeddedData [in]

Type: **const BYTE***

A pointer to the data you want to add to the indexer.

embeddedDataSize [in]

Type: **ULONG**

The size of the data pointed to by *embeddedData*.

qualifiers [in, optional]

Type: **PCWSTR**

An optional list of resource qualifiers for the resource, for example "language-en-US". Passing an empty string or **NULL** indicates a neutral resource that is applicable in any resource context. See Qualifiers in MRM for more info.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise some other value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Remarks

This function adds binary data (such as an image or audio file) directly to the indexer, so that it will be embedded in the output PRI file. This is in contrast to file-based resources, which only adds the filename to the PRI file and requires the file to be installed alongside the app.

See Files in MRM for more info on the pros and cons of using embedded data vs. file resources.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib

Requirement	Value
DLL	Mrmsupport.dll

See also

MrmIndexFile

MrmIndexFileAutoQualifiers

MrmIndexResourceContainerAutoQualifiers

MrmIndexString

File resources in MRM

Package resource indexing (PRI) APIs and custom build systems

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MrmIndexFile function

Article • 09/17/2024

Adds a file resource to a Resource Indexer. Note that it is the resource name that is indexed, not the file. In other words, this function does not search the filename (or file contents) looking for things to index.

See File resources in MRM for more info.

Syntax

```
C++

HRESULT HRESULT MrmIndexFile(
  _In_     MrmResourceIndexerHandle indexer,
  _In_     PCWSTR                   resourceUri,
  _In_     PCWSTR                   filePath,
  _In_opt_ PCWSTR                   qualifiers
);
```

Parameters

indexer [in]

Type: **MrmResourceIndexerHandle**

A handle identifying the resource indexer to add the resource to. This handle is returned via a call to **MrmCreateResourceIndexer** or one of the related **MrmCreateResourceIndexer...*** functions.

resourceUri [in]

Type: **PCWSTR**

The URI (name) to assign to the resource. By convention, file resources are added to the `files/` path (for example, `ms-resource:///files/logo.png`). See Resource names in MRM for more info.

filePath [in]

Type: **PCWSTR**

The file path to be added to the index. Typically, this should be a relative path to a file as it will exist after your app is intalled.

qualifiers [in, optional]

Type: **PCWSTR**

An optional list of resource qualifiers for the resource, for example "language-en-US". Passing an empty string or **NULL** indicates a neutral resource that is applicable in any resource context. See Qualifiers in MRM for more info.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise some other value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Remarks

The provided *filePath* is added directly to the indexer without any parsing. The file does not need to exist, and no checks are performed to see if it contains illegal path characters. The *resourceUri* is entirely unrelated to the file path, for example a file may have a *resourceUri* of `/files/logo.png` but a *filePath* of "Assets\Logos\2x\logo.png". Since the *filePath* is just a string, it can also contain absolute paths or parent-paths (those including "..") although such files may not exist at runtime.

Note that files added with **MrmIndexFile** do not get qualifiers automatically added based on the filename - any qualifiers must be explicitly added with the *qualifers* parameter. To automatically deduce qualifiers based on the filename (as is performed by Visual Studio and the **MakePri** tool), use **MrmIndexFileAutoQualifiers**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]

Requirement	Value
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

MrmIndexEmbeddedData

MrmIndexFileAutoQualifiers

MrmIndexResourceContainerAutoQualifiers

MrmIndexString

File resources in MRM

Package resource indexing (PRI) APIs and custom build systems

Feedback

MrmIndexFileAutoQualifiers function

Article • 09/17/2024

Adds a file resource to a Resource Indexer, inferring the resource name and qualifiers from the file path. Note that it is the resource name that is indexed, not the file. In other words, this function does not search the filename (or file contents) looking for things to index.

See File resources in MRM for more info.

Syntax

```
C++

HRESULT HRESULT MrmIndexFileAutoQualifiers(
  _In_ MrmResourceIndexerHandle indexer,
  _In_ PCWSTR                   filePath
);
```

Parameters

indexer [in]

Type: **MrmResourceIndexerHandle**

A handle identifying the resource indexer to add the resource to. This handle is returned via a call to **MrmCreateResourceIndexer** or one of the related **MrmCreateResourceIndexer...*** functions.

filePath [in]

Type: **PCWSTR**

The file path to be added to the index, and from which to infer both the resource name and the qualifiers. See the **Remarks** section below for more info.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise some other value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Note that this function may succeed even if you pass an invalid value for the *filePath* parameter. The error will be reported as **0x8007000B (HRESULT_FROM_WIN32(ERROR_BAD_FORMAT))** when you try to create the PRI file via MrmCreateResourceFile or one of the related functions. See **Remarks** for more info.

Remarks

This function is equivalent to MrmIndexFile except the *resourceUri* and *qualifiers* parameters are inferred from the *filePath* argument. Given a *filePath* of the form `path1\path2\pathn\filename`, the following basic algorithm is used:

1. Let `resourceName` be the string `ms-resource:///Files/`.
2. Let `qualifiers` be an empty string.
3. For each `path` segment in *filePath*, check if it consists of a valid qualifier list string (see Qualifiers in MRM for more info). If so, append those qualifiers to `qualifiers`, otherwise append it to `resourceName`.
4. For `filename`, split it up into segments separated by dots (`.`).
5. For each segment, if it is a valid qualifier list then add it to `qualifiers`, otherwise append it to `resourceName`.
6. Call the equivalent of **MrmIndexFile(indexer, resourceName, filePath, qualifiers)**

For example:

Expand table

filePath	resourecName	qualifiers
Assets**language-en\scale-200**\logo.png	ms-resource:///files/Assets/logo.png	language-en_scale-200
Images\icon.**scale-100.contrast-high**.jpg	ms-resource:///files/Images/icon.jpg	scale-100_contrast-high
Data\menu.txt	ms-resource:///files/Data/menu.txt	<N/A>
Images**scale-200**\HomePage**language-de**\welcome.**contrast-high**.png	ms-resource:///files/Images/HomePage/welcome.png	scale-200_language-de_contrast-high

Note that there is no way to determine what the inferred `resourceName` and `qualifiers` are for a given string; they are simply added to the indexer. The only way to determine

the values that were inferred is to create the PRI file and then dump it using one of the **MrmDump...** functions.

Valid file paths

Unlike **MrmIndexFile**, the *filePath* argument should be a legal file path, although the file does not need to exist. It should be a relative path, not including a parent directory escape (`..\`). The *projectRoot* specified during indexer creation can be used to generate relative file paths. For example, if the indexer was created with a *projectRoot* of `C:\Projects\MyApp` and you index the file `C:\Projects\MyApp\Assets\file.scale-200.jpg`, the function will automatically convert that to the relative filename `Assets\file.scale-200.jpg` (of course you could just add the relative name `Assets\file.scale-200.jpg` directly).

There are two important caveats with the validation of the *filePath* parameter:

1. Using forward slashes (`/`) instead of backslashes (`\`) bypasses the checking performed by this function, and can result in illegal filenames in the PRI file.
2. Invalid file paths are **not** detected when this function is called, but only when the resource file is created on.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

MrmIndexEmbeddedData

MrmIndexFile

MrmIndexResourceContainerAutoQualifiers

MrmIndexString

File resources in MRM

Package resource indexing (PRI) APIs and custom build systems

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MrmIndexResourceContainerAutoQualifiers function

Article • 09/17/2024

Adds the string resources embedded in a resource container to a Resource Indexer, inferring the resource names and qualifiers from the container path. A resource container is a file such as a `resw` or `resjson` file that contains a list of resource name / value pairs. You can also add the contents of an existing existing PRI file to the indexer with this function. Note that it is the resource names inside the container that are indexed, not the values.

Syntax

```
C++

HRESULT HRESULT MrmIndexResourceContainerAutoQualifiers(
  _In_ MrmResourceIndexerHandle indexer,
  _In_ PCWSTR                   containerPath
);
```

Parameters

indexer [in]

Type: **MrmResourceIndexerHandle**

A handle identifying the resource indexer to add the resources to. This handle is returned via a call to **MrmCreateResourceIndexer** or one of the related **MrmCreateResourceIndexer...*** functions.

containerPath [in]

Type: **PCWSTR**

A path to a `resw`, `resjson`, or `.pri` file containing string resources that you want to add to the index. This path is relative to the project root specified when creating the indexer via one of the **MrmCreateResourceIndexer...** functions. The file must exist.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise some other value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Remarks

This function infers the resource names and qualifiers from both *containerPath* and the resources inside the container itself. The file identified by *containerPath* does not need to be included in your final app installation package.

The algorithm for computing resource names differs depending on the file type:

For resw and resjson files:

Given a *containerPath* of the form `path1\path2\pathn\filename.ext`, the following basic algorithm is used:

1. Let `resourceName` be the string `ms-resource:///`.
2. Let `qualifiers` be an empty string.
3. For each `path` segment in *filePath*, check if it consists of a valid qualifier list string (see Qualifiers in MRM for more info). If so, append those qualifiers to `qualifiers`, otherwise ignore it.
4. For `filename`, split it up into segments separated by dots (`.`). Note that `.ext` is ignored.
    - If there are exactly two segments, and the second segment is a valid qualifier list, add it to `qualifiers` and append the first segment to `resourceName`.
    - Otherwise, append the entire `filename` (but not `.ext`) `resourceName`.
5. For each resource name inside `filename`, replaces any dots (`.`) with forward slashes (`/`) and append it to `resourceName`, then add to the indexer with the specified value and `qualifiers`.
    - Note there is no attempt to infer qualifiers from the resource name inside the container; the name is just added verbatim to the `resourceName`

For example:

Expand table

filePath	resource name (inside file)	resourceName	qualifiers
resources.resw	my_string	ms-resource:///resources/my_string	<N/A>
images.resw	logo.scale-200.png	ms-resource:///images/logo/scale-200/png	<N/A>
language-es\text.**homeregion-mx**.resjson	greeting	ms-resource:///text/greeting	language-es_homeregion-mx
content**language-jp**\menu.scale-100.submenu.resjson	title.text	ms-resource:///menu.scale-100.submenu/title/text	language-jp

The last line of the table illustrates three of the more subtle parts of the algorithm:

1. Directory names that aren't qualifiers are ignored (in this case, "content").
2. Embedded qualifiers in filenames are ignored if there are additional segments (in this case, ".submenu").
3. Dots inside resource names are turned into slashes (but dots inside file paths are retained).

In some cases, having embedded dots or slashes inside a resource name inside a container will lead to an error **0x80073b39** when creating the resource file (not when adding it to the index). In general, it is best to avoid embedded slashes or dots in names, and to avoid embedded qualifiers since they will be ignored.

For PRI files:

Because PRI files already contain resources with qualifier information, no qualifiers are inferred from *filePath*; it is simply the name of the file to read. When indexing resources from within the PRI file, for a given resource name `ms-resource:///rootPath/theRest...` the following algorithm is used:

1. If the type of resource is `String`, then replace `rootPath` with "strings" and use the result as the resource name.
2. Otherwise (the type of the resource is `Path` or `EmbeddedData`), use the resource name as-is.

The value and qualifiers are added verbatim from the PRI file, with this new resource name.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

MrmIndexEmbeddedData

MrmIndexFile

MrmIndexFileAutoQualifiers

MrmIndexString

File resources in MRM

Package resource indexing (PRI) APIs and custom build systems

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MrmIndexString function

Article • 09/17/2024

Adds a string resource to a Resource Indexer. Note that it is the resource name that is indexed, not the string. In other words, this function does not search the string looking for things to index.

Syntax

C++

```
HRESULT HRESULT MrmIndexString(
  _In_     MrmResourceIndexerHandle indexer,
  _In_     PCWSTR                   resourceUri,
  _In_     PCWSTR                   resourceString,
  _In_opt_ PCWSTR                   qualifiers
);
```

Parameters

indexer [in]

Type: **MrmResourceIndexerHandle**

A handle identifying the resource indexer to add the resource to. This handle is returned via a call to **MrmCreateResourceIndexer** or one of the related **MrmCreateResourceIndexer...*** functions.

resourceUri [in]

Type: **PCWSTR**

The URI (name) to assign to the resource. By convention, string resources are added to the `strings/` path (for example, `ms-resource:///strings/appName`). See Resource names in MRM for more info.

resourceString [in]

Type: **PCWSTR**

The string to add to the index.

qualifiers [in, optional]

Type: **PCWSTR**

An optional list of resource qualifiers for the resource, for example "language-en-US". Passing an empty string or **NULL** indicates a neutral resource that is applicable in any resource context. See Qualifiers in MRM for more info.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise some other value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

MrmIndexEmbeddedData

MrmIndexFile

MrmIndexFileAutoQualifiers

MrmIndexResourceContainerAutoQualifiers

File resources in MRM

Package resource indexing (PRI) APIs and custom build systems

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MrmPeekResourceIndexerMessages function

Article • 09/17/2024

Retrieves diagnostic messages generated by a resource indexer. These may help in debugging problems with the indexer, although not all errors are reported.

Syntax

C++

```
HRESULT HRESULT MrmPeekResourceIndexerMessages(
  _In_  MrmResourceIndexerHandle  indexer,
  _Out_ MrmResourceIndexerMessage **messages,
  _Out_ ULONG                     *numMsgs
);
```

Parameters

indexer [in]

Type: **MrmResourceIndexerHandle**

A handle identifying the resource indexer to retrieve messages from. This handle is returned via a call to **MrmCreateResourceIndexer** or one of the related **MrmCreateResourceIndexer...*** functions.

messages [out]

Type: **MrmResourceIndexerMessage****

The address of an **MrmResourceIndexerMessage** pointer. On successful return, contains an array of **MrmResourceIndexerMessage** structures.

numMsgs [out]

Type: **ULONG***

The number of messages returned in *messages*.

Return value

Type: **HRESULT**

S_OK if the function succeeded, otherwise some other value. Use the **SUCCEEDED** or **FAILED** macros (defined in winerror.h) to determine success or failure.

Remarks

Do not free the memory pointed to by **messages* as it is owned by the indexer. The memory will automatically be freed when calling **MrmDestroyIndexerAndMessages** after you have finished using the indexer.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h
Library	Mrmsupport.lib
DLL	Mrmsupport.dll

See also

MrmResourceIndexerMessage

Package resource indexing (PRI) APIs and custom build systems

MrmResourceIndexerHandle structure

Article • 09/17/2024

Represents a handle to a resource indexer object. Most MRM functions require an indexer handle, which can be obtained via one of the **MrmCreateResourceIndexer...** functions.

Syntax

C++

```
typedef struct _MrmResourceIndexerHandle {
  PVOID handle;
} MrmResourceIndexerHandle, *PMrmResourceIndexerHandle;
```

Members

handle

Type: **PVOID**

An opaque handle to a resource indexer object. Do not use this value directly.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h

See also

MrmCreateResourceIndexer

MrmCreateResourceIndexerFromPreviousPriData

MrmCreateResourceIndexerFromPreviousPriFile

MrmCreateResourceIndexerFromPreviousSchemaData

MrmCreateResourceIndexerFromPreviousSchemaFile

MrmDestroyIndexerAndMessages

Package resource indexing (PRI) APIs and custom build systems

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MrmResourceIndexerMessage structure

Article • 09/17/2024

Represents a message generated by a resource indexer. Messages are retrieve by calling the MrmPeekResourceIndexerMessages function.

Syntax

```
C++
```

```
typedef struct _MrmResourceIndexerMessage {
  MrmResourceIndexerMessageSeverity severity;
  ULONG                             id;
  PCWSTR                            text;
} MrmResourceIndexerMessage, *PMrmResourceIndexerMessage;
```

Members

severity

Type: MrmResourceIndexerMessageSeverity

The severity of the message.

id

Type: **ULONG**

The unique identifier of the message.

text

Type: **PCWSTR**

The text of the message. Do not free this memory.

Remarks

Do not free the memory pointed to by *text* or the message structure itself. See MrmPeekResourceIndexerMessages for more info.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h

See also

MrmPeekResourceIndexerMessages

Package resource indexing (PRI) APIs and custom build systems

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MrmDumpType enumeration

Article • 09/17/2024

Defines constants that specify the type of PRI file dump to produce when calling one of the **MrmDumpPri...** functions.

Syntax

```
C++
```

```
typedef enum _MrmDumpType {
  MrmDumpType_Basic     = 0,
  MrmDumpType_Detailed  = 1,
  MrmDumpType_Schema    = 2
} MrmDumpType;
```

Constants

MrmDumpType_Basic

Include string and file resources, but exclude embedded binary data. This is generally the most useful dump type for debugging.

MrmDumpType_Detailed

Include all resources, including embedded binary data, plus additional indexing information. This is most useful for round-tripping XML back to a PRI file or debugging advanced problems.

MrmDumpType_Schema

Include only resource names and indices, not candidates. This is the smallest dump type, and can be used when building versioned PRI files (see **Remarks** for more info).

Remarks

PRI files are stored an undocumented binary format, so the only way to view their contents is to dump them to XML via one of the **MrmDumpPri...** functions. Dumping the XML is useful for three purposes:

1. The XML is human-readable, enabling you to see the contents of the file, debug issues with unresolved resources or incorrectly-qualified candidates, and so on.
2. The XML is machine-readable, making it possible for tools to manipulate PRI files and generate new PRI files via the intermediate XML format.

    - *Note that there is no built-in mechanism to turn an XML dump back into a PRI file; it must be done manually by creating a Resource Indexer, adding the resource candidates from the XML file, and writing out the new file.*

3. The **Schema** dump type can be used to create versioned PRI files.

The **Basic** dump type is sufficient for most debugging purposes. It lists all resource names and candidate values, with the exception of embedded binary data - but since embedded binary data isn't human-readable anyway, that is typically not a problem.

The **Detailed** dump type is the most complete - including BASE64-encoded embedded binary data - but includes a lot of extra data that make it harder to read (especially if there are large embedded binary resources). A **Detailed** dump file can be used to round-trip back into the original PRI file.

The **Schema** dump type is the smallest dump type, and only includes information about the resource names and indices in the PRI file (the "Schema" of the file). A **Schema** dump can be used with one of the **MrmCreateResourceIndexerFromPreviousSchema...** functions to create versionable PRI files - see the **Remarks** in MrmCreateResourceIndexerFromPreviousSchemaFile for more info. Note there is no guarantee that the resources listed in a **Schema** dump actually have candidates defined in the PRI file, so this is insufficient for debugging "missing" resource errors.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h

See also

MrmDumpPriDataInMemory

MrmDumpPriFile

MrmDumpPriFileInMemory

Package resource indexing (PRI) APIs and custom build systems

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MrmPackagingMode enumeration

Article • 09/17/2024

Defines constants that specify what kind of PRI file(s) should be created by MrmCreateResourceFile and MrmCreateResourceFileInMemory.

Syntax

```
C++
```

```
typedef enum _MrmPackagingMode {
  MrmPackagingModeStandaloneFile  = 0,
  MrmPackagingModeAutoSplit       = 1,
  MrmPackagingModeResourcePack    = 2
} MrmPackagingMode;
```

Constants

MrmPackagingModeStandaloneFile

A single PRI file should be created containing all resources. This is the most common usage, and the only supported mode for unpackaged desktop apps.

MrmPackagingModeAutoSplit

One or more PRI files should be created, determined by language or scale. See **Remarks** for more info.

MrmPackagingModeResourcePack

A single resource-pack PRI file should be created. See **Remarks** for more info.

Remarks

PRI files created for unpackaged apps must use **MrmPackagingModeStandaloneFile** as MRT Core only supports a single PRI file. For packaged apps, you should also use **MrmPackagingModeStandaloneFile** unless you plan on building Resource Packs to minimize download / install size. A single stand-alone PRI file contains all resources in all languages / scales, and so is usable in all target markets. If you are building a packaged app and want to create Resource Packs, you can use one of the other two values.

MrmPackagingModeAutoSplit creates a PRI file for each of the following kinds of resources:

1. A main PRI, including all resources with the default language qualifier and all neutral resources.
2. One language-specific PRI for each set of resources using a non-default language qualifier.
3. One scale-specific PRI for each scale factor that is not a default qualifier and did not also have a language specifier.

For example, assume the indexer was created with English as the default qualifer, and it contains resources with all the following qualifiers:

1. <none>
2. language-en
3. language-en_scale-100
4. language-de
5. language-de_scale-200
6. scale-400

In this case, 3 PRI files will be created:

- The file `resources.pri` will contain resources #1 (neutral) and #2 & #3 (default language).
- The file `resources.language-de.pri` will contain resources #4 & #5 (non-default language).
- The file `resources.scale-400.pri` will contain resource #6 (scale but no language).

Note that if the option **MrmPackagingOptionsSplitLanguageVariants** is specified when creating the PRI files, a PRI file will be created for each language variant (eg, en-US, en-AU, and en-GB) instead of just one for each language.

MrmPackagingModeResourcePack produces a single PRI file that contains resources in a single language. All resources must have the same language as the default qualifier language. The resource indexer *must* have been created using MrmCreateResourceIndexerFromPreviousPriFile or MrmCreateResourceIndexerFromPreviousPriData and passing the main PRI file as the *priFile* or *priData* parameters, respectively. Unlike the **AutoSplit** mode, there is no mechanism to create individual scale-based resource packs.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h

See also

MrmCreateResourceFile

MrmCreateResourceFileInMemory

Package resource indexing (PRI) APIs and custom build systems

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MrmPackagingOptions enumeration

Article • 09/17/2024

Defines constants that specify options for the PRI file created by MrmCreateResourceFile and MrmCreateResourceFileInMemory.

Syntax

```
C++
```

```
typedef enum _MrmPackagingOptions {
  MrmPackagingOptionsNone                          = 0x00,
  MrmPackagingOptionsOmitSchemaFromResourcePacks   = 0x01,
  MrmPackagingOptionsSplitLanguageVariants         = 0x02
} MrmPackagingOptions;
```

Constants

MrmPackagingOptionsNone

No options. Most callers should use this value.

MrmPackagingOptionsOmitSchemaFromResourcePacks

Resource Pack PRIs should be created without an embedded schema. See **Remarks** for more info.

MrmPackagingOptionsSplitLanguageVariants

Auto-split PRIs should be created based on language variants. See **Remarks** for more info.

Remarks

PRI files created for unpackaged desktop apps should specify **MrmPackagingOptionsNone** as resource packs are not supported. Most packaged apps should also specify **MrmPackagingOptionsNone** as it eases development.

MrmPackagingOptionsOmitSchemaFromResourcePacks is used when creating a resource pack PRIs, either with the **AutoSplit** or **ResourcePack** packaging modes. The resource pack PRIs will not contain the resource names (only the indexes) which reduces

the file size but makes certain debugging operations (such as dumping to XML) harder. See the **Remarks** section of MrmCreateResourceIndexerFromPreviousSchemaFile for additional info about schemas.

MrmPackagingOptionsSplitLanguageVariants is used when creating resource pack PRIs with the **AutoSplit** mode. In this case, rather than creating a single PRI per language, each language variant (eg en-US, en-AU, and en-GB) will have their own PRI files generated.

See MrmPackagingMode for more info.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h

See also

MrmCreateResourceFile

MrmCreateResourceFileInMemory

Package resource indexing (PRI) APIs and custom build systems

Feedback

MrmPlatformVersion enumeration

Article • 09/17/2024

Defines constants that specify the target platform version for a PRI file. The target platform version is the minimum version of Windows that the file can be used with. In most cases, you should just use the value **MrmPlatformVersion_Windows10_0_0_5** unless you need to run on older releases of Windows.

Syntax

```
C++
```

```
typedef enum _MrmPlatformVersion {
  MrmPlatformVersion_Default          = 0,
  MrmPlatformVersion_Windows10_0_0_0  = 0x010A0000,
  MrmPlatformVersion_Windows10_0_0_5  = 0x010A0005
} MrmPlatformVersion;
```

Constants

MrmPlatformVersion_Default

The PRI file is supported on the default platform version, which is currently the same as 10.0.0.0.

MrmPlatformVersion_Windows10_0_0_0

The PRI file is supported on the initial release of Windows 10 (build 10240) and above.

MrmPlatformVersion_Windows10_0_0_5

The PRI file is supported on the 1809 release of Windows 10 (build 17763) and above. Most callers should specify this value.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h

See also

MrmCreateResourceIndexer

MrmCreateResourceIndexerFromPreviousPriData

MrmCreateResourceIndexerFromPreviousPriFile

MrmCreateResourceIndexerFromPreviousSchemaData

MrmCreateResourceIndexerFromPreviousSchemaFile

Package resource indexing (PRI) APIs and custom build systems

Feedback

MrmResourceIndexerMessageSeverity enumeration

Article • 09/17/2024

Defines constants that specify the severity of an resource indexer message.

Syntax

C++

```
typedef enum _MrmResourceIndexerMessageSeverity {
  MrmResourceIndexerMessageSeverityVerbose  = 1,
  MrmResourceIndexerMessageSeverityInfo     = 2,
  MrmResourceIndexerMessageSeverityWarning  = 3,
  MrmResourceIndexerMessageSeverityError    = 4
} MrmResourceIndexerMessageSeverity;
```

Constants

MrmResourceIndexerMessageSeverityVerbose

Indicates that the message is verbose.

MrmResourceIndexerMessageSeverityInfo

Indicates that the message is informational.

MrmResourceIndexerMessageSeverityWarning

Indicates that the message is a warning.

MrmResourceIndexerMessageSeverityError

Indicates that the message is an error.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server [desktop apps only]
Header	MrmResourceIndexer.h

See also

MrmResourceIndexerMessage

Package resource indexing (PRI) APIs and custom build systems

Feedback