

# Contents

## Data Exchange

### Dde.h

Overview

DDEACK structure

DDEADVISE structure

DDEDATA structure

DDEPOKE structure

DdeSetQualityOfService function

FreeDDEIParam function

ImpersonateDdeClientWindow function

PackDDEIParam function

ReuseDDEIParam function

UnpackDDEIParam function

### Ddeml.h

Overview

CONVCONTEXT structure

CONVINFO structure

DdeAbandonTransaction function

DdeAccessData function

DdeAddData function

DdeClientTransaction function

DdeCmpStringHandles function

DdeConnect function

DdeConnectList function

DdeCreateDataHandle function

DdeCreateStringHandleA function

DdeCreateStringHandleW function

DdeDisconnect function

DdeDisconnectList function

DdeEnableCallback function  
DdeFreeDataHandle function  
DdeFreeStringHandle function  
DdeGetData function  
DdeGetLastError function  
DdeImpersonateClient function  
DdeInitializeA function  
DdeInitializeW function  
DdeKeepStringHandle function  
DDEML\_MSG\_HOOK\_DATA structure  
DdeNameService function  
DdePostAdvise function  
DdeQueryConvInfo function  
DdeQueryNextServer function  
DdeQueryStringA function  
DdeQueryStringW function  
DdeReconnect function  
DdeSetUserHandle function  
DdeUnaccessData function  
DdeUninitialize function  
HSZPAIR structure  
MONCBSTRUCT structure  
MONCONVSTRUCT structure  
MONERRSTRUCT structure  
MONHSZSTRUCTA structure  
MONHSZSTRUCTW structure  
MONLINKSTRUCT structure  
MONMSGSTRUCT structure  
PFNCALLBACK callback function

Winbase.h

Overview

AddAtomA function

[AddAtomW function](#)

[DeleteAtom function](#)

[FindAtomA function](#)

[FindAtomW function](#)

[GetAtomNameA function](#)

[GetAtomNameW function](#)

[GlobalAddAtomA function](#)

[GlobalAddAtomExA function](#)

[GlobalAddAtomExW function](#)

[GlobalAddAtomW function](#)

[GlobalDeleteAtom function](#)

[GlobalFindAtomA function](#)

[GlobalFindAtomW function](#)

[GlobalGetAtomNameA function](#)

[GlobalGetAtomNameW function](#)

[InitAtomTable function](#)

[MAKEINTATOM macro](#)

## [Wingdi.h](#)

[Overview](#)

[METAFILEPICT structure](#)

## [Winuser.h](#)

[Overview](#)

[AddClipboardFormatListener function](#)

[ChangeClipboardChain function](#)

[CloseClipboard function](#)

[COPYDATASTRUCT structure](#)

[CountClipboardFormats function](#)

[EmptyClipboard function](#)

[EnumClipboardFormats function](#)

[GetClipboardData function](#)

[GetClipboardFormatNameA function](#)

[GetClipboardFormatNameW function](#)

GetClipboardOwner function  
GetClipboardSequenceNumber function  
GetClipboardViewer function  
GetOpenClipboardWindow function  
GetPriorityClipboardFormat function  
GetUpdatedClipboardFormats function  
IsClipboardFormatAvailable function  
OpenClipboard function  
RegisterClipboardFormatA function  
RegisterClipboardFormatW function  
RemoveClipboardFormatListener function  
SetClipboardData function  
SetClipboardViewer function

# Data Exchange

6/30/2020 • 9 minutes to read • [Edit Online](#)

Overview of the Data Exchange technology.

To develop Data Exchange, you need these headers:

- [dde.h](#)
- [ddeml.h](#)
- [wingdi.h](#)

For programming guidance for this technology, see:

- [Data Exchange](#)

## Functions

TITLE	DESCRIPTION
<a href="#">AddAtomA</a>	Adds a character string to the local atom table and returns a unique value (an atom) identifying the string.
<a href="#">AddAtomW</a>	Adds a character string to the local atom table and returns a unique value (an atom) identifying the string.
<a href="#">AddClipboardFormatListener</a>	Places the given window in the system-maintained clipboard format listener list.
<a href="#">ChangeClipboardChain</a>	Removes a specified window from the chain of clipboard viewers.
<a href="#">CloseClipboard</a>	Closes the clipboard.
<a href="#">CountClipboardFormats</a>	Retrieves the number of different data formats currently on the clipboard.
<a href="#">DdeAbandonTransaction</a>	Abandons the specified asynchronous transaction and releases all resources associated with the transaction.
<a href="#">DdeAccessData</a>	Provides access to the data in the specified Dynamic Data Exchange (DDE) object. An application must call the <a href="#">DdeUnaccessData</a> function when it has finished accessing the data in the object.
<a href="#">DdeAddData</a>	Adds data to the specified Dynamic Data Exchange (DDE) object.
<a href="#">DdeClientTransaction</a>	Begins a data transaction between a client and a server. Only a Dynamic Data Exchange (DDE) client application can call this function, and the application can use it only after establishing a conversation with the server.

TITLE	DESCRIPTION
<a href="#">DdeCmpStringHandles</a>	Compares the values of two string handles. The value of a string handle is not related to the case of the associated string.
<a href="#">DdeConnect</a>	Establishes a conversation with a server application that supports the specified service name and topic name pair. If more than one such server exists, the system selects only one.
<a href="#">DdeConnectList</a>	Establishes a conversation with all server applications that support the specified service name and topic name pair.
<a href="#">DdeCreateDataHandle</a>	Creates a Dynamic Data Exchange (DDE) object and fills the object with data from the specified buffer. A DDE application uses this function during transactions that involve passing data to the partner application.
<a href="#">DdeCreateStringHandleA</a>	Creates a handle that identifies the specified string. A Dynamic Data Exchange (DDE) client or server application can pass the string handle as a parameter to other Dynamic Data Exchange Management Library (DDEML) functions.
<a href="#">DdeCreateStringHandleW</a>	Creates a handle that identifies the specified string. A Dynamic Data Exchange (DDE) client or server application can pass the string handle as a parameter to other Dynamic Data Exchange Management Library (DDEML) functions.
<a href="#">DdeDisconnect</a>	Terminates a conversation started by either the DdeConnect or DdeConnectList function and invalidates the specified conversation handle.
<a href="#">DdeDisconnectList</a>	Destroys the specified conversation list and terminates all conversations associated with the list.
<a href="#">DdeEnableCallback</a>	Enables or disables transactions for a specific conversation or for all conversations currently established by the calling application.
<a href="#">DdeFreeDataHandle</a>	Frees a Dynamic Data Exchange (DDE) object and deletes the data handle associated with the object.
<a href="#">DdeFreeStringHandle</a>	Frees a string handle in the calling application.
<a href="#">DdeGetData</a>	Copies data from the specified Dynamic Data Exchange (DDE) object to the specified local buffer.
<a href="#">DdeGetLastError</a>	Retrieves the most recent error code set by the failure of a Dynamic Data Exchange Management Library (DDEML) function and resets the error code to DMLERR_NO_ERROR.
<a href="#">DdeImpersonateClient</a>	Impersonates a Dynamic Data Exchange (DDE) client application in a DDE client conversation.

TITLE	DESCRIPTION
<a href="#">DdeInitializeA</a>	Registers an application with the Dynamic Data Exchange Management Library (DDEML). An application must call this function before calling any other Dynamic Data Exchange Management Library (DDEML) function.
<a href="#">DdeInitializeW</a>	Registers an application with the Dynamic Data Exchange Management Library (DDEML). An application must call this function before calling any other Dynamic Data Exchange Management Library (DDEML) function.
<a href="#">DdeKeepStringHandle</a>	Increments the usage count associated with the specified handle.
<a href="#">DdeNameService</a>	Registers or unregisters the service names a Dynamic Data Exchange (DDE) server supports.
<a href="#">DdePostAdvise</a>	Causes the system to send an XTYP_ADVREQ transaction to the calling (server) application's Dynamic Data Exchange (DDE) callback function for each client with an active advise loop on the specified topic and item.
<a href="#">DdeQueryConvInfo</a>	Retrieves information about a Dynamic Data Exchange (DDE) transaction and about the conversation in which the transaction takes place.
<a href="#">DdeQueryNextServer</a>	Retrieves the next conversation handle in the specified conversation list.
<a href="#">DdeQueryStringA</a>	Copies text associated with a string handle into a buffer.
<a href="#">DdeQueryStringW</a>	Copies text associated with a string handle into a buffer.
<a href="#">DdeReconnect</a>	Enables a client Dynamic Data Exchange Management Library (DDEML) application to attempt to reestablish a conversation with a service that has terminated a conversation with the client.
<a href="#">DdeSetQualityOfService</a>	Specifies the quality of service (QOS) a raw Dynamic Data Exchange (DDE) application desires for future DDE conversations it initiates.
<a href="#">DdeSetUserHandle</a>	Associates an application-defined value with a conversation handle or a transaction identifier. This is useful for simplifying the processing of asynchronous transactions. An application can use the DdeQueryConvInfo function to retrieve this value.
<a href="#">DdeUnaccessData</a>	Unaccesses a Dynamic Data Exchange (DDE) object. An application must call this function after it has finished accessing the object.
<a href="#">DdeUninitialize</a>	Frees all Dynamic Data Exchange Management Library (DDEML) resources associated with the calling application.

TITLE	DESCRIPTION
<a href="#">DeleteAtom</a>	Decrements the reference count of a local string atom. If the atom's reference count is reduced to zero, DeleteAtom removes the string associated with the atom from the local atom table.
<a href="#">EmptyClipboard</a>	Empties the clipboard and frees handles to data in the clipboard. The function then assigns ownership of the clipboard to the window that currently has the clipboard open.
<a href="#">EnumClipboardFormats</a>	Enumerates the data formats currently available on the clipboard.
<a href="#">FindAtomA</a>	Searches the local atom table for the specified character string and retrieves the atom associated with that string.
<a href="#">FindAtomW</a>	Searches the local atom table for the specified character string and retrieves the atom associated with that string.
<a href="#">FreeDDEIParam</a>	Frees the memory specified by the IParam parameter of a posted Dynamic Data Exchange (DDE) message. An application receiving a posted DDE message should call this function after it has used the UnpackDDEIParam function to unpack the IParam value.
<a href="#">GetAtomNameA</a>	Retrieves a copy of the character string associated with the specified local atom.
<a href="#">GetAtomNameW</a>	Retrieves a copy of the character string associated with the specified local atom.
<a href="#">GetClipboardData</a>	Retrieves data from the clipboard in a specified format. The clipboard must have been opened previously.
<a href="#">GetClipboardFormatNameA</a>	Retrieves from the clipboard the name of the specified registered format. The function copies the name to the specified buffer.
<a href="#">GetClipboardFormatNameW</a>	Retrieves from the clipboard the name of the specified registered format. The function copies the name to the specified buffer.
<a href="#">GetClipboardOwner</a>	Retrieves the window handle of the current owner of the clipboard.
<a href="#">GetClipboardSequenceNumber</a>	Retrieves the clipboard sequence number for the current window station.
<a href="#">GetClipboardViewer</a>	Retrieves the handle to the first window in the clipboard viewer chain.
<a href="#">GetOpenClipboardWindow</a>	Retrieves the handle to the window that currently has the clipboard open.



TITLE	DESCRIPTION
<a href="#">GetPriorityClipboardFormat</a>	Retrieves the first available clipboard format in the specified list.
<a href="#">GetUpdatedClipboardFormats</a>	Retrieves the currently supported clipboard formats.
<a href="#">GlobalAddAtomA</a>	Adds a character string to the global atom table and returns a unique value (an atom) identifying the string.
<a href="#">GlobalAddAtomExA</a>	Adds a character string to the global atom table and returns a unique value (an atom) identifying the string.
<a href="#">GlobalAddAtomExW</a>	Adds a character string to the global atom table and returns a unique value (an atom) identifying the string.
<a href="#">GlobalAddAtomW</a>	Adds a character string to the global atom table and returns a unique value (an atom) identifying the string.
<a href="#">GlobalDeleteAtom</a>	Decrements the reference count of a global string atom. If the atom's reference count reaches zero, GlobalDeleteAtom removes the string associated with the atom from the global atom table.
<a href="#">GlobalFindAtomA</a>	Searches the global atom table for the specified character string and retrieves the global atom associated with that string.
<a href="#">GlobalFindAtomW</a>	Searches the global atom table for the specified character string and retrieves the global atom associated with that string.
<a href="#">GlobalGetAtomNameA</a>	Retrieves a copy of the character string associated with the specified global atom.
<a href="#">GlobalGetAtomNameW</a>	Retrieves a copy of the character string associated with the specified global atom.
<a href="#">ImpersonateDdeClientWindow</a>	Enables a Dynamic Data Exchange (DDE) server application to impersonate a DDE client application's security context. This protects secure server data from unauthorized DDE clients.
<a href="#">InitAtomTable</a>	Initializes the local atom table and sets the number of hash buckets to the specified size.
<a href="#">IsClipboardFormatAvailable</a>	Determines whether the clipboard contains data in the specified format.
<a href="#">MAKEINTATOM</a>	Converts the specified atom into a string, so it can be passed to functions which accept either atoms or strings.
<a href="#">OpenClipboard</a>	Opens the clipboard for examination and prevents other applications from modifying the clipboard content.

TITLE	DESCRIPTION
<a href="#">PackDDEIParam</a>	Packs a Dynamic Data Exchange (DDE) IParam value into an internal structure used for sharing DDE data between processes.
<a href="#">PFNCALLBACK</a>	An application-defined callback function used with the Dynamic Data Exchange Management Library (DDEML) functions.
<a href="#">RegisterClipboardFormatA</a>	Registers a new clipboard format. This format can then be used as a valid clipboard format.
<a href="#">RegisterClipboardFormatW</a>	Registers a new clipboard format. This format can then be used as a valid clipboard format.
<a href="#">RemoveClipboardFormatListener</a>	Removes the given window from the system-maintained clipboard format listener list.
<a href="#">ReuseDDEIParam</a>	Enables an application to reuse a packed Dynamic Data Exchange (DDE) IParam parameter, rather than allocating a new packed IParam. Using this function reduces reallocations for applications that pass packed DDE messages.
<a href="#">SetClipboardData</a>	Places data on the clipboard in a specified clipboard format.
<a href="#">SetClipboardViewer</a>	Adds the specified window to the chain of clipboard viewers. Clipboard viewer windows receive a WM_DRAWCLIPBOARD message whenever the content of the clipboard changes. This function is used for backward compatibility with earlier versions of Windows.
<a href="#">UnpackDDEIParam</a>	Unpacks a Dynamic Data Exchange (DDE)IParam value received from a posted DDE message.

## Structures

TITLE	DESCRIPTION
<a href="#">CONVCONTEXT</a>	Contains information supplied by a Dynamic Data Exchange (DDE) client application. The information is useful for specialized or cross-language DDE conversations.
<a href="#">CONVINFO</a>	Contains information about a Dynamic Data Exchange (DDE) conversation.
<a href="#">COPYDATASTRUCT</a>	Contains data to be passed to another application by the WM_COPYDATA message.
<a href="#">DDEACK</a>	Contains status flags that a DDE application passes to its partner as part of the WM_DDE_ACK message.
<a href="#">DDEADVISE</a>	Contains flags that specify how a DDE server application should send data to a client application during an advise loop. A client passes a handle to a DDEADVISE structure to a server as part of a WM_DDE_ADVISE message.

TITLE	DESCRIPTION
DDEDATA	Contains the data, and information about the data, sent as part of a WM_DDE_DATA message.
DDEML_MSG_HOOK_DATA	Contains information about a Dynamic Data Exchange (DDE) message, and provides read access to the data referenced by the message. This structure is intended to be used by a Dynamic Data Exchange Management Library (DDEML) monitoring application.
DDEPOKE	Contains the data, and information about the data, sent as part of a WM_DDE_POKE message.
HSZPAIR	Contains a DDE service name and topic name. A DDE server application can use this structure during an XTYP_WILDCONNECT transaction to enumerate the service-topic pairs that it supports.
METAFILEPICT	Defines the metafile picture format used for exchanging metafile data through the clipboard.
MONCBSTRUCT	Contains information about the current Dynamic Data Exchange (DDE) transaction. A DDE debugging application can use this structure when monitoring transactions that the system passes to the DDE callback functions of other applications.
MONCONVSTRUCT	Contains information about a Dynamic Data Exchange (DDE) conversation. A DDE monitoring application can use this structure to obtain information about a conversation that has been established or has terminated.
MONERRSTRUCT	Contains information about the current Dynamic Data Exchange (DDE) error. A DDE monitoring application can use this structure to monitor errors returned by DDE Management Library functions.
MONHSZSTRUCTA	Contains information about a Dynamic Data Exchange (DDE) string handle. A DDE monitoring application can use this structure when monitoring the activity of the string manager component of the DDE Management Library.
MONHSZSTRUCTW	Contains information about a Dynamic Data Exchange (DDE) string handle. A DDE monitoring application can use this structure when monitoring the activity of the string manager component of the DDE Management Library.
MONLINKSTRUCT	Contains information about a Dynamic Data Exchange (DDE) advise loop. A DDE monitoring application can use this structure to obtain information about an advise loop that has started or ended.
MONMSGSTRUCT	Contains information about a Dynamic Data Exchange (DDE) message. A DDE monitoring application can use this structure to obtain information about a DDE message that was sent or posted.

# dde.h header

1/15/2021 • 2 minutes to read • [Edit Online](#)

This header is used by Data Exchange. For more information, see:

- [Data Exchange](#) dde.h contains the following programming interfaces:

## Functions

TITLE	DESCRIPTION
<a href="#">DdeSetQualityOfService</a>	Specifies the quality of service (QOS) a raw Dynamic Data Exchange (DDE) application desires for future DDE conversations it initiates.
<a href="#">FreeDDEIParam</a>	Frees the memory specified by the IParam parameter of a posted Dynamic Data Exchange (DDE) message. An application receiving a posted DDE message should call this function after it has used the UnpackDDEIParam function to unpack the IParam value.
<a href="#">ImpersonateDdeClientWindow</a>	Enables a Dynamic Data Exchange (DDE) server application to impersonate a DDE client application's security context. This protects secure server data from unauthorized DDE clients.
<a href="#">PackDDEIParam</a>	Packs a Dynamic Data Exchange (DDE) IParam value into an internal structure used for sharing DDE data between processes.
<a href="#">ReuseDDEIParam</a>	Enables an application to reuse a packed Dynamic Data Exchange (DDE) IParam parameter, rather than allocating a new packed IParam. Using this function reduces reallocations for applications that pass packed DDE messages.
<a href="#">UnpackDDEIParam</a>	Unpacks a Dynamic Data Exchange (DDE)IParam value received from a posted DDE message.

## Structures

TITLE	DESCRIPTION
<a href="#">DDEACK</a>	Contains status flags that a DDE application passes to its partner as part of the WM_DDE_ACK message.
<a href="#">DDEADVISE</a>	Contains flags that specify how a DDE server application should send data to a client application during an advise loop. A client passes a handle to a DDEADVISE structure to a server as part of a WM_DDE_ADVISE message.
<a href="#">DDEDATA</a>	Contains the data, and information about the data, sent as part of a WM_DDE_DATA message.

TITLE	DESCRIPTION
DDEPOKE	Contains the data, and information about the data, sent as part of a WM_DDE_POKE message.

# DDEACK structure (dde.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Contains status flags that a DDE application passes to its partner as part of the [WM\\_DDE\\_ACK](#) message. The flags provide details about the application's response to the messages [WM\\_DDE\\_DATA](#), [WM\\_DDE\\_POKE](#), [WM\\_DDE\\_EXECUTE](#), [WM\\_DDE\\_ADVISE](#), [WM\\_DDE\\_UNADVISE](#), and [WM\\_DDE\\_REQUEST](#).

## Syntax

```
typedef struct {
    unsigned short bAppReturnCode : 8;
    unsigned short reserved : 6;
    unsigned short fBusy : 1;
    unsigned short fAck : 1;
    unsigned short usFlags;
} DDEACK;
```

## Members

**bAppReturnCode**

Type: **unsigned short**

An application-defined return code.

**reserved**

Type: **unsigned short**

Reserved.

**fBusy**

Type: **unsigned short**

Indicates whether the application was busy and unable to respond to the partner's message at the time the message was received. A nonzero value indicates the partner was busy and unable to respond. The **fBusy** member is defined only when the **fAck** member is zero.

**fAck**

Type: **unsigned short**

Indicates whether the application accepted the message from its partner. A nonzero value indicates the partner accepted the message.

**usFlags**

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]

<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Header</b>	dde.h (include Windows.h)

# See also

[About Dynamic Data Exchange](#)

## Conceptual

## Reference

[WM\\_DDE\\_ACK](#)

[WM\\_DDE\\_ADVISE](#)

[WM\\_DDE\\_DATA](#)

[WM\\_DDE\\_EXECUTE](#)

[WM\\_DDE\\_POKE](#)

[WM\\_DDE\\_REQUEST](#)

[WM\\_DDE\\_UNADVISE](#)

# DDEADVISE structure (dde.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Contains flags that specify how a DDE server application should send data to a client application during an advise loop. A client passes a handle to a **DDEADVISE** structure to a server as part of a [WM\\_DDE\\_ADVISE](#) message.

## Syntax

```
typedef struct {
    unsigned short reserved : 14;
    unsigned short fDeferUpd : 1;
    unsigned short fAckReq : 1;
    unsigned short usFlags;
    short          cfFormat;
} DDEADVISE;
```

## Members

reserved

Type: **unsigned short**

Reserved.

fDeferUpd

Type: **unsigned short**

Indicates whether the server should defer sending updated data to the client. If this value is nonzero, the server should send a [WM\\_DDE\\_DATA](#) message with a **NULL** data handle whenever the data item changes. In response, the client can post a [WM\\_DDE\\_REQUEST](#) message to the server to get a handle to the updated data.

fAckReq

Type: **short**

Indicates whether the server should set the **fAckReq** flag in the [WM\\_DDE\\_DATA](#) messages it posts to the client. If this value is nonzero, the server should set the **fAckReq** bit.

usFlags

cfFormat

Type: **short**

The client application's preferred data format. The format must be a standard or registered clipboard format. The following standard clipboard formats can be used:

**CF\_BITMAP (2)**

**CF\_DIB (8)**

**CF\_DIF (5)**

**CF\_ENHMETAFILE (14)**

**CF\_METAFILEPICT (3)**

**CF\_OEMTEXT (7)**



**CF\_PALETTE** (9)  
**CF\_PENDATA** (10)  
**CF\_RIFF** (11)  
**CF\_SYLK** (4)  
**CF\_TEXT** (1)  
**CF\_TIFF** (6)  
**CF\_WAVE** (12)  
**CF\_UNICODETEXT** (13)

## Requirements

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

## See also

[About Dynamic Data Exchange](#)

### Conceptual

### Reference

[WM\\_DDE\\_ADVISE](#)

[WM\\_DDE\\_DATA](#)

[WM\\_DDE\\_UNADVISE](#)

# DDEDATA structure (dde.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Contains the data, and information about the data, sent as part of a [WM\\_DDE\\_DATA](#) message.

## Syntax

```
typedef struct {
    unsigned short unused : 12;
    unsigned short fResponse : 1;
    unsigned short fRelease : 1;
    unsigned short reserved : 1;
    unsigned short fAckReq : 1;
    unsigned short usFlags;
    short          cfFormat;
    BYTE           Value[1];
} DDEDATA;
```

## Members

unused

Type: **unsigned short**

Unused.

fResponse

Type: **unsigned short**

Indicates whether the data was sent in response to a [WM\\_DDE\\_REQUEST](#) message or a [WM\\_DDE\\_ADVISE](#) message. If this value is nonzero, the data was sent in response to a [WM\\_DDE\\_REQUEST](#) message.

fRelease

Type: **unsigned short**

Indicates whether the application receiving the [WM\\_DDE\\_POKE](#) message should free the data. If this value is nonzero, the application should free the data.

reserved

Type: **unsigned short**

Reserved.

fAckReq

Type: **BYTE**

Indicates whether the application receiving the [WM\\_DDE\\_DATA](#) message should acknowledge receipt of the data by sending a [WM\\_DDE\\_ACK](#) message. If this value is nonzero, the application should send the acknowledgment.

usFlags

cfFormat

Type: `short`

The format of the data. The format should be a standard or registered clipboard format. The following standard clipboard formats can be used:

- `CF_BITMAP` (2)
- `CF_DIB` (8)
- `CF_DIF` (5)
- `CF_ENHMETAFILE` (14)
- `CF_METAFILEPICT` (3)
- `CF_OEMTEXT` (7)
- `CF_PALETTE` (9)
- `CF_PENDATA` (10)
- `CF_RIFF` (11)
- `CF_SYLK` (4)
- `CF_TEXT` (1)
- `CF_TIFF` (6)
- `CF_WAVE` (12)
- `CF_UNICODETEXT` (13)

Value

Type: `BYTE[1]`

Contains the data. The length and type of data depend on the `cfFormat` member.

## Requirements

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

## See also

[About Dynamic Data Exchange](#)

### Conceptual

### Reference

[WM\\_DDE\\_ACK](#)

[WM\\_DDE\\_ADVISE](#)

[WM\\_DDE\\_DATA](#)

[WM\\_DDE\\_POKE](#)

[WM\\_DDE\\_REQUEST](#)

# DDEPOKE structure (dde.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Contains the data, and information about the data, sent as part of a [WM\\_DDE\\_POKE](#) message.

## Syntax

```
typedef struct {
    unsigned short unused : 13;
    unsigned short fRelease : 1;
    unsigned short fReserved : 2;
    unsigned short usFlags;
    short          cfFormat;
    BYTE           Value[1];
} DDEPOKE;
```

## Members

unused

Type: **unsigned short**

Unused.

fRelease

Type: **unsigned short**

Indicates whether the application receiving the [WM\\_DDE\\_POKE](#) message should free the data. If this value is nonzero, the application should free the data.

fReserved

Type: **unsigned short**

Reserved.

usFlags

cfFormat

Type: **short**

The format of the data. The format should be a standard or registered clipboard format. The following standard clipboard formats can be used:

**CF\_BITMAP (2)**

**CF\_DIB (8)**

**CF\_DIF (5)**

**CF\_ENHMETAFILE (14)**

**CF\_METAFILEPICT (3)**

**CF\_OEMTEXT (7)**

**CF\_PALETTE (9)**

**CF\_PENDATA (10)**

**CF\_RIFF (11)**

**CF\_SYLK (4)**

**CF\_TEXT (1)**  
**CF\_TIFF (6)**  
**CF\_WAVE (12)**  
**CF\_UNICODETEXT (13)**

Value

Type: **BYTE[1]**

Contains the data. The length and type of data depend on the value of the **cfFormat** member.

## Requirements

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

## See also

[About Dynamic Data Exchange](#)

### Conceptual

### Reference

[WM\\_DDE\\_POKE](#)

# DdeSetQualityOfService function (dde.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Specifies the quality of service (QOS) a raw Dynamic Data Exchange (DDE) application desires for future DDE conversations it initiates. The specified QOS applies to any conversations started while those settings are in place. A DDE conversation's quality of service lasts for the duration of the conversation; calls to the **DdeSetQualityOfService** function during a conversation do not affect that conversation's QOS.

## Syntax

```
BOOL DdeSetQualityOfService(  
    HWND                hwndClient,  
    const SECURITY_QUALITY_OF_SERVICE *pqosNew,  
    PSECURITY_QUALITY_OF_SERVICE    pqosPrev  
);
```

## Parameters

hwndClient

Type: **HWND**

A handle to the DDE client window that specifies the source of **WM\_DDE\_INITIATE** messages a client will send to start DDE conversations.

pqosNew

Type: **const SECURITY\_QUALITY\_OF\_SERVICE\***

A pointer to a **SECURITY\_QUALITY\_OF\_SERVICE** structure for the desired quality of service values.

pqosPrev

Type: **PSECURITY\_QUALITY\_OF\_SERVICE**

A pointer to a **SECURITY\_QUALITY\_OF\_SERVICE** structure that receives the previous quality of service values associated with the window identified by *hwndClient*.

This parameter is optional. If an application has no interest in *hwndClient*'s previous QOS values, it should set *pqosPrev* to **NULL**.

## 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 quality of service has not been specified for a client window, *hwndClient*, prior to sending a **WM\_DDE\_INITIATE** with the *wParam* set to *hwndClient*, the system uses the following default quality of service values for the client window:

```
{
    Length = sizeof(SEcurity_QUALITY_OF_SERVICE);
    ImpersonationLevel = SecurityImpersonation;
    ContextTrackingMode = SECURITY_STATIC_TRACKING;
    EffectiveOnly = TRUE;
}
```

Use the **DdeSetQualityOfService** function to associate a different quality of service with the client window. After you change the quality of service, the new settings affect any subsequent conversations that are started. Once an application starts a DDE conversation using a particular quality of service value, it must terminate the conversation and restart the conversation in order to have a different value take effect.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	dde.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

## See also

[About Dynamic Data Exchange](#)

**Conceptual**

**Other Resources**

**Reference**

[SECURITY\\_QUALITY\\_OF\\_SERVICE](#)

[WM\\_DDE\\_INITIATE](#)

# FreeDDEIParam function (dde.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Frees the memory specified by the *IParam* parameter of a posted Dynamic Data Exchange (DDE) message. An application receiving a posted DDE message should call this function after it has used the [UnpackDDEIParam](#) function to unpack the *IParam* value.

## Syntax

```
BOOL FreeDDEIParam(  
    UINT    msg,  
    LPARAM  IParam  
);
```

## Parameters

**msg**

Type: **UINT**

The posted DDE message.

**IParam**

Type: **LPARAM**

The *IParam* parameter of the posted DDE message.

## Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

## Remarks

An application should call this function only for posted DDE messages.

This function frees the memory specified by the *IParam* parameter. It does not free the contents of *IParam*.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows



<b>Header</b>	dde.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll
<b>API set</b>	ext-ms-win-ntuser-misc-l1-1-0 (introduced in Windows 8)

# See also

[About Dynamic Data Exchange](#)

## Conceptual

[PackDDEIParam](#)

## Reference

[ReuseDDEIParam](#)

[UnpackDDEIParam](#)

# ImpersonateDdeClientWindow function (dde.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Enables a Dynamic Data Exchange (DDE) server application to impersonate a DDE client application's security context. This protects secure server data from unauthorized DDE clients.

## Syntax

```
BOOL ImpersonateDdeClientWindow(  
    HWND hWndClient,  
    HWND hWndServer  
);
```

## Parameters

`hWndClient`

Type: **HWND**

A handle to the DDE client window to be impersonated. The client window must have established a DDE conversation with the server window identified by the *hWndServer* parameter.

`hWndServer`

Type: **HWND**

A handle to the DDE server window. An application must create the server window 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](#).

## Remarks

An application should call the [RevertToSelf](#) function to undo the impersonation set by the **ImpersonateDdeClientWindow** function.

A DDEML application should use the [DdeImpersonateClient](#) function.

### Security Considerations

Using this function incorrectly might compromise the security of your program. It is very important to check the return value of the call. If the function fails for any reason, the client is not impersonated and any subsequent client request is made in the security context of the calling process. If the calling process is running as a highly privileged account, such as LocalSystem or as a member of an administrative group, the user may be able to perform actions that would otherwise be disallowed. Therefore, if the call fails or raises an error do not continue execution of the client request.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	dde.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

## See also

[About Dynamic Data Exchange](#)

### Conceptual

[DdelpersonateClient](#)

### Other Resources

### Reference

[RevertToSelf](#)

# PackDDEIParam function (dde.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Packs a Dynamic Data Exchange (DDE) *IParam* value into an internal structure used for sharing DDE data between processes.

## Syntax

```
LPARAM PackDDEIParam(  
    UINT      msg,  
    UINT_PTR  uiLo,  
    UINT_PTR  uiHi  
);
```

## Parameters

**msg**

Type: **UINT**

The DDE message to be posted.

**uiLo**

Type: **UINT\_PTR**

A value that corresponds to the 16-bit Windows low-order word of an *IParam* parameter for the DDE message being posted.

**uiHi**

Type: **UINT\_PTR**

A value that corresponds to the 16-bit Windows high-order word of an *IParam* parameter for the DDE message being posted.

## Return value

Type: **LPARAM**

The return value is the *IParam* value.

## Remarks

The return value must be posted as the *IParam* parameter of a DDE message; it must not be used for any other purpose. After the application posts a return value, it need not perform any action to dispose of the *IParam* parameter.

An application should call this function only for posted DDE messages.

## Requirements

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

## See also

[About Dynamic Data Exchange](#)

### Conceptual

[FreeDDEIParam](#)

### Reference

[ReuseDDEIParam](#)

[UnpackDDEIParam](#)

# ReuseDDElParam function (dde.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Enables an application to reuse a packed Dynamic Data Exchange (DDE) *lParam* parameter, rather than allocating a new packed *lParam*. Using this function reduces reallocations for applications that pass packed DDE messages.

## Syntax

```
LPARAM ReuseDDElParam(  
    LPARAM    lParam,  
    UINT      msgIn,  
    UINT      msgOut,  
    UINT_PTR  uiLo,  
    UINT_PTR  uiHi  
);
```

## Parameters

**lParam**

Type: **LPARAM**

The *lParam* parameter of the posted DDE message being reused.

**msgIn**

Type: **UINT**

The identifier of the received DDE message.

**msgOut**

Type: **UINT**

The identifier of the DDE message to be posted. The DDE message will reuse the packed *lParam* parameter.

**uiLo**

Type: **UINT\_PTR**

The value to be packed into the low-order word of the reused *lParam* parameter.

**uiHi**

Type: **UINT\_PTR**

The value to be packed into the high-order word of the reused *lParam* parameter.

## Return value

Type: **LPARAM**

The return value is the new *lParam* value.

## Remarks

The return value must be posted as the *IParam* parameter of a DDE message; it must not be used for any other purpose. Once the return value is posted, the posting application need not perform any action to dispose of the *IParam* parameter.

Use **ReuseDDEIParam** instead of [FreeDDEIParam](#) if the *IParam* parameter will be reused in a responding message. **ReuseDDEIParam** returns the *IParam* appropriate for reuse.

This function allocates or frees *IParam* parameters as needed, depending on the packing requirements of the incoming and outgoing messages. This reduces reallocations in passing DDE messages.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	dde.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

## See also

[About Dynamic Data Exchange](#)

### Conceptual

[FreeDDEIParam](#)

[PackDDEIParam](#)

### Reference

[UnpackDDEIParam](#)

# UnpackDDElParam function (dde.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Unpacks a Dynamic Data Exchange (DDE) *lParam* value received from a posted DDE message.

## Syntax

```
BOOL UnpackDDElParam(  
    UINT      msg,  
    LPARAM    lParam,  
    PUINT_PTR puiLo,  
    PUINT_PTR puiHi  
);
```

## Parameters

`msg`

Type: **UINT**

The posted DDE message.

`lParam`

Type: **LPARAM**

The *lParam* parameter of the posted DDE message that was received. The application must free the memory object specified by the *lParam* parameter by calling the [FreeDDElParam](#) function.

`puiLo`

Type: **PUINT\_PTR**

A pointer to a variable that receives the low-order word of *lParam*.

`puiHi`

Type: **PUINT\_PTR**

A pointer to a variable that receives the high-order word of *lParam*.

## Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

## Remarks

[PackDDElParam](#) eases the porting of 16-bit DDE applications to 32-bit DDE applications.

## Requirements



<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	dde.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll
<b>API set</b>	ext-ms-win-ntuser-misc-l1-1-0 (introduced in Windows 8)

## See also

[About Dynamic Data Exchange](#)

### Conceptual

[FreeDDEIParam](#)

[PackDDEIParam](#)

### Reference

[ReuseDDEIParam](#)

# ddeml.h header

1/15/2021 • 5 minutes to read • [Edit Online](#)

This header is used by Data Exchange. For more information, see:

- [Data Exchange](#) ddeml.h contains the following programming interfaces:

## Functions

TITLE	DESCRIPTION
<a href="#">DdeAbandonTransaction</a>	Abandons the specified asynchronous transaction and releases all resources associated with the transaction.
<a href="#">DdeAccessData</a>	Provides access to the data in the specified Dynamic Data Exchange (DDE) object. An application must call the DdeUnaccessData function when it has finished accessing the data in the object.
<a href="#">DdeAddData</a>	Adds data to the specified Dynamic Data Exchange (DDE) object.
<a href="#">DdeClientTransaction</a>	Begins a data transaction between a client and a server. Only a Dynamic Data Exchange (DDE) client application can call this function, and the application can use it only after establishing a conversation with the server.
<a href="#">DdeCmpStringHandles</a>	Compares the values of two string handles. The value of a string handle is not related to the case of the associated string.
<a href="#">DdeConnect</a>	Establishes a conversation with a server application that supports the specified service name and topic name pair. If more than one such server exists, the system selects only one.
<a href="#">DdeConnectList</a>	Establishes a conversation with all server applications that support the specified service name and topic name pair.
<a href="#">DdeCreateDataHandle</a>	Creates a Dynamic Data Exchange (DDE) object and fills the object with data from the specified buffer. A DDE application uses this function during transactions that involve passing data to the partner application.
<a href="#">DdeCreateStringHandleA</a>	Creates a handle that identifies the specified string. A Dynamic Data Exchange (DDE) client or server application can pass the string handle as a parameter to other Dynamic Data Exchange Management Library (DDEML) functions.
<a href="#">DdeCreateStringHandleW</a>	Creates a handle that identifies the specified string. A Dynamic Data Exchange (DDE) client or server application can pass the string handle as a parameter to other Dynamic Data Exchange Management Library (DDEML) functions.

TITLE	DESCRIPTION
<a href="#">DdeDisconnect</a>	Terminates a conversation started by either the DdeConnect or DdeConnectList function and invalidates the specified conversation handle.
<a href="#">DdeDisconnectList</a>	Destroys the specified conversation list and terminates all conversations associated with the list.
<a href="#">DdeEnableCallback</a>	Enables or disables transactions for a specific conversation or for all conversations currently established by the calling application.
<a href="#">DdeFreeDataHandle</a>	Frees a Dynamic Data Exchange (DDE) object and deletes the data handle associated with the object.
<a href="#">DdeFreeStringHandle</a>	Frees a string handle in the calling application.
<a href="#">DdeGetData</a>	Copies data from the specified Dynamic Data Exchange (DDE) object to the specified local buffer.
<a href="#">DdeGetLastError</a>	Retrieves the most recent error code set by the failure of a Dynamic Data Exchange Management Library (DDEML) function and resets the error code to DMLERR_NO_ERROR.
<a href="#">DdeImpersonateClient</a>	Impersonates a Dynamic Data Exchange (DDE) client application in a DDE client conversation.
<a href="#">DdeInitializeA</a>	Registers an application with the Dynamic Data Exchange Management Library (DDEML). An application must call this function before calling any other Dynamic Data Exchange Management Library (DDEML) function.
<a href="#">DdeInitializeW</a>	Registers an application with the Dynamic Data Exchange Management Library (DDEML). An application must call this function before calling any other Dynamic Data Exchange Management Library (DDEML) function.
<a href="#">DdeKeepStringHandle</a>	Increments the usage count associated with the specified handle.
<a href="#">DdeNameService</a>	Registers or unregisters the service names a Dynamic Data Exchange (DDE) server supports.
<a href="#">DdePostAdvise</a>	Causes the system to send an XTYP_ADVREQ transaction to the calling (server) application's Dynamic Data Exchange (DDE) callback function for each client with an active advise loop on the specified topic and item.
<a href="#">DdeQueryConvInfo</a>	Retrieves information about a Dynamic Data Exchange (DDE) transaction and about the conversation in which the transaction takes place.
<a href="#">DdeQueryNextServer</a>	Retrieves the next conversation handle in the specified conversation list.
<a href="#">DdeQueryStringA</a>	Copies text associated with a string handle into a buffer.

TITLE	DESCRIPTION
<a href="#">DdeQueryStringW</a>	Copies text associated with a string handle into a buffer.
<a href="#">DdeReconnect</a>	Enables a client Dynamic Data Exchange Management Library (DDEML) application to attempt to reestablish a conversation with a service that has terminated a conversation with the client.
<a href="#">DdeSetUserHandle</a>	Associates an application-defined value with a conversation handle or a transaction identifier. This is useful for simplifying the processing of asynchronous transactions. An application can use the <a href="#">DdeQueryConvInfo</a> function to retrieve this value.
<a href="#">DdeUnaccessData</a>	Unaccesses a Dynamic Data Exchange (DDE) object. An application must call this function after it has finished accessing the object.
<a href="#">DdeUninitialize</a>	Frees all Dynamic Data Exchange Management Library (DDEML) resources associated with the calling application.

## Callback functions

TITLE	DESCRIPTION
<a href="#">PFNCALLBACK</a>	An application-defined callback function used with the Dynamic Data Exchange Management Library (DDEML) functions.

## Structures

TITLE	DESCRIPTION
<a href="#">CONVCONTEXT</a>	Contains information supplied by a Dynamic Data Exchange (DDE) client application. The information is useful for specialized or cross-language DDE conversations.
<a href="#">CONVINFO</a>	Contains information about a Dynamic Data Exchange (DDE) conversation.
<a href="#">DDEML_MSG_HOOK_DATA</a>	Contains information about a Dynamic Data Exchange (DDE) message, and provides read access to the data referenced by the message. This structure is intended to be used by a Dynamic Data Exchange Management Library (DDEML) monitoring application.
<a href="#">HSZPAIR</a>	Contains a DDE service name and topic name. A DDE server application can use this structure during an XTYP_WILDCONNECT transaction to enumerate the service-topic pairs that it supports.

TITLE	DESCRIPTION
<a href="#">MONCBSTRUCT</a>	Contains information about the current Dynamic Data Exchange (DDE) transaction. A DDE debugging application can use this structure when monitoring transactions that the system passes to the DDE callback functions of other applications.
<a href="#">MONCONVSTRUCT</a>	Contains information about a Dynamic Data Exchange (DDE) conversation. A DDE monitoring application can use this structure to obtain information about a conversation that has been established or has terminated.
<a href="#">MONERRSTRUCT</a>	Contains information about the current Dynamic Data Exchange (DDE) error. A DDE monitoring application can use this structure to monitor errors returned by DDE Management Library functions.
<a href="#">MONHSZSTRUCTA</a>	Contains information about a Dynamic Data Exchange (DDE) string handle. A DDE monitoring application can use this structure when monitoring the activity of the string manager component of the DDE Management Library.
<a href="#">MONHSZSTRUCTW</a>	Contains information about a Dynamic Data Exchange (DDE) string handle. A DDE monitoring application can use this structure when monitoring the activity of the string manager component of the DDE Management Library.
<a href="#">MONLINKSTRUCT</a>	Contains information about a Dynamic Data Exchange (DDE) advise loop. A DDE monitoring application can use this structure to obtain information about an advise loop that has started or ended.
<a href="#">MONMSGSTRUCT</a>	Contains information about a Dynamic Data Exchange (DDE) message. A DDE monitoring application can use this structure to obtain information about a DDE message that was sent or posted.

# CONVCONTEXT structure (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Contains information supplied by a Dynamic Data Exchange (DDE) client application. The information is useful for specialized or cross-language DDE conversations.

## Syntax

```
typedef struct tagCONVCONTEXT {
    UINT          cb;
    UINT          wFlags;
    UINT          wCountryID;
    int           iCodePage;
    DWORD         dwLangID;
    DWORD         dwSecurity;
    SECURITY_QUALITY_OF_SERVICE qos;
} CONVCONTEXT, *PCONVCONTEXT;
```

## Members

cb

Type: **UINT**

The structure's size, in bytes.

wFlags

Type: **UINT**

The conversation context flags. Currently, no flags are defined for this member.

wCountryID

Type: **UINT**

The country/region code identifier for topic-name and item-name strings.

iCodePage

Type: **int**

The code page for topic-name and item-name strings. Non-multilingual clients should set this member to **CP\_WINANSI**. Unicode clients should set this value to **CP\_WINUNICODE**.

dwLangID

Type: **DWORD**

The [language identifier](#) for topic-name and item-name strings.

dwSecurity

Type: **DWORD**

A private (application-defined) security code.

qos

Type: [SECURITY\\_QUALITY\\_OF\\_SERVICE](#)

The quality of service a DDE client wants from the system during a given conversation. The quality of service level specified lasts for the duration of the conversation. It cannot be changed once the conversation is started.

## Remarks

### Security Warning

For added security, your application can specify a security code with the **dwSecurity** member. The application could then examine this value in the [DdeCallback](#) function to check the identity of the client application. However, a value that is hard-coded into an application might be discovered. Thus, you may want to provide the security code in some other way, such as through user input.

## Requirements

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

## See also

[Dynamic Data Exchange Management Library](#)

# CONVINFO structure (ddeml.h)

1/15/2021 • 4 minutes to read • [Edit Online](#)

Contains information about a Dynamic Data Exchange (DDE) conversation.

## Syntax

```
typedef struct tagCONVINFO {
    DWORD        cb;
    DWORD_PTR    hUser;
    HCONV        hConvPartner;
    HSZ          hszSvcPartner;
    HSZ          hszServiceReq;
    HSZ          hszTopic;
    HSZ          hszItem;
    UINT         wFmt;
    UINT         wType;
    UINT         wStatus;
    UINT         wConvst;
    UINT         wLastError;
    HCONVLIST    hConvList;
    CONVCONTEXT  ConvCtxt;
    HWND         hwnd;
    HWND         hwndPartner;
} CONVINFO, *PCONVINFO;
```

## Members

cb

Type: **DWORD**

The structure's size, in bytes.

hUser

Type: **DWORD\_PTR**

Application-defined data.

hConvPartner

Type: **HCONV**

A handle to the partner application in the DDE conversation. This member is zero if the partner has not registered itself (using the [DdeInitialize](#) function) to make DDEML function calls. An application should not pass this member to any DDEML function except [DdeQueryConvInfo](#).

hszSvcPartner

Type: **HSZ**

A handle to the service name of the partner application.

hszServiceReq

Type: **HSZ**



A handle to the service name of the server application that was requested for connection.

hszTopic

Type: **HSZ**

A handle to the name of the requested topic.

hszItem

Type: **HSZ**

A handle to the name of the requested item. This member is transaction specific.

wFmt

Type: **UINT**

The format of the data being exchanged. This member is transaction specific.

wType

Type: **UINT**

The type of the current transaction. This member is transaction specific; it can be one of the following values.

VALUE	MEANING
<b>XTYP_ADVDATA</b> 0x4010	Notifies a client that advise data from a server has arrived.
<b>XTYP_ADVREQ</b> 0x2022	Requests a server to send updated data to the client during an advise loop. This transaction results when the server calls <a href="#">DdePostAdvise</a> .
<b>XTYP_ADVSTART</b> 0x1030	Requests a server to begin an advise loop with a client.
<b>XTYP_ADVSTOP</b> 0x8040	Notifies a server that an advise loop is stopping.
<b>XTYP_CONNECT</b> 0x1062	Requests a server to establish a conversation with a client.
<b>XTYP_CONNECT_CONFIRM</b> 0x8072	Notifies a server that a conversation with a client has been established.
<b>XTYP_DISCONNECT</b> 0x80C2	Notifies a server that a conversation has terminated.

<b>XTYP_EXECUTE</b> 0x4050	Requests a server to execute a command sent by a client.
<b>XTYP_MONITOR</b> 0x80F2	Notifies an application registered as <b>APPCMD_MONITOR</b> that DDE data is being transmitted.
<b>XTYP_POKE</b> 0x4090	Requests a server to accept unsolicited data from a client.
<b>XTYP_REGISTER</b> 0x80A2	Notifies other DDEML applications that a server has registered a service name.
<b>XTYP_REQUEST</b> 0x20B0	Requests a server to send data to a client.
<b>XTYP_UNREGISTER</b> 0x80D2	Notifies other DDEML applications that a server has unregistered a service name.
<b>XTYP_WILDCONNECT</b> 0x20E2	Requests a server to establish multiple conversations with the same client.
<b>XTYP_XACT_COMPLETE</b> 0x8080	Notifies a client that an asynchronous data transaction has been completed.

**wStatus**

Type: **UINT**

The status of the current conversation. This member can be one or more of the following values.

VALUE	MEANING
<b>ST_ADVISE</b> 0x0002	One or more links are in progress.
<b>ST_BLOCKED</b> 0x0008	The conversation is blocked.
<b>ST_BLOCKNEXT</b> 0x0080	The conversation will block after calling the next callback.

<b>ST_CLIENT</b> 0x0010	The conversation handle passed to the <a href="#">DdeQueryConvInfo</a> function is a client-side handle. If the handle is zero, the conversation handle passed to the <b>DdeQueryConvInfo</b> function is a server-side handle.
<b>ST_CONNECTED</b> 0x0001	The conversation is connected.
<b>ST_INLIST</b> 0x0040	The conversation is a member of a conversation list.
<b>ST_ISLOCAL</b> 0x0004	Both sides of the conversation are using the DDEML.
<b>ST_ISSELF</b> 0x0100	Both sides of the conversation are using the same instance of the DDEML.
<b>ST_TERMINATED</b> 0x0020	The conversation has been terminated by the partner.

wConvst

Type: UINT

The conversation state. This member can be one of the following values.

VALUE	MEANING
<b>XST_ADVACKRCVD</b> 13	The advise transaction has just been completed.
<b>XST_ADVDATAACKRCVD</b> 16	The advise data transaction has just been completed.
<b>XST_ADVDATASENT</b> 15	Advise data has been sent and is awaiting an acknowledgement.
<b>XST_ADVSENT</b> 11	An advise transaction is awaiting an acknowledgement.
<b>XST_CONNECTED</b> 2	The conversation has no active transactions.

<b>XST_DATARCVD</b> 6	The requested data has just been received.
<b>XST_EXECACKRCVD</b> 10	An execute transaction has just been completed.
<b>XST_EXECSENT</b> 9	An execute transaction is awaiting an acknowledgement.
<b>XST_INCOMPLETE</b> 1	The last transaction failed.
<b>XST_INIT1</b> 3	Mid-initiate state 1.
<b>XST_INIT2</b> 4	Mid-initiate state 2.
<b>XST_NULL</b> 0	Pre-initiate state.
<b>XST_POKEACKRCVD</b> 8	A poke transaction has just been completed.
<b>XST_POKESENT</b> 7	A poke transaction is awaiting an acknowledgement.
<b>XST_REQSENT</b> 5	A request transaction is awaiting an acknowledgement.
<b>XST_UNADVACKRCVD</b> 14	An unadvise transaction has just been completed.
<b>XST_UNADVSENT</b> 12	An unadvise transaction is awaiting an acknowledgement.

wLastError

Type: UINT

The error value associated with the last transaction.

hConvList

Type: **HCONVLIST**

A handle to the conversation list if the handle to the current conversation is in a conversation list. This member is **NULL** if the conversation is not in a conversation list.

ConvCtxt

Type: **CONVCONTEXT**

The conversation context.

hwnd

Type: **HWND**

A handle to the window of the calling application involved in the conversation.

hwndPartner

Type: **HWND**

A handle to the window of the partner application involved in the current conversation.

## Requirements

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

## See also

[CONVCONTEXT](#)

### Conceptual

[DdeInitialize](#)

[DdePostAdvise](#)

[DdeQueryConvInfo](#)

[Dynamic Data Exchange Management Library](#)

### Reference

# DdeAbandonTransaction function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Abandons the specified asynchronous transaction and releases all resources associated with the transaction.

## Syntax

```
BOOL DdeAbandonTransaction(  
    DWORD idInst,  
    HCONV hConv,  
    DWORD idTransaction  
);
```

## Parameters

`idInst`

Type: **DWORD**

The application instance identifier obtained by a previous call to the [DdeInitialize](#) function.

`hConv`

Type: **HCONV**

A handle to the conversation in which the transaction was initiated. If this parameter is 0L, all transactions are abandoned (that is, the *idTransaction* parameter is ignored).

`idTransaction`

Type: **DWORD**

The identifier of the transaction to be abandoned. If this parameter is 0L, all active transactions in the specified conversation are abandoned.

## Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Remarks

Only a Dynamic Data Exchange (DDE) client application should call **DdeAbandonTransaction**. If the server application responds to the transaction after the client has called **DdeAbandonTransaction**, the system discards the transaction results. This function has no effect on synchronous transactions.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	ddeml.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

# See also

## Conceptual

[DdeClientTransaction](#)

[DdeInitialize](#)

[DdeQueryConvInfo](#)

[Dynamic Data Exchange Management Library](#)

## Reference

# DdeAccessData function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Provides access to the data in the specified Dynamic Data Exchange (DDE) object. An application must call the [DdeUnaccessData](#) function when it has finished accessing the data in the object.

## Syntax

```
LPBYTE DdeAccessData(  
    HDEDATA hData,  
    LPDWORD pcbDataSize  
);
```

## Parameters

**hData**

Type: **HDEDATA**

A handle to the DDE object to be accessed.

**pcbDataSize**

Type: **LPDWORD**

A pointer to a variable that receives the size, in bytes, of the DDE object identified by the *hData* parameter. If this parameter is **NULL**, no size information is returned.

## Return value

Type: **LPBYTE**

If the function succeeds, the return value is a pointer to the first byte of data in the DDE object.

If the function fails, the return value is **NULL**.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Remarks

If the *hData* parameter has not been passed to a Dynamic Data Exchange Management Library (DDEML) function, an application can use the pointer returned by **DdeAccessData** for read-write access to the DDE object. If *hData* has already been passed to a DDEML function, the pointer should be used only for read access to the memory object.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]



Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	ddeml.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

# See also

## Conceptual

[DdeAddData](#)

[DdeCreateDataHandle](#)

[DdeFreeDataHandle](#)

[DdeUnaccessData](#)

[Dynamic Data Exchange Management Library](#)

## Reference

# DdeAddData function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Adds data to the specified Dynamic Data Exchange (DDE) object. An application can add data starting at any offset from the beginning of the object. If new data overlaps data already in the object, the new data overwrites the old data in the bytes where the overlap occurs. The contents of locations in the object that have not been written to are undefined.

## Syntax

```
HDDEDATA DdeAddData(  
    HDDEDATA hData,  
    LPBYTE pSrc,  
    DWORD cb,  
    DWORD cbOff  
);
```

## Parameters

**hData**

Type: **HDDEDATA**

A handle to the DDE object that receives additional data.

**pSrc**

Type: **LPBYTE**

The data to be added to the DDE object.

**cb**

Type: **DWORD**

The length, in bytes, of the data to be added to the DDE object, including the terminating **NULL**, if the data is a string.

**cbOff**

Type: **DWORD**

An offset, in bytes, from the beginning of the DDE object. The additional data is copied to the object beginning at this offset.

## Return value

Type: **HDDEDATA**

If the function succeeds, the return value is a new handle to the DDE object. The new handle is used in all references to the object.

If the function fails, the return value is zero.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Remarks

After a data handle has been used as a parameter in another [Dynamic Data Exchange Management Library](#) function or has been returned by a DDE callback function, the handle can be used only for read access to the DDE object identified by the handle.

If the amount of memory originally allocated is less than is needed to hold the added data, **DdeAddData** reallocates a global memory object of the appropriate size.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	ddeml.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

## See also

### Conceptual

[DdeAccessData](#)

[DdeCreateDataHandle](#)

[DdeUnaccessData](#)

[Dynamic Data Exchange Management Library](#)

### Reference

# DdeClientTransaction function (ddeml.h)

1/15/2021 • 4 minutes to read • [Edit Online](#)

Begins a data transaction between a client and a server. Only a Dynamic Data Exchange (DDE) client application can call this function, and the application can use it only after establishing a conversation with the server.

## Syntax

```
HDDEDATA DdeClientTransaction(  
    LPBYTE pData,  
    DWORD cbData,  
    HCONV hConv,  
    HSZ hszItem,  
    UINT wFmt,  
    UINT wType,  
    DWORD dwTimeout,  
    LPDWORD pdwResult  
);
```

## Parameters

`pData`

Type: **LPBYTE**

The beginning of the data the client must pass to the server.

Optionally, an application can specify the data handle (**HDDEDATA**) to pass to the server and in that case the *cbData* parameter should be set to -1. This parameter is required only if the *wType* parameter is **XTYP\_EXECUTE** or **XTYP\_POKE**. Otherwise, this parameter should be **NULL**.

For the optional usage of this parameter, **XTYP\_POKE** transactions where *pData* is a data handle, the handle must have been created by a previous call to the **DdeCreateDataHandle** function, employing the same data format specified in the *wFmt* parameter.

`cbData`

Type: **DWORD**

The length, in bytes, of the data pointed to by the *pData* parameter, including the terminating **NULL**, if the data is a string. A value of -1 indicates that *pData* is a data handle that identifies the data being sent.

`hConv`

Type: **HCONV**

A handle to the conversation in which the transaction is to take place.

`hszItem`

Type: **HSZ**

A handle to the data item for which data is being exchanged during the transaction. This handle must have been created by a previous call to the **DdeCreateStringHandle** function. This parameter is ignored (and should be set to 0L) if the *wType* parameter is **XTYP\_EXECUTE**.

wFmt

Type: **UINT**

The standard clipboard format in which the data item is being submitted or requested.

If the transaction specified by the *wType* parameter does not pass data or is [XTYP\\_EXECUTE](#), this parameter should be zero.

If the transaction specified by the *wType* parameter references non-execute DDE data ( [XTYP\\_POKE](#), [XTYP\\_ADVSTART](#), [XTYP\\_ADVSTOP](#), [XTYP\\_REQUEST](#)), the *wFmt* value must be either a valid predefined (CF\_) DDE format or a valid registered clipboard format.

wType

Type: **UINT**

The transaction type. This parameter can be one of the following values.

VALUE	MEANING
<b>XTYP_ADVSTART</b> 0x1030	Begins an advise loop. Any number of distinct advise loops can exist within a conversation. An application can alter the advise loop type by combining the <a href="#">XTYP_ADVSTART</a> transaction type with one or more of the following flags: <ul style="list-style-type: none"><li>• <b>XTYPF_NODATA</b>. Instructs the server to notify the client of any data changes without actually sending the data. This flag gives the client the option of ignoring the notification or requesting the changed data from the server.</li><li>• <b>XTYPF_ACKREQ</b>. Instructs the server to wait until the client acknowledges that it received the previous data item before sending the next data item. This flag prevents a fast server from sending data faster than the client can process it.</li></ul>
<b>XTYP_ADVSTOP</b> 0x8040	Ends an advise loop.
<b>XTYP_EXECUTE</b> 0x4050	Begins an execute transaction.
<b>XTYP_POKE</b> 0x4090	Begins a poke transaction.
<b>XTYP_REQUEST</b> 0x20B0	Begins a request transaction.

dwTimeout

Type: **DWORD**

The maximum amount of time, in milliseconds, that the client will wait for a response from the server application in a synchronous transaction. This parameter should be **TIMEOUT\_ASYNC** for asynchronous transactions.

pdwResult

Type: LPDWORD

A pointer to a variable that receives the result of the transaction. An application that does not check the result can use **NULL** for this value. For synchronous transactions, the low-order word of this variable contains any applicable DDE\_ flags resulting from the transaction. This provides support for applications dependent on DDE\_APPSTATUS bits. It is, however, recommended that applications no longer use these bits because they may not be supported in future versions of the [Dynamic Data Exchange Management Library](#) (DDEML). For asynchronous transactions, this variable is filled with a unique transaction identifier for use with the [DdeAbandonTransaction](#) function and the XTYP\_XACT\_COMPLETE transaction.

## Return value

Type: HDEDATA

If the function succeeds, the return value is a data handle that identifies the data for successful synchronous transactions in which the client expects data from the server. The return value is nonzero for successful asynchronous transactions and for synchronous transactions in which the client does not expect data. The return value is zero for all unsuccessful transactions.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Remarks

When an application has finished using the data handle returned by **DdeClientTransaction**, the application should free the handle by calling the [DdeFreeDataHandle](#) function.

Transactions can be synchronous or asynchronous. During a synchronous transaction, **DdeClientTransaction** does not return until the transaction either completes successfully or fails. Synchronous transactions cause a client to enter a modal loop while waiting for various asynchronous events. Because of this, a client application can still respond to user input while waiting on a synchronous transaction, but the application cannot begin a second synchronous transaction because of the activity associated with the first. **DdeClientTransaction** fails if any instance of the same task has a synchronous transaction already in progress.

During an asynchronous transaction, **DdeClientTransaction** returns after the transaction has begun, passing a transaction identifier for reference. When the server's DDE callback function finishes processing an asynchronous transaction, the system sends an XTYP\_XACT\_COMPLETE transaction to the client. This transaction provides the client with the results of the asynchronous transaction that it initiated by calling **DdeClientTransaction**. A client application can choose to abandon an asynchronous transaction by calling the [DdeAbandonTransaction](#) function.

## Requirements

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

<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

# See also

## Conceptual

[DdeAbandonTransaction](#)

[DdeAccessData](#)

[DdeConnect](#)

[DdeConnectList](#)

[DdeCreateDataHandle](#)

[DdeCreateStringHandle](#)

[DdeFreeDataHandle](#)

[Dynamic Data Exchange Management Library](#)

## Reference

[XTYP\\_ADVSTART](#)

[XTYP\\_ADVSTOP](#)

[XTYP\\_EXECUTE](#)

[XTYP\\_POKE](#)

[XTYP\\_REQUEST](#)

# DdeCmpStringHandles function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Compares the values of two string handles. The value of a string handle is not related to the case of the associated string.

## Syntax

```
int DdeCmpStringHandles(  
    HSZ hsz1,  
    HSZ hsz2  
);
```

## Parameters

**hsz1**

Type: **HSZ**

A handle to the first string.

**hsz2**

Type: **HSZ**

A handle to the second string.

## Return value

Type: **int**

The return value can be one of the following values.

RETURN VALUE	DESCRIPTION
-1	The value of <i>hsz1</i> is either 0 or less than the value of <i>hsz2</i> .
0	The values of <i>hsz1</i> and <i>hsz2</i> are equal (both can be 0).
1	The value of <i>hsz2</i> is either 0 or less than the value of <i>hsz1</i> .

## Remarks

An application that must do a case-sensitive comparison of two string handles should compare the string handles directly. An application should use **DdeCmpStringHandles** for all other comparisons to preserve the case-insensitive nature of Dynamic Data Exchange (DDE).

**DdeCmpStringHandles** cannot be used to sort string handles alphabetically.



# Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	ddeml.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

## See also

### Conceptual

[DdeAccessData](#)

[DdeCreateStringHandle](#)

[DdeFreeStringHandle](#)

[Dynamic Data Exchange Management Library](#)

### Reference

# DdeConnect function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Establishes a conversation with a server application that supports the specified service name and topic name pair. If more than one such server exists, the system selects only one.

## Syntax

```
HCONV DdeConnect(  
    DWORD      idInst,  
    HSZ        hszService,  
    HSZ        hszTopic,  
    PCONVCONTEXT pCC  
);
```

## Parameters

**idInst**

Type: **DWORD**

The application instance identifier obtained by a previous call to the [DdeInitialize](#) function.

**hszService**

Type: **HSZ**

A handle to the string that specifies the service name of the server application with which a conversation is to be established. This handle must have been created by a previous call to the [DdeCreateStringHandle](#) function. If this parameter is 0L, a conversation is established with any available server.

**hszTopic**

Type: **HSZ**

A handle to the string that specifies the name of the topic on which a conversation is to be established. This handle must have been created by a previous call to [DdeCreateStringHandle](#). If this parameter is 0L, a conversation on any topic supported by the selected server is established.

**pCC**

Type: **PCONVCONTEXT**

A pointer to the [CONVCONTEXT](#) structure that contains conversation context information. If this parameter is **NULL**, the server receives the default **CONVCONTEXT** structure during the [XTYP\\_CONNECT](#) or [XTYP\\_WILDCONNECT](#) transaction.

## Return value

Type: **HCONV**

If the function succeeds, the return value is the handle to the established conversation.

If the function fails, the return value is 0L.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Remarks

The client application cannot make assumptions regarding the server selected. If an instance-specific name is specified in the *hszService* parameter, a conversation is established with only the specified instance. Instance-specific service names are passed to an application's Dynamic Data Exchange (DDE) callback function during the [XTYP\\_REGISTER](#) and [XTYP\\_UNREGISTER](#) transactions.

All members of the default [CONVCONTEXT](#) structure are set to zero except *cb*, which specifies the size of the structure, and *iCodePage*, which specifies [CP\\_WINANSI](#) (the default code page) or [CP\\_WINUNICODE](#), depending on whether the ANSI or Unicode version of the [DdeInitialize](#) function was called by the client application.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	ddeml.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

## See also

[CONVCONTEXT](#)

### Conceptual

[DdeConnectList](#)

[DdeCreateStringHandle](#)

[DdeDisconnect](#)

[DdeDisconnectList](#)

[DdeInitialize](#)

[Dynamic Data Exchange Management Library](#)

### Reference

[XTYP\\_REGISTER](#)

[XTYP\\_UNREGISTER](#)

# DdeConnectList function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Establishes a conversation with all server applications that support the specified service name and topic name pair. An application can also use this function to obtain a list of conversation handles by passing the function an existing conversation handle. The [Dynamic Data Exchange Management Library](#) removes the handles of any terminated conversations from the conversation list. The resulting conversation list contains the handles of all currently established conversations that support the specified service name and topic name.

## Syntax

```
HCONVLIST DdeConnectList(  
    DWORD          idInst,  
    HSZ            hszService,  
    HSZ            hszTopic,  
    HCONVLIST      hConvList,  
    PCONVCONTEXT   pCC  
);
```

## Parameters

idInst

Type: **DWORD**

The application instance identifier obtained by a previous call to the [DdeInitialize](#) function.

hszService

Type: **HSZ**

A handle to the string that specifies the service name of the server application with which a conversation is to be established. If this parameter is 0L, the system attempts to establish conversations with all available servers that support the specified topic name.

hszTopic

Type: **HSZ**

A handle to the string that specifies the name of the topic on which a conversation is to be established. This handle must have been created by a previous call to the [DdeCreateStringHandle](#) function. If this parameter is 0L, the system will attempt to establish conversations on all topics supported by the selected server (or servers).

hConvList

Type: **HCONVLIST**

A handle to the conversation list to be enumerated. This parameter should be 0L if a new conversation list is to be established.

pCC

Type: **PCONVCONTEXT**

A pointer to the [CONVCONTEXT](#) structure that contains conversation-context information. If this parameter is

**NULL**, the server receives the default **CONVCONTEXT** structure during the [XTYP\\_CONNECT](#) or [XTYP\\_WILDCONNECT](#) transaction.

## Return value

Type: **HCONVLIST**

If the function succeeds, the return value is the handle to a new conversation list.

If the function fails, the return value is 0L. The handle to the old conversation list is no longer valid.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Remarks

An application must free the conversation list handle returned by the **DdeConnectList** function, regardless of whether any conversation handles within the list are active. To free the handle, an application can call [DdeDisconnectList](#).

All members of the default [CONVCONTEXT](#) structure are set to zero except *cb*, specifying the size of the structure, and *iCodePage*, specifying **CP\_WINANSI** (the default code page) or **CP\_WINUNICODE**, depending on whether the ANSI or Unicode version of the [DdeInitialize](#) function was called by the client application.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	ddeml.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

## See also

[CONVCONTEXT](#)

### Conceptual

[DdeConnect](#)

[DdeCreateStringHandle](#)

[DdeDisconnect](#)

[DdeDisconnectList](#)

[DdeInitialize](#)

[DdeQueryNextServer](#)

## Reference

# DdeCreateDataHandle function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Creates a Dynamic Data Exchange (DDE) object and fills the object with data from the specified buffer. A DDE application uses this function during transactions that involve passing data to the partner application.

## Syntax

```
HDDEDATA DdeCreateDataHandle(  
    DWORD    idInst,  
    LPBYTE   pSrc,  
    DWORD    cb,  
    DWORD    cbOff,  
    HSZ      hszItem,  
    UINT     wFmt,  
    UINT     afCmd  
);
```

## Parameters

`idInst`

Type: **DWORD**

The application instance identifier obtained by a previous call to the [DdeInitialize](#) function.

`pSrc`

Type: **LPBYTE**

The data to be copied to the DDE object. If this parameter is **NULL**, no data is copied to the object.

`cb`

Type: **DWORD**

The amount of memory, in bytes, to copy from the buffer pointed to by *pSrc*. (include the terminating NULL, if the data is a string). If this parameter is zero, the *pSrc* parameter is ignored.

`cbOff`

Type: **DWORD**

An offset, in bytes, from the beginning of the buffer pointed to by the *pSrc* parameter. The data beginning at this offset is copied from the buffer to the DDE object.

`hszItem`

Type: **HSZ**

A handle to the string that specifies the data item corresponding to the DDE object. This handle must have been created by a previous call to the [DdeCreateStringHandle](#) function. If the data handle is to be used in an [XTYP\\_EXECUTE](#) transaction, this parameter must be 0L.

`wFmt`

Type: **UINT**

The standard clipboard format of the data.

afCmd

Type: **UINT**

The creation flags. This parameter can be **HDATA\_APPOWNED**, which specifies that the server application calling the **DdeCreateDataHandle** function owns the data handle this function creates. This flag enables the application to share the data handle with other DDEML applications rather than creating a separate handle to pass to each application. If this flag is specified, the application must eventually free the shared memory object associated with the handle by using the [DdeFreeDataHandle](#) function. If this flag is not specified, the handle becomes invalid in the application that created the handle after the data handle is returned by the application's DDE callback function or is used as a parameter in another DDEML function.

## Return value

Type: **HDDEDATA**

If the function succeeds, the return value is a data handle.

If the function fails, the return value is 0L.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Remarks

Any unfilled locations in the DDE object are undefined.

After a data handle has been used as a parameter in another DDEML function or has been returned by a DDE callback function, the handle can be used only for read access to the DDE object identified by the handle.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	ddeml.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

## See also

### Conceptual

[DdeAccessData](#)

[DdeCreateStringHandle](#)



[DdeFreeDataHandle](#)

[DdeGetData](#)

[DdeInitialize](#)

[Dynamic Data Exchange Management Library](#)

**Reference**

# DdeCreateStringHandleA function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Creates a handle that identifies the specified string. A Dynamic Data Exchange (DDE) client or server application can pass the string handle as a parameter to other Dynamic Data Exchange Management Library (DDEML) functions.

## Syntax

```
HSZ DdeCreateStringHandleA(  
    DWORD   idInst,  
    LPCSTR  psz,  
    int     iCodePage  
);
```

## Parameters

`idInst`

Type: **DWORD**

The application instance identifier obtained by a previous call to the [DdeInitialize](#) function.

`psz`

Type: **LPTSTR**

The null-terminated string for which a handle is to be created. This string can be up to 255 characters. The reason for this limit is that DDEML string management functions are implemented using atoms.

`iCodePage`

Type: **int**

The code page to be used to render the string. This value should be either **CP\_WINANSI** (the default code page) or **CP\_WINUNICODE**, depending on whether the ANSI or Unicode version of [DdeInitialize](#) was called by the client application.

## Return value

Type: **HSZ**

If the function succeeds, the return value is a string handle.

If the function fails, the return value is 0L.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Remarks

The value of a string handle is not related to the case of the string it identifies.

When an application either creates a string handle or receives one in the callback function and then uses the [DdeKeepStringHandle](#) function to keep it, the application must free that string handle when it is no longer

needed.

An instance-specific string handle cannot be mapped from string handle to string and back to string handle. This is shown in the following example, in which the [DdeQueryString](#) function creates a string from a string handle and [DdeCreateStringHandle](#) creates a string handle from that string, but the two handles are not the same:

```
DWORD idInst;
DWORD cb;
HSZ hszInst, hszNew;
PSZ pszInst;

DdeQueryString(idInst, hszInst, pszInst, cb, CP_WINANSI);
hszNew = DdeCreateStringHandle(idInst, pszInst, CP_WINANSI);
// hszNew != hszInst !
```

**NOTE**

The `ddeml.h` header defines `DdeCreateStringHandle` 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

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	ddeml.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

# See also

Conceptual

[DdeAccessData](#)

[DdeCmpStringHandles](#)

[DdeFreeStringHandle](#)

[DdeInitialize](#)

[DdeKeepStringHandle](#)

[DdeQueryString](#)

[Dynamic Data Exchange Management Library](#)

## Reference

# DdeCreateStringHandleW function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Creates a handle that identifies the specified string. A Dynamic Data Exchange (DDE) client or server application can pass the string handle as a parameter to other Dynamic Data Exchange Management Library (DDEML) functions.

## Syntax

```
HSZ DdeCreateStringHandleW(  
    DWORD    idInst,  
    LPCWSTR  psz,  
    int      iCodePage  
);
```

## Parameters

`idInst`

Type: **DWORD**

The application instance identifier obtained by a previous call to the [DdeInitialize](#) function.

`psz`

Type: **LPTSTR**

The null-terminated string for which a handle is to be created. This string can be up to 255 characters. The reason for this limit is that DDEML string management functions are implemented using atoms.

`iCodePage`

Type: **int**

The code page to be used to render the string. This value should be either **CP\_WINANSI** (the default code page) or **CP\_WINUNICODE**, depending on whether the ANSI or Unicode version of [DdeInitialize](#) was called by the client application.

## Return value

Type: **HSZ**

If the function succeeds, the return value is a string handle.

If the function fails, the return value is 0L.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Remarks

The value of a string handle is not related to the case of the string it identifies.

When an application either creates a string handle or receives one in the callback function and then uses the [DdeKeepStringHandle](#) function to keep it, the application must free that string handle when it is no longer

needed.

An instance-specific string handle cannot be mapped from string handle to string and back to string handle. This is shown in the following example, in which the [DdeQueryString](#) function creates a string from a string handle and [DdeCreateStringHandle](#) creates a string handle from that string, but the two handles are not the same:

```
DWORD idInst;
DWORD cb;
HSZ hszInst, hszNew;
PSZ pszInst;

DdeQueryString(idInst, hszInst, pszInst, cb, CP_WINANSI);
hszNew = DdeCreateStringHandle(idInst, pszInst, CP_WINANSI);
// hszNew != hszInst !
```

**NOTE**

The `ddeml.h` header defines `DdeCreateStringHandle` 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

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	ddeml.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

# See also

Conceptual

[DdeAccessData](#)

[DdeCmpStringHandles](#)

[DdeFreeStringHandle](#)

[DdeInitialize](#)

[DdeKeepStringHandle](#)

[DdeQueryString](#)

[Dynamic Data Exchange Management Library](#)

## Reference

# DdeDisconnect function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Terminates a conversation started by either the [DdeConnect](#) or [DdeConnectList](#) function and invalidates the specified conversation handle.

## Syntax

```
BOOL DdeDisconnect(  
    HCONV hConv  
);
```

## Parameters

**hConv**

Type: **HCONV**

A handle to the active conversation to be terminated.

## Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Remarks

Any incomplete transactions started before calling **DdeDisconnect** are immediately abandoned. The [XTYP\\_DISCONNECT](#) transaction is sent to the Dynamic Data Exchange (DDE) callback function of the partner in the conversation. Generally, only client applications must terminate conversations.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	ddeml.h (include Windows.h)
<b>Library</b>	User32.lib



DLL	User32.dll

# See also

## Conceptual

[DdeConnect](#)

[DdeConnectList](#)

[DdeDisconnectList](#)

[Dynamic Data Exchange Management Library](#)

## Reference

[XTYP\\_DISCONNECT](#)

# DdeDisconnectList function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Destroys the specified conversation list and terminates all conversations associated with the list.

## Syntax

```
BOOL DdeDisconnectList(  
    HCONVLIST hConvList  
);
```

## Parameters

`hConvList`

Type: **HCONVLIST**

A handle to the conversation list. This handle must have been created by a previous call to the [DdeConnectList](#) function.

## Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Remarks

An application can use the [DdeDisconnect](#) function to terminate individual conversations in the list.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	ddeml.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

## See also

### Conceptual

[DdeConnect](#)

[DdeConnectList](#)

[DdeDisconnect](#)

[Dynamic Data Exchange Management Library](#)

### Reference

# DdeEnableCallback function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Enables or disables transactions for a specific conversation or for all conversations currently established by the calling application.

## Syntax

```
BOOL DdeEnableCallback(  
    DWORD idInst,  
    HCONV hConv,  
    UINT wCmd  
);
```

## Parameters

idInst

Type: **DWORD**

The application-instance identifier obtained by a previous call to the [DdeInitialize](#) function.

hConv

Type: **HCONV**

A handle to the conversation to enable or disable. If this parameter is **NULL**, the function affects all conversations.

wCmd

Type: **UINT**

The function code. This parameter can be one of the following values.

VALUE	MEANING
<b>EC_ENABLEALL</b> 0	Enables all transactions for the specified conversation.
<b>EC_ENABLEONE</b> 0x0080	Enables one transaction for the specified conversation.

<b>EC_DISABLE</b> 0x0008	Disables all blockable transactions for the specified conversation. A server application can disable the following transactions: <ul style="list-style-type: none"> <li>• <a href="#">XTYP_ADVSTART</a></li> <li>• <a href="#">XTYP_ADVSTOP</a></li> <li>• <a href="#">XTYP_EXECUTE</a></li> <li>• <a href="#">XTYP_POKE</a></li> <li>• <a href="#">XTYP_REQUEST</a></li> </ul> A client application can disable the following transactions: <ul style="list-style-type: none"> <li>• <a href="#">XTYP_ADVDATA</a></li> <li>• <a href="#">XTYP_XACT_COMPLETE</a></li> </ul>
<b>EC_QUERYWAITING</b> 2	Determines whether any transactions are in the queue for the specified conversation.

## Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

If the *wCmd* parameter is **EC\_QUERYWAITING**, and the application transaction queue contains one or more unprocessed transactions that are not being processed, the return value is **TRUE**; otherwise, it is **FALSE**.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Remarks

An application can disable transactions for a specific conversation by returning the **CBR\_BLOCK** return code from its Dynamic Data Exchange (DDE) callback function. When you reenable the conversation by using the **DdeEnableCallback** function, the operating system generates the same transaction that was in process when the conversation was disabled.

Using the **EC\_QUERYWAITING** flag does not change the enable state of the conversation and does not cause transactions to be issued within the context of the call to **DdeEnableCallback**.

If **DdeEnableCallback** is called with **EC\_QUERYWAITING** and the function returns a nonzero, an application should try to quickly allow message processing, return from its callback, or enable callbacks. Such a result does not guarantee that subsequent callbacks will be made. Calling **DdeEnableCallback** with **EC\_QUERYWAITING** lets an application with blocked callbacks determine whether there are any transactions pending on the blocked conversation. Of course, even if such a call returns zero, an application should always process messages in a timely manner.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]

<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	ddeml.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

# See also

## Conceptual

[DdeConnect](#)

[DdeConnectList](#)

[DdeDisconnect](#)

[DdelInitialize](#)

[Dynamic Data Exchange Management Library](#)

## Reference

# DdeFreeDataHandle function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Frees a Dynamic Data Exchange (DDE) object and deletes the data handle associated with the object.

## Syntax

```
BOOL DdeFreeDataHandle(  
    HDEADATA hData  
);
```

## Parameters

hData

Type: **HDEADATA**

A handle to the DDE object to be freed. This handle must have been created by a previous call to the [DdeCreateDataHandle](#) function or returned by the [DdeClientTransaction](#) function.

## Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Remarks

An application must call **DdeFreeDataHandle** under the following circumstances:

- To free a DDE object that the application allocated by calling the [DdeCreateDataHandle](#) function if the object's data handle was never passed by the application to another Dynamic Data Exchange Management Library (DDEML) function
- To free a DDE object that the application allocated by specifying the **HDATA\_APPOWNED** flag in a call to [DdeCreateDataHandle](#)
- To free a DDE object whose handle the application received from the [DdeClientTransaction](#) function

The system automatically frees an unowned object when its handle is returned by a DDE callback function or is used as a parameter in a DDEML function.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	ddeml.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

# See also

## Conceptual

[DdeAccessData](#)

[DdeClientTransaction](#)

[DdeCreateDataHandle](#)

[Dynamic Data Exchange Management Library](#)

## Reference



# DdeFreeStringHandle function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Frees a string handle in the calling application.

## Syntax

```
BOOL DdeFreeStringHandle(  
    DWORD idInst,  
    HSZ    hsz  
);
```

## Parameters

idInst

Type: **DWORD**

The application instance identifier obtained by a previous call to the [DdeInitialize](#) function.

hsz

Type: **HSZ**

A handle to the string handle to be freed. This handle must have been created by a previous call to the [DdeCreateStringHandle](#) function.

## Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

## Remarks

An application can free string handles it creates with [DdeCreateStringHandle](#) but should not free those that the system passed to the application's Dynamic Data Exchange (DDE) callback function or those returned in the [CONVINFO](#) structure by the [DdeQueryConvInfo](#) function.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows

<b>Header</b>	ddeml.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

# See also

[CONVINFO](#)

## Conceptual

[DdeCmpStringHandles](#)

[DdeCreateStringHandle](#)

[DdeInitialize](#)

[DdeKeepStringHandle](#)

[DdeQueryConvInfo](#)

[DdeQueryString](#)

[Dynamic Data Exchange Management Library](#)

## Reference

# DdeGetData function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Copies data from the specified Dynamic Data Exchange (DDE) object to the specified local buffer.

## Syntax

```
DWORD DdeGetData(  
    HDEDEDATA hData,  
    LPBYTE pDst,  
    DWORD cbMax,  
    DWORD cbOff  
);
```

## Parameters

**hData**

Type: **HDEDEDATA**

A handle to the DDE object that contains the data to copy.

**pDst**

Type: **LPBYTE**

A pointer to the buffer that receives the data. If this parameter is **NULL**, the **DdeGetData** function returns the amount of data, in bytes, that would be copied to the buffer.

**cbMax**

Type: **DWORD**

The maximum amount of data, in bytes, to copy to the buffer pointed to by the *pDst* parameter. Typically, this parameter specifies the length of the buffer pointed to by *pDst*.

**cbOff**

Type: **DWORD**

An offset within the DDE object. Data is copied from the object beginning at this offset.

## Return value

Type: **DWORD**

If the *pDst* parameter points to a buffer, the return value is the size, in bytes, of the memory object associated with the data handle or the size specified in the *cbMax* parameter, whichever is lower.

If the *pDst* parameter is **NULL**, the return value is the size, in bytes, of the memory object associated with the data handle.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

# Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	ddeml.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

## See also

### Conceptual

[DdeAccessData](#)

[DdeCreateDataHandle](#)

[DdeFreeDataHandle](#)

[Dynamic Data Exchange Management Library](#)

### Reference

# DdeGetLastError function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Retrieves the most recent error code set by the failure of a Dynamic Data Exchange Management Library (DDEML) function and resets the error code to DMLERR\_NO\_ERROR.

## Syntax

```
UINT DdeGetLastError(  
    DWORD idInst  
);
```

## Parameters

idInst

Type: **DWORD**

The application instance identifier obtained by a previous call to the [DdeInitialize](#) function.

## Return value

Type: **UINT**

If the function succeeds, the return value is the last error code, which can be one of the following values.

RETURN CODE/VALUE	DESCRIPTION
<b>DMLERR_ADVACKTIMEOUT</b> 0x4000	A request for a synchronous advise transaction has timed out.
<b>DMLERR_BUSY</b> 0x4001	The response to the transaction caused the <b>DDE_FBUSY</b> flag to be set.
<b>DMLERR_DATAACKTIMEOUT</b> 0x4002	A request for a synchronous data transaction has timed out.
<b>DMLERR_DLL_NOT_INITIALIZED</b> 0x4003	A DDEML function was called without first calling the <a href="#">DdeInitialize</a> function, or an invalid instance identifier was passed to a DDEML function.
<b>DMLERR_DLL_USAGE</b> 0x4004	An application initialized as <b>APPCLASS_MONITOR</b> has attempted to perform a DDE transaction, or an application initialized as <b>APPCMD_CLIENTONLY</b> has attempted to perform server transactions.

<b>DMLERR_EXECACKTIMEOUT</b> 0x4005	A request for a synchronous execute transaction has timed out.
<b>DMLERR_INVALIDPARAMETER</b> 0x4006	<p>A parameter failed to be validated by the DDEML. Some of the possible causes follow:</p> <p>The application used a data handle initialized with a different item name handle than was required by the transaction.</p> <p>The application used a data handle that was initialized with a different clipboard data format than was required by the transaction.</p> <p>The application used a client-side conversation handle with a server-side function or vice versa.</p> <p>The application used a freed data handle or string handle.</p> <p>More than one instance of the application used the same object.</p>
<b>DMLERR_LOW_MEMORY</b> 0x4007	A DDEML application has created a prolonged race condition (in which the server application outruns the client), causing large amounts of memory to be consumed.
<b>DMLERR_MEMORY_ERROR</b> 0x4008	A memory allocation has failed.
<b>DMLERR_NO_CONV_ESTABLISHED</b> 0x400a	A client's attempt to establish a conversation has failed.
<b>DMLERR_NOTPROCESSED</b> 0x4009	A transaction has failed.
<b>DMLERR_POKEACKTIMEOUT</b> 0x400b	A request for a synchronous poke transaction has timed out.
<b>DMLERR_POSTMSG_FAILED</b> 0x400c	An internal call to the <a href="#">PostMessage</a> function has failed.
<b>DMLERR_REENTRANCY</b> 0x400d	An application instance with a synchronous transaction already in progress attempted to initiate another synchronous transaction, or the <a href="#">DdeEnableCallback</a> function was called from within a DDEML callback function.
<b>DMLERR_SERVER_DIED</b> 0x400e	A server-side transaction was attempted on a conversation terminated by the client, or the server terminated before completing a transaction.

<b>DMLERR_SYS_ERROR</b> 0x400f	An internal error has occurred in the DDEML.
<b>DMLERR_UNADVACKTIMEOUT</b> 0x4010	A request to end an advise transaction has timed out.
<b>DMLERR_UNFOUND_QUEUE_ID</b> 0x4011	An invalid transaction identifier was passed to a DDEML function. Once the application has returned from an <a href="#">XTYP_XACT_COMPLETE</a> callback, the transaction identifier for that callback function is no longer valid.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	ddeml.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

## See also

### Conceptual

[DdeEnableCallback](#)

[DdeInitialize](#)

[Dynamic Data Exchange Management Library](#)

[PostMessage](#)

### Reference

# DdeImpersonateClient function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Impersonates a Dynamic Data Exchange (DDE) client application in a DDE client conversation.

## Syntax

```
BOOL DdeImpersonateClient(  
    HCONV hConv  
);
```

## Parameters

hConv

Type: **HCONV**

A handle to the DDE client conversation to be impersonated.

## 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

Impersonation is the ability of a process to take on the security attributes of another process. When a client in a DDE conversation requests information from a DDE server, the server impersonates the client. When the server requests access to an object, the system verifies the access against the client's security attributes.

When the impersonation is complete, the server normally calls the [RevertToSelf](#) function.

### Security Considerations

If the call to **DdeImpersonateClient** fails for any reason, the client is not impersonated and the client request is made in the security context of the calling process. If the calling process is running as a highly privileged account, such as LocalSystem, or as a member of an administrative group, the user may be able to perform actions that would otherwise be disallowed. Therefore it is important that you always check the return value of the call, and if it fails to raise an error, do not continue execution of the client request.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]



<b>Target Platform</b>	Windows
<b>Header</b>	ddeml.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

# See also

## Conceptual

[Dynamic Data Exchange Management Library](#)

[ImpersonateNamedPipeClient](#)

## Other Resources

[RevertToSelf](#)

# DdeInitializeA function (ddeml.h)

1/15/2021 • 5 minutes to read • [Edit Online](#)

Registers an application with the [Dynamic Data Exchange Management Library](#) (DDEML). An application must call this function before calling any other Dynamic Data Exchange Management Library (DDEML) function.

## Syntax

```
UINT DdeInitializeA(  
    LPDWORD    pidInst,  
    PFNCALLBACK pfnCallback,  
    DWORD       afCmd,  
    DWORD       ulRes  
);
```

## Parameters

`pidInst`

Type: **LPDWORD**

The application instance identifier. At initialization, this parameter should point to 0. If the function succeeds, this parameter points to the instance identifier for the application. This value should be passed as the *idInst* parameter in all other DDEML functions that require it. If an application uses multiple instances of the DDEML dynamic-link library (DLL), the application should provide a different callback function for each instance.

If *pidInst* points to a nonzero value, reinitialization of the DDEML is implied. In this case, *pidInst* must point to a valid application-instance identifier.

`pfnCallback`

Type: **PFNCALLBACK**

A pointer to the application-defined DDE callback function. This function processes DDE transactions sent by the system. For more information, see the [DdeCallback](#) callback function.

`afCmd`

Type: **DWORD**

A set of **APPCMD\_**, **CBF\_**, and **MF\_** flags. The **APPCMD\_** flags provide special instructions to **DdeInitialize**. The **CBF\_** flags specify filters that prevent specific types of transactions from reaching the callback function. The **MF\_** flags specify the types of DDE activity that a DDE monitoring application monitors. Using these flags enhances the performance of a DDE application by eliminating unnecessary calls to the callback function.

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

VALUE	MEANING
-------	---------

<b>APPCLASS_MONITOR</b> 0x00000001L	<p>Makes it possible for the application to monitor DDE activity in the system. This flag is for use by DDE monitoring applications. The application specifies the types of DDE activity to monitor by combining one or more monitor flags with the <b>APPCLASS_MONITOR</b> flag. For details, see the following Remarks section.</p>
<b>APPCLASS_STANDARD</b> 0x00000000L	<p>Registers the application as a standard (nonmonitoring) DDEML application.</p>
<b>APPCMD_CLIENTONLY</b> 0x00000010L	<p>Prevents the application from becoming a server in a DDE conversation. The application can only be a client. This flag reduces consumption of resources by the DDEML. It includes the functionality of the <b>CBF_FAIL_ALLSVRXACTIONS</b> flag.</p>
<b>APPCMD_FILTERINITS</b> 0x00000020L	<p>Prevents the DDEML from sending <b>XTYP_CONNECT</b> and <b>XTYP_WILDCONNECT</b> transactions to the application until the application has created its string handles and registered its service names or has turned off filtering by a subsequent call to the <b>DdeNameService</b> or <b>DdeInitialize</b> function. This flag is always in effect when an application calls <b>DdeInitialize</b> for the first time, regardless of whether the application specifies the flag. On subsequent calls to <b>DdeInitialize</b>, not specifying this flag turns off the application's service-name filters, but specifying it turns on the application's service name filters.</p>
<b>CBF_FAIL_ALLSVRXACTIONS</b> 0x0003f000	<p>Prevents the callback function from receiving server transactions. The system returns <b>DDE_FNOTPROCESSED</b> to each client that sends a transaction to this application. This flag is equivalent to combining all <b>CBF_FAIL_</b> flags.</p>
<b>CBF_FAIL_ADVISES</b> 0x00004000	<p>Prevents the callback function from receiving <b>XTYP_ADVSTART</b> and <b>XTYP_ADVSTOP</b> transactions. The system returns <b>DDE_FNOTPROCESSED</b> to each client that sends an <b>XTYP_ADVSTART</b> or <b>XTYP_ADVSTOP</b> transaction to the server.</p>
<b>CBF_FAIL_CONNECTIONS</b> 0x00002000	<p>Prevents the callback function from receiving <b>XTYP_CONNECT</b> and <b>XTYP_WILDCONNECT</b> transactions.</p>
<b>CBF_FAIL_EXECUTES</b> 0x00008000	<p>Prevents the callback function from receiving <b>XTYP_EXECUTE</b> transactions. The system returns <b>DDE_FNOTPROCESSED</b> to a client that sends an <b>XTYP_EXECUTE</b> transaction to the server.</p>
<b>CBF_FAIL_POKES</b> 0x00010000	<p>Prevents the callback function from receiving <b>XTYP_POKE</b> transactions. The system returns <b>DDE_FNOTPROCESSED</b> to a client that sends an <b>XTYP_POKE</b> transaction to the server.</p>
<b>CBF_FAIL_REQUESTS</b> 0x00020000	<p>Prevents the callback function from receiving <b>XTYP_REQUEST</b> transactions. The system returns <b>DDE_FNOTPROCESSED</b> to a client that sends an <b>XTYP_REQUEST</b> transaction to the server.</p>

<b>CBF_FAIL_SELFCONNECTIONS</b> 0x00001000	Prevents the callback function from receiving <a href="#">XTYP_CONNECT</a> transactions from the application's own instance. This flag prevents an application from establishing a DDE conversation with its own instance. An application should use this flag if it needs to communicate with other instances of itself but not with itself.
<b>CBF_SKIP_ALLNOTIFICATIONS</b> 0x003c0000	Prevents the callback function from receiving any notifications. This flag is equivalent to combining all CBF_SKIP_ flags.
<b>CBF_SKIP_CONNECT_CONFIRMS</b> 0x00040000	Prevents the callback function from receiving <a href="#">XTYP_CONNECT_CONFIRM</a> notifications.
<b>CBF_SKIP_DISCONNECTS</b> 0x00200000	Prevents the callback function from receiving <a href="#">XTYP_DISCONNECT</a> notifications.
<b>CBF_SKIP_REGISTRATIONS</b> 0x00080000	Prevents the callback function from receiving <a href="#">XTYP_REGISTER</a> notifications.
<b>CBF_SKIP_UNREGISTRATIONS</b> 0x00100000	Prevents the callback function from receiving <a href="#">XTYP_UNREGISTER</a> notifications.
<b>MF_CALLBACKS</b> 0x08000000	Notifies the callback function whenever a transaction is sent to any DDE callback function in the system.
<b>MF_CONV</b> 0x40000000	Notifies the callback function whenever a conversation is established or terminated.
<b>MF_ERRORS</b> 0x10000000	Notifies the callback function whenever a DDE error occurs.
<b>MF_HSZ_INFO</b> 0x01000000	Notifies the callback function whenever a DDE application creates, frees, or increments the usage count of a string handle or whenever a string handle is freed as a result of a call to the <a href="#">DdeUninitialize</a> function.
<b>MF_LINKS</b> 0x20000000	Notifies the callback function whenever an advise loop is started or ended.
<b>MF_POSTMSGs</b> 0x04000000	Notifies the callback function whenever the system or an application posts a DDE message.

<b>MF_SENDMSGS</b> 0x02000000	Notifies the callback function whenever the system or an application sends a DDE message.
----------------------------------	---

ulRes

Type: **DWORD**

Reserved; must be set to zero.

## Return value

Type: **UINT**

If the function succeeds, the return value is **DMLERR\_NO\_ERROR**.

If the function fails, the return value is one of the following values:

## Remarks

An application that uses multiple instances of the DDEML must not pass DDEML objects between instances.

A DDE monitoring application should not attempt to perform DDE operations (establish conversations, issue transactions, and so on) within the context of the same application instance.

A synchronous transaction fails with a **DMLERR\_REENTRANCY** error if any instance of the same task has a synchronous transaction already in progress.

The **CBF\_FAIL\_ALLSVRXACTIONS** flag causes the DDEML to filter all server transactions and can be changed by a subsequent call to **DdeInitialize**. The **APPCMD\_CLIENTONLY** flag prevents the DDEML from creating key resources for the server and cannot be changed by a subsequent call to **DdeInitialize**.

There is an ANSI version and a Unicode version of **DdeInitialize**. The version called determines the type of the window procedures used to control DDE conversations (ANSI or Unicode), and the default value for the *iCodePage* member of the **CONVCONTEXT** structure (**CP\_WINANSI** or **CP\_WINUNICODE**).

### NOTE

The `ddeml.h` header defines `DdeInitialize` 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

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	<code>ddeml.h</code> (include <code>Windows.h</code> )

<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

## See also

[Dynamic Data Exchange Management Library Overview](#)

# DdeInitializeW function (ddeml.h)

1/15/2021 • 5 minutes to read • [Edit Online](#)

Registers an application with the [Dynamic Data Exchange Management Library](#) (DDEML). An application must call this function before calling any other Dynamic Data Exchange Management Library (DDEML) function.

## Syntax

```
UINT DdeInitializeW(  
    LPDWORD    pidInst,  
    PFNCALLBACK pfnCallback,  
    DWORD      afCmd,  
    DWORD      ulRes  
);
```

## Parameters

`pidInst`

Type: **LPDWORD**

The application instance identifier. At initialization, this parameter should point to 0. If the function succeeds, this parameter points to the instance identifier for the application. This value should be passed as the *idInst* parameter in all other DDEML functions that require it. If an application uses multiple instances of the DDEML dynamic-link library (DLL), the application should provide a different callback function for each instance.

If *pidInst* points to a nonzero value, reinitialization of the DDEML is implied. In this case, *pidInst* must point to a valid application-instance identifier.

`pfnCallback`

Type: **PFNCALLBACK**

A pointer to the application-defined DDE callback function. This function processes DDE transactions sent by the system. For more information, see the [DdeCallback](#) callback function.

`afCmd`

Type: **DWORD**

A set of **APPCMD\_**, **CBF\_**, and **MF\_** flags. The **APPCMD\_** flags provide special instructions to **DdeInitialize**. The **CBF\_** flags specify filters that prevent specific types of transactions from reaching the callback function. The **MF\_** flags specify the types of DDE activity that a DDE monitoring application monitors. Using these flags enhances the performance of a DDE application by eliminating unnecessary calls to the callback function.

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

VALUE	MEANING
-------	---------

<b>APPCLASS_MONITOR</b> 0x00000001L	<p>Makes it possible for the application to monitor DDE activity in the system. This flag is for use by DDE monitoring applications. The application specifies the types of DDE activity to monitor by combining one or more monitor flags with the <b>APPCLASS_MONITOR</b> flag. For details, see the following Remarks section.</p>
<b>APPCLASS_STANDARD</b> 0x00000000L	<p>Registers the application as a standard (nonmonitoring) DDEML application.</p>
<b>APPCMD_CLIENTONLY</b> 0x00000010L	<p>Prevents the application from becoming a server in a DDE conversation. The application can only be a client. This flag reduces consumption of resources by the DDEML. It includes the functionality of the <b>CBF_FAIL_ALLSVRXACTIONS</b> flag.</p>
<b>APPCMD_FILTERINITS</b> 0x00000020L	<p>Prevents the DDEML from sending <b>XTYP_CONNECT</b> and <b>XTYP_WILDCONNECT</b> transactions to the application until the application has created its string handles and registered its service names or has turned off filtering by a subsequent call to the <b>DdeNameService</b> or <b>DdeInitialize</b> function. This flag is always in effect when an application calls <b>DdeInitialize</b> for the first time, regardless of whether the application specifies the flag. On subsequent calls to <b>DdeInitialize</b>, not specifying this flag turns off the application's service-name filters, but specifying it turns on the application's service name filters.</p>
<b>CBF_FAIL_ALLSVRXACTIONS</b> 0x0003f000	<p>Prevents the callback function from receiving server transactions. The system returns <b>DDE_FNOTPROCESSED</b> to each client that sends a transaction to this application. This flag is equivalent to combining all <b>CBF_FAIL_</b> flags.</p>
<b>CBF_FAIL_ADVISES</b> 0x00004000	<p>Prevents the callback function from receiving <b>XTYP_ADVSTART</b> and <b>XTYP_ADVSTOP</b> transactions. The system returns <b>DDE_FNOTPROCESSED</b> to each client that sends an <b>XTYP_ADVSTART</b> or <b>XTYP_ADVSTOP</b> transaction to the server.</p>
<b>CBF_FAIL_CONNECTIONS</b> 0x00002000	<p>Prevents the callback function from receiving <b>XTYP_CONNECT</b> and <b>XTYP_WILDCONNECT</b> transactions.</p>
<b>CBF_FAIL_EXECUTES</b> 0x00008000	<p>Prevents the callback function from receiving <b>XTYP_EXECUTE</b> transactions. The system returns <b>DDE_FNOTPROCESSED</b> to a client that sends an <b>XTYP_EXECUTE</b> transaction to the server.</p>
<b>CBF_FAIL_POKES</b> 0x00010000	<p>Prevents the callback function from receiving <b>XTYP_POKE</b> transactions. The system returns <b>DDE_FNOTPROCESSED</b> to a client that sends an <b>XTYP_POKE</b> transaction to the server.</p>
<b>CBF_FAIL_REQUESTS</b> 0x00020000	<p>Prevents the callback function from receiving <b>XTYP_REQUEST</b> transactions. The system returns <b>DDE_FNOTPROCESSED</b> to a client that sends an <b>XTYP_REQUEST</b> transaction to the server.</p>



<b>CBF_FAIL_SELFCONNECTIONS</b> 0x00001000	Prevents the callback function from receiving <a href="#">XTYP_CONNECT</a> transactions from the application's own instance. This flag prevents an application from establishing a DDE conversation with its own instance. An application should use this flag if it needs to communicate with other instances of itself but not with itself.
<b>CBF_SKIP_ALLNOTIFICATIONS</b> 0x003c0000	Prevents the callback function from receiving any notifications. This flag is equivalent to combining all CBF_SKIP_ flags.
<b>CBF_SKIP_CONNECT_CONFIRMS</b> 0x00040000	Prevents the callback function from receiving <a href="#">XTYP_CONNECT_CONFIRM</a> notifications.
<b>CBF_SKIP_DISCONNECTS</b> 0x00200000	Prevents the callback function from receiving <a href="#">XTYP_DISCONNECT</a> notifications.
<b>CBF_SKIP_REGISTRATIONS</b> 0x00080000	Prevents the callback function from receiving <a href="#">XTYP_REGISTER</a> notifications.
<b>CBF_SKIP_UNREGISTRATIONS</b> 0x00100000	Prevents the callback function from receiving <a href="#">XTYP_UNREGISTER</a> notifications.
<b>MF_CALLBACKS</b> 0x08000000	Notifies the callback function whenever a transaction is sent to any DDE callback function in the system.
<b>MF_CONV</b> 0x40000000	Notifies the callback function whenever a conversation is established or terminated.
<b>MF_ERRORS</b> 0x10000000	Notifies the callback function whenever a DDE error occurs.
<b>MF_HSZ_INFO</b> 0x01000000	Notifies the callback function whenever a DDE application creates, frees, or increments the usage count of a string handle or whenever a string handle is freed as a result of a call to the <a href="#">DdeUninitialize</a> function.
<b>MF_LINKS</b> 0x20000000	Notifies the callback function whenever an advise loop is started or ended.
<b>MF_POSTMSGs</b> 0x04000000	Notifies the callback function whenever the system or an application posts a DDE message.

<b>MF_SENDMSGS</b> 0x02000000	Notifies the callback function whenever the system or an application sends a DDE message.
----------------------------------	---

ulRes

Type: **DWORD**

Reserved; must be set to zero.

## Return value

Type: **UINT**

If the function succeeds, the return value is **DMLERR\_NO\_ERROR**.

If the function fails, the return value is one of the following values:

## Remarks

An application that uses multiple instances of the DDEML must not pass DDEML objects between instances.

A DDE monitoring application should not attempt to perform DDE operations (establish conversations, issue transactions, and so on) within the context of the same application instance.

A synchronous transaction fails with a **DMLERR\_REENTRANCY** error if any instance of the same task has a synchronous transaction already in progress.

The **CBF\_FAIL\_ALLSVRXACTIONS** flag causes the DDEML to filter all server transactions and can be changed by a subsequent call to **DdeInitialize**. The **APPCMD\_CLIENTONLY** flag prevents the DDEML from creating key resources for the server and cannot be changed by a subsequent call to **DdeInitialize**.

There is an ANSI version and a Unicode version of **DdeInitialize**. The version called determines the type of the window procedures used to control DDE conversations (ANSI or Unicode), and the default value for the *iCodePage* member of the **CONVCONTEXT** structure (**CP\_WINANSI** or **CP\_WINUNICODE**).

### NOTE

The `ddeml.h` header defines `DdeInitialize` 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

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	<code>ddeml.h</code> (include <code>Windows.h</code> )

<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

## See also

[Dynamic Data Exchange Management Library Overview](#)

# DdeKeepStringHandle function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Increments the usage count associated with the specified handle. This function enables an application to save a string handle passed to the application's Dynamic Data Exchange (DDE) callback function. Otherwise, a string handle passed to the callback function is deleted when the callback function returns. This function should also be used to keep a copy of a string handle referenced by the [CONVINFO](#) structure returned by the [DdeQueryConvInfo](#) function.

## Syntax

```
BOOL DdeKeepStringHandle(  
    DWORD idInst,  
    HSZ    hsz  
);
```

## Parameters

`idInst`

Type: **DWORD**

The application instance identifier obtained by a previous call to the [DdeInitialize](#) function.

`hsz`

Type: **HSZ**

A handle to the string handle to be saved.

## Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	ddeml.h (include Windows.h)
<b>Library</b>	User32.lib

DLL	User32.dll

# See also

[CONVINFO](#)

## Conceptual

[DdeCreateStringHandle](#)

[DdeFreeStringHandle](#)

[DdeInitialize](#)

[DdeQueryConvInfo](#)

[DdeQueryString](#)

[Dynamic Data Exchange Management Library](#)

## Reference

# DDEML\_MSG\_HOOK\_DATA structure (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Contains information about a Dynamic Data Exchange (DDE) message, and provides read access to the data referenced by the message. This structure is intended to be used by a Dynamic Data Exchange Management Library (DDEML) monitoring application.

## Syntax

```
typedef struct tagDDEML_MSG_HOOK_DATA {
    UINT_PTR uiLo;
    UINT_PTR uiHi;
    DWORD    cbData;
    DWORD    Data[8];
} DDEML_MSG_HOOK_DATA, *PDDEML_MSG_HOOK_DATA;
```

## Members

**uiLo**

Type: **UINT\_PTR**

The unpacked low-order word of the *lParam* parameter associated with the DDE message.

**uiHi**

Type: **UINT\_PTR**

The unpacked high-order word of the *lParam* parameter associated with the DDE message.

**cbData**

Type: **DWORD**

The amount of data being passed with the message, in bytes. This value can be greater than 32.

**Data**

Type: **DWORD[8]**

The first 32 bytes of data being passed with the message ( **8 \* sizeof(DWORD)** ).

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Header</b>	ddeml.h (include Windows.h)

## See also

### Conceptual

[Dynamic Data Exchange Management Library](#)

[MONCBSTRUCT](#)

[MONCONVSTRUCT](#)

[MONERRSTRUCT](#)

[MONHSZSTRUCT](#)

[MONLINKSTRUCT](#)

[MONMSGSTRUCT](#)

### Reference

# DdeNameService function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Registers or unregisters the service names a Dynamic Data Exchange (DDE) server supports. This function causes the system to send [XTYP\\_REGISTER](#) or [XTYP\\_UNREGISTER](#) transactions to other running [Dynamic Data Exchange Management Library](#) (DDEML) client applications.

## Syntax

```
HDDEDATA DdeNameService(  
    DWORD idInst,  
    HSZ hsz1,  
    HSZ hsz2,  
    UINT afCmd  
);
```

## Parameters

`idInst`

Type: **DWORD**

The application instance identifier obtained by a previous call to the [DdeInitialize](#) function.

`hsz1`

Type: **HSZ**

A handle to the string that specifies the service name the server is registering or unregistering. An application that is unregistering all of its service names should set this parameter to 0L.

`hsz2`

Type: **HSZ**

Reserved; should be set to 0L.

`afCmd`

Type: **UINT**

The service name options. This parameter can be one of the following values.

VALUE	MEANING
<b>DNS_REGISTER</b> 0x0001	Registers the error code service name.
<b>DNS_UNREGISTER</b> 0x0002	Unregisters the error code service name. If the <i>hsz1</i> parameter is 0L, all service names registered by the server will be unregistered.



<b>DNS_FILTERON</b> 0x0004	Turns on service name initiation filtering. The filter prevents a server from receiving <a href="#">XTYP_CONNECT</a> transactions for service names it has not registered. This is the default setting for this filter.  If a server application does not register any service names, the application cannot receive <a href="#">XTYP_WILDCONNECT</a> transactions.
<b>DNS_FILTEROFF</b> 0x0008	Turns off service name initiation filtering. If this flag is specified, the server receives an <a href="#">XTYP_CONNECT</a> transaction whenever another DDE application calls the <a href="#">DdeConnect</a> function, regardless of the service name.

## Return value

Type: **HDDEDATA**

If the function succeeds, it returns a nonzero value. That value is not a true **HDDEDATA** value, merely a Boolean indicator of success. The function is typed **HDDEDATA** to allow for possible future expansion of the function and a more sophisticated return value.

If the function fails, the return value is 0L.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Remarks

The service name identified by the *hsz1* parameter should be a base name (that is, the name should contain no instance-specific information). The system generates an instance-specific name and sends it along with the base name during the [XTYP\\_REGISTER](#) and [XTYP\\_UNREGISTER](#) transactions. The receiving applications can then connect to the specific application instance.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	ddeml.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

## See also

Conceptual

[DdeConnect](#)

[DdeConnectList](#)

[DdelInitialize](#)

[Dynamic Data Exchange Management Library](#)

## **Reference**

[XTYP\\_REGISTER](#)

[XTYP\\_UNREGISTER](#)

# DdePostAdvise function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Causes the system to send an [XTYP\\_ADVREQ](#) transaction to the calling (server) application's Dynamic Data Exchange (DDE) callback function for each client with an active advise loop on the specified topic and item. A server application should call this function whenever the data associated with the topic name or item name pair changes.

## Syntax

```
BOOL DdePostAdvise(  
    DWORD idInst,  
    HSZ   hszTopic,  
    HSZ   hszItem  
);
```

## Parameters

`idInst`

Type: **DWORD**

The application instance identifier obtained by a previous call to the [DdeInitialize](#) function.

`hszTopic`

Type: **HSZ**

A handle to a string that specifies the topic name. To send notifications for all topics with active advise loops, an application can set this parameter to 0L.

`hszItem`

Type: **HSZ**

A handle to a string that specifies the item name. To send notifications for all items with active advise loops, an application can set this parameter to 0L.

## Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Remarks

A server that has nonenumerable topics or items should set the *hszTopic* and *hszItem* parameters to **NULL** so that the system generates transactions for all active advise loops. The server's DDE callback function returns **NULL** for any advise loops that must not be updated.

If a server calls **DdePostAdvise** with a topic, item, and format name set that includes the set currently being handled in an [XTYP\\_ADVREQ](#) callback, a stack overflow can result.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	ddeml.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

## See also

### Conceptual

[DdeInitialize](#)

[Dynamic Data Exchange Management Library](#)

### Reference

[XTYP\\_ADVREQ](#)

# DdeQueryConvInfo function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Retrieves information about a Dynamic Data Exchange (DDE) transaction and about the conversation in which the transaction takes place.

## Syntax

```
UINT DdeQueryConvInfo(  
    HCONV      hConv,  
    DWORD      idTransaction,  
    PCONVINFO  pConvInfo  
);
```

## Parameters

**hConv**

Type: **HCONV**

A handle to the conversation.

**idTransaction**

Type: **DWORD**

The transaction. For asynchronous transactions, this parameter should be a transaction identifier returned by the [DdeClientTransaction](#) function. For synchronous transactions, this parameter should be QID\_SYNC.

**pConvInfo**

Type: **PCONVINFO**

A pointer to the [CONVINFO](#) structure that receives information about the transaction and conversation. The *cb* member of the **CONVINFO** structure must specify the length of the buffer allocated for the structure.

## Return value

Type: **UINT**

If the function succeeds, the return value is the number of bytes copied into the [CONVINFO](#) structure.

If the function fails, the return value is **FALSE**.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Remarks

An application should not free a string handle referenced by the [CONVINFO](#) structure. If an application must use one of these string handles, it should call the [DdeKeepStringHandle](#) function to create a copy of the handle.

If the *idTransaction* parameter is set to QID\_SYNC, the *hUser* member of the [CONVINFO](#) structure is associated with the conversation and can be used to hold data associated with the conversation. If *idTransaction* is the identifier of an asynchronous transaction, the *hUser* member is associated only with the current transaction and

is valid only for the duration of the transaction.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	ddeml.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

## See also

[CONVINFO](#)

### Conceptual

[DdeClientTransaction](#)

[DdeConnect](#)

[DdeConnectList](#)

[DdeKeepStringHandle](#)

[DdeQueryNextServer](#)

[Dynamic Data Exchange Management Library](#)

### Reference

# DdeQueryNextServer function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Retrieves the next conversation handle in the specified conversation list.

## Syntax

```
HCONV DdeQueryNextServer(  
    HCONVLIST hConvList,  
    HCONV     hConvPrev  
);
```

## Parameters

**hConvList**

Type: **HCONVLIST**

A handle to the conversation list. This handle must have been created by a previous call to the [DdeConnectList](#) function.

**hConvPrev**

Type: **HCONV**

A handle to the conversation handle previously returned by this function. If this parameter is 0L, the function returns the first conversation handle in the list.

## Return value

Type: **HCONV**

If the list contains any more conversation handles, the return value is the next conversation handle in the list; otherwise, it is 0L.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	ddeml.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

## See also

### Conceptual

[DdeConnectList](#)

[DdeDisconnectList](#)

[Dynamic Data Exchange Management Library](#)

### Reference



# DdeQueryStringA function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Copies text associated with a string handle into a buffer.

## Syntax

```
DWORD DdeQueryStringA(  
    DWORD idInst,  
    HSZ    hsz,  
    LPSTR psz,  
    DWORD cchMax,  
    int    iCodePage  
);
```

## Parameters

idInst

Type: **DWORD**

The application instance identifier obtained by a previous call to the [DdeInitialize](#) function.

hsz

Type: **HSZ**

A handle to the string to copy. This handle must have been created by a previous call to the [DdeCreateStringHandle](#) function.

psz

Type: **LPTSTR**

A pointer to a buffer that receives the string. To obtain the length of the string, this parameter should be set to **NULL**.

cchMax

Type: **DWORD**

The length, in characters, of the buffer pointed to by the *psz* parameter. For the ANSI version of the function, this is the number of bytes; for the Unicode version, this is the number of characters. If the string is longer than (*cchMax* - 1), it will be truncated. If the *psz* parameter is set to **NULL**, this parameter is ignored.

iCodePage

Type: **int**

The code page used to render the string. This value should be either **CP\_WINANSI** or **CP\_WINUNICODE**.

## Return value

Type: **DWORD**

If the *psz* parameter specified a valid pointer, the return value is the length, in characters, of the returned text (not including the terminating null character). If the *psz* parameter specified a **NULL** pointer, the return value is the length of the text associated with the *hsz* parameter (not including the terminating null character). If an error occurs, the return value is 0L.

## Remarks

The string returned in the buffer is always null-terminated. If the string is longer than ( *cchMax*– 1), only the first ( *cchMax*– 1) characters of the string are copied.

If the *psz* parameter is **NULL**, the **DdeQueryString** function obtains the length, in bytes, of the string associated with the string handle. The length does not include the terminating null character.

### NOTE

The `ddeml.h` header defines `DdeQueryString` 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

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	ddeml.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

## See also

### Conceptual

[DdeCmpStringHandles](#)

[DdeCreateStringHandle](#)

[DdeFreeStringHandle](#)

[DdeInitialize](#)

[Dynamic Data Exchange Management Library](#)

### Reference

# DdeQueryStringW function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Copies text associated with a string handle into a buffer.

## Syntax

```
DWORD DdeQueryString(  
    DWORD    idInst,  
    HSZ      hsz,  
    LPWSTR   psz,  
    DWORD    cchMax,  
    int      iCodePage  
);
```

## Parameters

`idInst`

Type: **DWORD**

The application instance identifier obtained by a previous call to the [DdeInitialize](#) function.

`hsz`

Type: **HSZ**

A handle to the string to copy. This handle must have been created by a previous call to the [DdeCreateStringHandle](#) function.

`psz`

Type: **LPTSTR**

A pointer to a buffer that receives the string. To obtain the length of the string, this parameter should be set to **NULL**.

`cchMax`

Type: **DWORD**

The length, in characters, of the buffer pointed to by the *psz* parameter. For the ANSI version of the function, this is the number of bytes; for the Unicode version, this is the number of characters. If the string is longer than (*cchMax* - 1), it will be truncated. If the *psz* parameter is set to **NULL**, this parameter is ignored.

`iCodePage`

Type: **int**

The code page used to render the string. This value should be either **CP\_WINANSI** or **CP\_WINUNICODE**.

## Return value

Type: **DWORD**

If the *psz* parameter specified a valid pointer, the return value is the length, in characters, of the returned text (not including the terminating null character). If the *psz* parameter specified a **NULL** pointer, the return value is the length of the text associated with the *hpsz* parameter (not including the terminating null character). If an error occurs, the return value is 0L.

## Remarks

The string returned in the buffer is always null-terminated. If the string is longer than ( *cchMax*– 1), only the first ( *cchMax*– 1) characters of the string are copied.

If the *psz* parameter is **NULL**, the **DdeQueryString** function obtains the length, in bytes, of the string associated with the string handle. The length does not include the terminating null character.

### NOTE

The `ddeml.h` header defines `DdeQueryString` 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

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	ddeml.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

## See also

### Conceptual

[DdeCmpStringHandles](#)

[DdeCreateStringHandle](#)

[DdeFreeStringHandle](#)

[DdeInitialize](#)

[Dynamic Data Exchange Management Library](#)

### Reference

# DdeReconnect function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Enables a client [Dynamic Data Exchange Management Library](#) (DDEML) application to attempt to reestablish a conversation with a service that has terminated a conversation with the client. When the conversation is reestablished, the Dynamic Data Exchange Management Library (DDEML) attempts to reestablish any preexisting advise loops.

## Syntax

```
HCONV DdeReconnect(  
    HCONV hConv  
);
```

## Parameters

hConv

Type: **HCONV**

A handle to the conversation to be reestablished. A client must have obtained the conversation handle by a previous call to the [DdeConnect](#) function or from an [XTYP\\_DISCONNECT](#) transaction.

## Return value

Type: **HCONV**

If the function succeeds, the return value is the handle to the reestablished conversation.

If the function fails, the return value is 0L.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	ddeml.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

## See also

### Conceptual

[DdeConnect](#)

[DdeDisconnect](#)

[Dynamic Data Exchange Management Library](#)

### Reference

# DdeSetUserHandle function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Associates an application-defined value with a conversation handle or a transaction identifier. This is useful for simplifying the processing of asynchronous transactions. An application can use the [DdeQueryConvInfo](#) function to retrieve this value.

## Syntax

```
BOOL DdeSetUserHandle(  
    HCONV      hConv,  
    DWORD      id,  
    DWORD_PTR  hUser  
);
```

## Parameters

**hConv**

Type: **HCONV**

A handle to the conversation.

**id**

Type: **DWORD**

The transaction identifier to associate with the value specified by the *hUser* parameter. An application should set this parameter to QID\_SYNC to associate *hUser* with the conversation identified by the *hConv* parameter.

**hUser**

Type: **DWORD\_PTR**

The value to be associated with the conversation handle.

## Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]

<b>Target Platform</b>	Windows
<b>Header</b>	ddeml.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

## See also

### Conceptual

[DdeQueryConvInfo](#)

[Dynamic Data Exchange Management Library](#)

### Reference



# DdeUnaccessData function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Unaccesses a Dynamic Data Exchange (DDE) object. An application must call this function after it has finished accessing the object.

## Syntax

```
BOOL DdeUnaccessData(  
    HDDEDATA hData  
);
```

## Parameters

**hData**

Type: **HDDEDATA**

A handle to the DDE object.

## Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

The [DdeGetLastError](#) function can be used to get the error code, which can be one of the following values:

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	ddeml.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

## See also

Conceptual

[DdeAccessData](#)

[DdeAddData](#)

[DdeCreateDataHandle](#)

[DdeFreeDataHandle](#)

[Dynamic Data Exchange Management Library](#)

**Reference**

# DdeUninitialize function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Frees all Dynamic Data Exchange Management Library (DDEML) resources associated with the calling application.

## Syntax

```
BOOL DdeUninitialize(  
    DWORD idInst  
);
```

## Parameters

idInst

Type: **DWORD**

The application instance identifier obtained by a previous call to the [DdeInitialize](#) function.

## Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

## Remarks

DdeUninitialize terminates any conversations currently open for the application.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	ddeml.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

## See also

## Conceptual

[DdeDisconnect](#)

[DdeDisconnectList](#)

[DdeInitialize](#)

[Dynamic Data Exchange Management Library](#)

## Reference

# HSZPAIR structure (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Contains a DDE service name and topic name. A DDE server application can use this structure during an [XTYP\\_WILDCONNECT](#) transaction to enumerate the service-topic pairs that it supports.

## Syntax

```
typedef struct tagHSZPAIR {  
    HSZ hszSvc;  
    HSZ hszTopic;  
} HSZPAIR, *PHSZPAIR;
```

## Members

`hszSvc`

Type: [HSZ](#)

A handle to the service name.

`hszTopic`

Type: [HSZ](#)

A handle to the topic name.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Header</b>	ddeml.h (include Windows.h)

## See also

[About Dynamic Data Exchange](#)

# MONCBSTRUCT structure (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Contains information about the current Dynamic Data Exchange (DDE) transaction. A DDE debugging application can use this structure when monitoring transactions that the system passes to the DDE callback functions of other applications.

## Syntax

```
typedef struct tagMONCBSTRUCT {
    UINT        cb;
    DWORD       dwTime;
    HANDLE      hTask;
    DWORD       dwRet;
    UINT        wType;
    UINT        wFmt;
    HCONV       hConv;
    HSZ         hsz1;
    HSZ         hsz2;
    HDDEDATA    hData;
    ULONG_PTR   dwData1;
    ULONG_PTR   dwData2;
    CONVCONTEXT cc;
    DWORD       cbData;
    DWORD       Data[8];
} MONCBSTRUCT, *PMONCBSTRUCT;
```

## Members

cb

Type: **UINT**

The structure's size, in bytes.

dwTime

Type: **DWORD**

The Windows time at which the transaction occurred. Windows time is the number of milliseconds that have elapsed since the system was booted.

hTask

Type: **HANDLE**

A handle to the task (application instance) containing the DDE callback function that received the transaction.

dwRet

Type: **DWORD**

The value returned by the DDE callback function that processed the transaction.

wType

Type: **UINT**

The transaction type.

wFmt

Type: **UINT**

The format of the data exchanged (if any) during the transaction.

hConv

Type: **HCONV**

A handle to the conversation in which the transaction took place.

hsz1

Type: **HSZ**

A handle to a string.

hsz2

Type: **HSZ**

A handle to a string.

hData

Type: **HDDEDATA**

A handle to the data exchanged (if any) during the transaction.

dwData1

Type: **ULONG\_PTR**

Additional data.

dwData2

Type: **ULONG\_PTR**

Additional data.

cc

Type: **CONVCONTEXT**

The language information used to share data in different languages.

cbData

Type: **DWORD**

The amount, in bytes, of data being passed with the transaction. This value can be more than 32 bytes.

Data

Type: **DWORD[8]**

Contains the first 32 bytes of data being passed with the transaction ( `8 * sizeof(DWORD)` ).

## Requirements

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

# See also

[CONVCONTEXT](#)

## Conceptual

[Dynamic Data Exchange Management Library](#)

[MONERRSTRUCT](#)

[MONHSZSTRUCT](#)

[MONLINKSTRUCT](#)

[MONMSGSTRUCT](#)

## Reference



# MONCONVSTRUCT structure (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Contains information about a Dynamic Data Exchange (DDE) conversation. A DDE monitoring application can use this structure to obtain information about a conversation that has been established or has terminated.

## Syntax

```
typedef struct tagMONCONVSTRUCT {
    UINT    cb;
    BOOL    fConnect;
    DWORD   dwTime;
    HANDLE  hTask;
    HSZ     hszSvc;
    HSZ     hszTopic;
    HCONV   hConvClient;
    HCONV   hConvServer;
} MONCONVSTRUCT, *PMONCONVSTRUCT;
```

## Members

cb

Type: **UINT**

The structure's size, in bytes.

fConnect

Type: **BOOL**

Indicates whether the conversation is currently established. A value of **TRUE** indicates the conversation is established; **FALSE** indicates it is not.

dwTime

Type: **DWORD**

The Windows time at which the conversation was established or terminated. Windows time is the number of milliseconds that have elapsed since the system was booted.

hTask

Type: **HANDLE**

A handle to a task (application instance) that is a partner in the conversation.

hszSvc

Type: **HSZ**

A handle to the service name on which the conversation is established.

hszTopic

Type: **HSZ**

A handle to the topic name on which the conversation is established.

`hConvClient`

Type: **HCONV**

A handle to the client conversation.

`hConvServer`

Type: **HCONV**

A handle to the server conversation.

## Remarks

Because string handles are local to the process, the **hszSvc** and **hszTopic** members are global atoms. Similarly, conversation handles are local to the instance; therefore, the **hConvClient** and **hConvServer** members are window handles.

The **hConvClient** and **hConvServer** members of the **MONCONVSTRUCT** structure do not hold the same value as would be seen by the applications engaged in the conversation. Instead, they hold a globally unique pair of values that identify the conversation.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Header</b>	ddeml.h (include Windows.h)

## See also

### Conceptual

[Dynamic Data Exchange Management Library](#)

[MONCBSTRUCT](#)

[MONERRSTRUCT](#)

[MONHSZSTRUCT](#)

[MONLINKSTRUCT](#)

[MONMSGSTRUCT](#)

### Reference

# MONERRSTRUCT structure (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Contains information about the current Dynamic Data Exchange (DDE) error. A DDE monitoring application can use this structure to monitor errors returned by DDE Management Library functions.

## Syntax

```
typedef struct tagMONERRSTRUCT {
    UINT    cb;
    UINT    wLastError;
    DWORD   dwTime;
    HANDLE  hTask;
} MONERRSTRUCT, *PMONERRSTRUCT;
```

## Members

cb

Type: **UINT**

The structure's size, in bytes.

wLastError

Type: **UINT**

The current error.

dwTime

Type: **DWORD**

The Windows time at which the error occurred. Windows time is the number of milliseconds that have elapsed since the system was booted.

hTask

Type: **HANDLE**

A handle to the task (application instance) that called the DDE function that caused the error.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Header</b>	ddeml.h (include Windows.h)

## See also

### Conceptual

[Dynamic Data Exchange Management Library](#)

[MONCBSTRUCT](#)

[MONCONVSTRUCT](#)

[MONHSZSTRUCT](#)

[MONLINKSTRUCT](#)

[MONMSGSTRUCT](#)

### Reference

# MONHSZSTRUCTA structure (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Contains information about a Dynamic Data Exchange (DDE) string handle. A DDE monitoring application can use this structure when monitoring the activity of the string manager component of the DDE Management Library.

## Syntax

```
typedef struct tagMONHSZSTRUCTA {
    UINT    cb;
    BOOL    fsAction;
    DWORD   dwTime;
    HSZ     hsz;
    HANDLE  hTask;
    CHAR    str[1];
} MONHSZSTRUCTA, *PMONHSZSTRUCTA;
```

## Members

cb

Type: **UINT**

The structure's size, in bytes.

fsAction

Type: **BOOL**

The action being performed on the string identified by the **hsz** member.

VALUE	MEANING
<b>MH_CLEANUP</b> 4	An application is freeing its DDE resources, causing the system to delete string handles the application had created. (The application called the <a href="#">DdeUninitialize</a> function.)
<b>MH_CREATE</b> 1	An application is creating a string handle. (The application called the <a href="#">DdeCreateStringHandle</a> function.)
<b>MH_DELETE</b> 3	An application is deleting a string handle. (The application called the <a href="#">DdeFreeStringHandle</a> function.)
<b>MH_KEEP</b> 2	An application is increasing the usage count of a string handle. (The application called the <a href="#">DdeKeepStringHandle</a> function.)

dwTime

Type: **DWORD**

The Windows time at which the action specified by the **fsAction** member takes place. Windows time is the number of milliseconds that have elapsed since the system was booted.

hsz

Type: **HSZ**

A handle to the string. Because string handles are local to the process, this member is a global atom.

hTask

Type: **HANDLE**

A handle to the task (application instance) performing the action on the string handle.

str

Type: **TCHAR[1]**

Pointer to the string identified by the **hsz** member.

## Remarks

### NOTE

The ddeml.h header defines MONHSZSTRUCT 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

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

## See also

### Conceptual

[Dynamic Data Exchange Management Library](#)

[MONCBSTRUCT](#)

[MONCONVSTRUCT](#)

[MONERRSTRUCT](#)

[MONLINKSTRUCT](#)

[MONMSGSTRUCT](#)

### Reference

# MONHSZSTRUCTW structure (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Contains information about a Dynamic Data Exchange (DDE) string handle. A DDE monitoring application can use this structure when monitoring the activity of the string manager component of the DDE Management Library.

## Syntax

```
typedef struct tagMONHSZSTRUCTW {
    UINT    cb;
    BOOL    fsAction;
    DWORD   dwTime;
    HSZ     hsz;
    HANDLE  hTask;
    WCHAR   str[1];
} MONHSZSTRUCTW, *PMONHSZSTRUCTW;
```

## Members

cb

Type: **UINT**

The structure's size, in bytes.

fsAction

Type: **BOOL**

The action being performed on the string identified by the **hsz** member.

VALUE	MEANING
<b>MH_CLEANUP</b> 4	An application is freeing its DDE resources, causing the system to delete string handles the application had created. (The application called the <a href="#">DdeUninitialize</a> function.)
<b>MH_CREATE</b> 1	An application is creating a string handle. (The application called the <a href="#">DdeCreateStringHandle</a> function.)
<b>MH_DELETE</b> 3	An application is deleting a string handle. (The application called the <a href="#">DdeFreeStringHandle</a> function.)
<b>MH_KEEP</b> 2	An application is increasing the usage count of a string handle. (The application called the <a href="#">DdeKeepStringHandle</a> function.)

dwTime

Type: **DWORD**

The Windows time at which the action specified by the **fsAction** member takes place. Windows time is the number of milliseconds that have elapsed since the system was booted.

hsz

Type: **HSZ**

A handle to the string. Because string handles are local to the process, this member is a global atom.

hTask

Type: **HANDLE**

A handle to the task (application instance) performing the action on the string handle.

str

Type: **TCHAR[1]**

Pointer to the string identified by the **hsz** member.

## Remarks

### NOTE

The `ddeml.h` header defines `MONHSZSTRUCT` 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

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

## See also

### Conceptual

[Dynamic Data Exchange Management Library](#)

[MONCBSTRUCT](#)

[MONCONVSTRUCT](#)

[MONERRSTRUCT](#)

[MONLINKSTRUCT](#)

[MONMSGSTRUCT](#)

### Reference



# MONLINKSTRUCT structure (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Contains information about a Dynamic Data Exchange (DDE) advise loop. A DDE monitoring application can use this structure to obtain information about an advise loop that has started or ended.

## Syntax

```
typedef struct tagMONLINKSTRUCT {
    UINT    cb;
    DWORD   dwTime;
    HANDLE  hTask;
    BOOL    fEstablished;
    BOOL    fNoData;
    HSZ     hszSvc;
    HSZ     hszTopic;
    HSZ     hszItem;
    UINT    wFmt;
    BOOL    fServer;
    HCONV   hConvServer;
    HCONV   hConvClient;
} MONLINKSTRUCT, *PMONLINKSTRUCT;
```

## Members

cb

Type: **UINT**

The structure's size, in bytes.

dwTime

Type: **DWORD**

The Windows time at which the advise loop was started or ended. Windows time is the number of milliseconds that have elapsed since the system was booted.

hTask

Type: **HANDLE**

A handle to a task (application instance) that is a partner in the advise loop.

fEstablished

Type: **BOOL**

Indicates whether an advise loop was successfully established. A value of **TRUE** indicates an advise loop was established; **FALSE** indicates it was not.

fNoData

Type: **BOOL**

Indicates whether the **XTYPF\_NODATA** flag is set for the advise loop. A value of **TRUE** indicates the flag is set; **FALSE** indicates it is not.

hszSvc

Type: **HSZ**

A handle to the service name of the server in the advise loop.

hszTopic

Type: **HSZ**

A handle to the topic name on which the advise loop is established.

hszItem

Type: **HSZ**

A handle to the item name that is the subject of the advise loop.

wFmt

Type: **UINT**

The format of the data exchanged (if any) during the advise loop.

fServer

Type: **BOOL**

Indicates whether the link notification came from the server. A value of **TRUE** indicates the notification came from the server; **FALSE** indicates otherwise.

hConvServer

Type: **HCONV**

A handle to the server conversation.

hConvClient

Type: **HCONV**

A handle to the client conversation.

## Remarks

Because string handles are local to the process, the **hszSvc**, **hszTopic**, and **hszItem** members are global atoms.

The **hConvClient** and **hConvServer** members of the **MONLINKSTRUCT** structure do not hold the same value as would be seen by the applications engaged in the conversation. Instead, they hold a globally unique pair of values that identify the conversation.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Header</b>	ddeml.h (include Windows.h)

# See also

## Conceptual

[Dynamic Data Exchange Management Library](#)

[MONCBSTRUCT](#)

[MONERRSTRUCT](#)

[MONHSZSTRUCT](#)

[MONMSGSTRUCT](#)

## Reference

# MONMSGSTRUCT structure (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Contains information about a Dynamic Data Exchange (DDE) message. A DDE monitoring application can use this structure to obtain information about a DDE message that was sent or posted.

## Syntax

```
typedef struct tagMONMSGSTRUCT {
    UINT          cb;
    HWND          hwndTo;
    DWORD         dwTime;
    HANDLE        hTask;
    UINT          wMsg;
    WPARAM        wParam;
    LPARAM        lParam;
    DDEML_MSG_HOOK_DATA dmhd;
} MONMSGSTRUCT, *PMONMSGSTRUCT;
```

## Members

**cb**

Type: **UINT**

The structure's size, in bytes.

**hwndTo**

Type: **HWND**

A handle to the window that receives the DDE message.

**dwTime**

Type: **DWORD**

The Windows time at which the message was sent or posted. Windows time is the number of milliseconds that have elapsed since the system was booted.

**hTask**

Type: **HANDLE**

A handle to the task (application instance) containing the window that receives the DDE message.

**wMsg**

Type: **UINT**

The identifier of the DDE message.

**wParam**

Type: **WPARAM**

The **wParam** parameter of the DDE message.

`lParam`

Type: **LPARAM**

The **lParam** parameter of the DDE message.

`dmhd`

Type: **DDEML\_MSG\_HOOK\_DATA**

Additional information about the DDE message.

## Requirements

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

## See also

### Conceptual

[DDEML\\_MSG\\_HOOK\\_DATA](#)

[Dynamic Data Exchange Management Library](#)

[MONCBSTRUCT](#)

[MONCONVSTRUCT](#)

[MONERRSTRUCT](#)

[MONHSZSTRUCT](#)

[MONLINKSTRUCT](#)

### Reference

# PFNCALLBACK callback function (ddeml.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

An application-defined callback function used with the [Dynamic Data Exchange Management Library](#) (DDEML) functions. It processes Dynamic Data Exchange (DDE) transactions. The **PFNCALLBACK** type defines a pointer to this callback function. *DdeCallback* is a placeholder for the application-defined function name.

## Syntax

```
PFNCALLBACK Pfncallback;  
  
HDDDEDATA Pfncallback(  
    UINT wType,  
    UINT wFmt,  
    HCONV hConv,  
    HSZ hsz1,  
    HSZ hsz2,  
    HDDDEDATA hData,  
    ULONG_PTR dwData1,  
    ULONG_PTR dwData2  
)  
{...}
```

## Parameters

**wType**

Type: **UINT**

The type of the current transaction. This parameter consists of a combination of transaction class flags and transaction type flags. The following table describes each of the transaction classes and provides a list of the transaction types in each class. For information about a specific transaction type, see the individual description of that type in **Remarks**.

**wFmt**

Type: **UINT**

The format in which data is sent or received.

**hConv**

Type: **HCONV**

A handle to the conversation associated with the current transaction.

**hsz1**

Type: **HSZ**

A handle to a string. The meaning of this parameter depends on the type of the current transaction. For the meaning of this parameter, see the description of the transaction type in **Remarks**.

**hsz2**

Type: **HSZ**

A handle to a string. The meaning of this parameter depends on the type of the current transaction. For the meaning of this parameter, see the description of the transaction type in **Remarks**.

hData

Type: **HDEDDATA**

A handle to DDE data. The meaning of this parameter depends on the type of the current transaction. For the meaning of this parameter, see the description of the transaction type in **Remarks**.

dwData1

Type: **ULONG\_PTR**

Transaction-specific data. For the meaning of this parameter, see the description of the transaction type in **Remarks**.

dwData2

Type: **ULONG\_PTR**

Transaction-specific data. For the meaning of this parameter, see the description of the transaction type in **Remarks**.

## Return value

Type: **HDEDDATA**

The return value depends on the transaction class. For more information about the return values, see descriptions of the individual transaction types.

## Remarks

### **XCLASS\_BOOL**

A DDE callback function should return **TRUE** or **FALSE** when it finishes processing a transaction that belongs to this class. The **XCLASS\_BOOL** transaction class consists of the following types:

- [XTYP\\_ADVSTART](#)
- [XTYP\\_CONNECT](#)

### **XCLASS\_DATA**

A DDE callback function should return a DDE handle, the **CBR\_BLOCK** return code, or **NULL** when it finishes processing a transaction that belongs to this class. The **XCLASS\_DATA** transaction class consists of the following types:

- [XTYP\\_ADVREQ](#)
- [XTYP\\_REQUEST](#)
- [XTYP\\_WILDCONNECT](#)

### **XCLASS\_FLAGS**

A DDE callback function should return **DDE\_FACK**, **DDE\_FBUSY**, or **DDE\_FNOTPROCESSED** when it finishes processing a transaction that belongs to this class. The **XCLASS\_FLAGS** transaction class consists of the following types:

- [XTYP\\_ADVDATA](#)
- [XTYP\\_EXECUTE](#)
- [XTYP\\_POKE](#)

## XCLASS\_NOTIFICATION

The transaction types that belong to this class are for notification purposes only. The return value from the callback function is ignored. The **XCLASS\_NOTIFICATION** transaction class consists of the following types:

- [XTYP\\_ADVSTOP](#)
- [XTYP\\_CONNECT\\_CONFIRM](#)
- [XTYP\\_DISCONNECT](#)
- [XTYP\\_ERROR](#)
- [XTYP\\_MONITOR](#)
- [XTYP\\_REGISTER](#)
- [XTYP\\_XACT\\_COMPLETE](#)
- [XTYP\\_UNREGISTER](#)

The callback function is called asynchronously for transactions that do not involve the creation or termination of conversations. An application that does not frequently accept incoming messages will have reduced DDE performance because the Dynamic Data Exchange Management Library (DDEML) uses messages to initiate transactions.

An application must register the callback function by specifying a pointer to the function in a call to the [DdeInitialize](#) function.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	ddeml.h (include Windows.h)

## See also

### Conceptual

[DdeEnableCallback](#)

[DdeInitialize](#)

[Dynamic Data Exchange Management Library](#)



# winbase.h header

1/15/2021 • 37 minutes to read • [Edit Online](#)

This header is used by Backup. For more information, see:

- [Backup](#) winbase.h contains the following programming interfaces:

## Functions

TITLE	DESCRIPTION
<a href="#">_lclose</a>	The _lclose function closes the specified file so that it is no longer available for reading or writing. This function is provided for compatibility with 16-bit versions of Windows. Win32-based applications should use the CloseHandle function.
<a href="#">_lcreat</a>	Creates or opens the specified file.
<a href="#">_llseek</a>	Repositions the file pointer for the specified file.
<a href="#">_lopen</a>	The _lopen function opens an existing file and sets the file pointer to the beginning of the file. This function is provided for compatibility with 16-bit versions of Windows. Win32-based applications should use the CreateFile function.
<a href="#">_lread</a>	The _lread function reads data from the specified file. This function is provided for compatibility with 16-bit versions of Windows. Win32-based applications should use the ReadFile function.
<a href="#">_lwrite</a>	Writes data to the specified file.
<a href="#">AccessCheckAndAuditAlarmA</a>	Determines whether a security descriptor grants a specified set of access rights to the client being impersonated by the calling thread.
<a href="#">AccessCheckByTypeAndAuditAlarmA</a>	Determines whether a security descriptor grants a specified set of access rights to the client being impersonated by the calling thread.
<a href="#">AccessCheckByTypeResultListAndAuditAlarmA</a>	Determines whether a security descriptor grants a specified set of access rights to the client being impersonated by the calling thread.
<a href="#">AccessCheckByTypeResultListAndAuditAlarmByHandleA</a>	Determines whether a security descriptor grants a specified set of access rights to the client that the calling thread is impersonating.
<a href="#">ActivateActCtx</a>	The ActivateActCtx function activates the specified activation context.

TITLE	DESCRIPTION
<a href="#">AddAtomA</a>	Adds a character string to the local atom table and returns a unique value (an atom) identifying the string.
<a href="#">AddAtomW</a>	Adds a character string to the local atom table and returns a unique value (an atom) identifying the string.
<a href="#">AddConditionalAce</a>	Adds a conditional access control entry (ACE) to the specified access control list (ACL).
<a href="#">AddIntegrityLabelToBoundaryDescriptor</a>	Adds a new required security identifier (SID) to the specified boundary descriptor.
<a href="#">AddRefActCtx</a>	The AddRefActCtx function increments the reference count of the specified activation context.
<a href="#">AddSecureMemoryCacheCallback</a>	Registers a callback function to be called when a secured memory range is freed or its protections are changed.
<a href="#">ApplicationRecoveryFinished</a>	Indicates that the calling application has completed its data recovery.
<a href="#">ApplicationRecoveryInProgress</a>	Indicates that the calling application is continuing to recover data.
<a href="#">BackupEventLogA</a>	Saves the specified event log to a backup file.
<a href="#">BackupEventLogW</a>	Saves the specified event log to a backup file.
<a href="#">BackupRead</a>	Back up a file or directory, including the security information.
<a href="#">BackupSeek</a>	Seeks forward in a data stream initially accessed by using the BackupRead or BackupWrite function.
<a href="#">BackupWrite</a>	Restore a file or directory that was backed up using BackupRead.
<a href="#">BeginUpdateResourceA</a>	Retrieves a handle that can be used by the UpdateResource function to add, delete, or replace resources in a binary module.
<a href="#">BeginUpdateResourceW</a>	Retrieves a handle that can be used by the UpdateResource function to add, delete, or replace resources in a binary module.
<a href="#">BindIoCompletionCallback</a>	Associates the I/O completion port owned by the thread pool with the specified file handle. On completion of an I/O request involving this file, a non-I/O worker thread will execute the specified callback function.
<a href="#">BuildCommDCBA</a>	Fills a specified DCB structure with values specified in a device-control string.

TITLE	DESCRIPTION
<a href="#">BuildCommDCBAndTimeoutsA</a>	Translates a device-definition string into appropriate device-control block codes and places them into a device control block.
<a href="#">BuildCommDCBAndTimeoutsW</a>	Translates a device-definition string into appropriate device-control block codes and places them into a device control block.
<a href="#">BuildCommDCBW</a>	Fills a specified DCB structure with values specified in a device-control string.
<a href="#">CallNamedPipeA</a>	Connects to a message-type pipe (and waits if an instance of the pipe is not available), writes to and reads from the pipe, and then closes the pipe.
<a href="#">CheckNameLegalDOS8Dot3A</a>	Determines whether the specified name can be used to create a file on a FAT file system.
<a href="#">CheckNameLegalDOS8Dot3W</a>	Determines whether the specified name can be used to create a file on a FAT file system.
<a href="#">ClearCommBreak</a>	Restores character transmission for a specified communications device and places the transmission line in a nonbreak state.
<a href="#">ClearCommError</a>	Retrieves information about a communications error and reports the current status of a communications device.
<a href="#">ClearEventLogA</a>	Clears the specified event log, and optionally saves the current copy of the log to a backup file.
<a href="#">ClearEventLogW</a>	Clears the specified event log, and optionally saves the current copy of the log to a backup file.
<a href="#">CloseEncryptedFileRaw</a>	Closes an encrypted file after a backup or restore operation, and frees associated system resources.
<a href="#">CloseEventLog</a>	Closes the specified event log.
<a href="#">CommConfigDialogA</a>	Displays a driver-supplied configuration dialog box.
<a href="#">CommConfigDialogW</a>	Displays a driver-supplied configuration dialog box.
<a href="#">ConvertFiberToThread</a>	Converts the current fiber into a thread.
<a href="#">ConvertThreadToFiber</a>	Converts the current thread into a fiber. You must convert a thread into a fiber before you can schedule other fibers.
<a href="#">ConvertThreadToFiberEx</a>	Converts the current thread into a fiber. You must convert a thread into a fiber before you can schedule other fibers.
<a href="#">CopyContext</a>	Copies a source context structure (including any XState) onto an initialized destination context structure.

TITLE	DESCRIPTION
<a href="#">CopyFile</a>	Copies an existing file to a new file.
<a href="#">CopyFile2</a>	Copies an existing file to a new file, notifying the application of its progress through a callback function.
<a href="#">CopyFileA</a>	Copies an existing file to a new file.
<a href="#">CopyFileExA</a>	Copies an existing file to a new file, notifying the application of its progress through a callback function.
<a href="#">CopyFileExW</a>	Copies an existing file to a new file, notifying the application of its progress through a callback function.
<a href="#">CopyFileTransactedA</a>	Copies an existing file to a new file as a transacted operation, notifying the application of its progress through a callback function.
<a href="#">CopyFileTransactedW</a>	Copies an existing file to a new file as a transacted operation, notifying the application of its progress through a callback function.
<a href="#">CopyFileW</a>	Copies an existing file to a new file.
<a href="#">CreateActCtxA</a>	The CreateActCtx function creates an activation context.
<a href="#">CreateActCtxW</a>	The CreateActCtx function creates an activation context.
<a href="#">CreateBoundaryDescriptorA</a>	Creates a boundary descriptor.
<a href="#">CreateDirectory</a>	Creates a new directory.
<a href="#">CreateDirectoryExA</a>	Creates a new directory with the attributes of a specified template directory.
<a href="#">CreateDirectoryExW</a>	Creates a new directory with the attributes of a specified template directory.
<a href="#">CreateDirectoryTransactedA</a>	Creates a new directory as a transacted operation, with the attributes of a specified template directory.
<a href="#">CreateDirectoryTransactedW</a>	Creates a new directory as a transacted operation, with the attributes of a specified template directory.
<a href="#">CreateFiber</a>	Allocates a fiber object, assigns it a stack, and sets up execution to begin at the specified start address, typically the fiber function. This function does not schedule the fiber.
<a href="#">CreateFiberEx</a>	Allocates a fiber object, assigns it a stack, and sets up execution to begin at the specified start address, typically the fiber function. This function does not schedule the fiber.
<a href="#">CreateFileMappingA</a>	Creates or opens a named or unnamed file mapping object for a specified file.

TITLE	DESCRIPTION
CreateFileMappingNumaA	Creates or opens a named or unnamed file mapping object for a specified file and specifies the NUMA node for the physical memory.
CreateFileTransactedA	Creates or opens a file, file stream, or directory as a transacted operation.
CreateFileTransactedW	Creates or opens a file, file stream, or directory as a transacted operation.
CreateHardLinkA	Establishes a hard link between an existing file and a new file.
CreateHardLinkTransactedA	Establishes a hard link between an existing file and a new file as a transacted operation.
CreateHardLinkTransactedW	Establishes a hard link between an existing file and a new file as a transacted operation.
CreateHardLinkW	Establishes a hard link between an existing file and a new file.
CreateJobObjectA	Creates or opens a job object.
CreateMailslotA	Creates a mailslot with the specified name and returns a handle that a mailslot server can use to perform operations on the mailslot.
CreateMailslotW	Creates a mailslot with the specified name and returns a handle that a mailslot server can use to perform operations on the mailslot.
CreateNamedPipeA	Creates an instance of a named pipe and returns a handle for subsequent pipe operations.
CreatePrivateNamespaceA	Creates a private namespace.
CreateProcessWithLogonW	Creates a new process and its primary thread. Then the new process runs the specified executable file in the security context of the specified credentials (user, domain, and password). It can optionally load the user profile for a specified user.
CreateProcessWithTokenW	Creates a new process and its primary thread. The new process runs in the security context of the specified token. It can optionally load the user profile for the specified user.
CreateSemaphoreA	Creates or opens a named or unnamed semaphore object.
CreateSemaphoreExA	Creates or opens a named or unnamed semaphore object and returns a handle to the object.
CreateSymbolicLinkA	Creates a symbolic link.
CreateSymbolicLinkTransactedA	Creates a symbolic link as a transacted operation.

TITLE	DESCRIPTION
CreateSymbolicLinkTransactedW	Creates a symbolic link as a transacted operation.
CreateSymbolicLinkW	Creates a symbolic link.
CreateTapePartition	Reformats a tape.
CreateUmsCompletionList	Creates a user-mode scheduling (UMS) completion list.
CreateUmsThreadContext	Creates a user-mode scheduling (UMS) thread context to represent a UMS worker thread.
DeactivateActCtx	The DeactivateActCtx function deactivates the activation context corresponding to the specified cookie.
DebugBreakProcess	Causes a breakpoint exception to occur in the specified process. This allows the calling thread to signal the debugger to handle the exception.
DebugSetProcessKillOnExit	Sets the action to be performed when the calling thread exits.
DecryptFileA	Decrypts an encrypted file or directory.
DecryptFileW	Decrypts an encrypted file or directory.
DefineDosDeviceA	Defines, redefines, or deletes MS-DOS device names.
DeleteAtom	Decrements the reference count of a local string atom. If the atom's reference count is reduced to zero, DeleteAtom removes the string associated with the atom from the local atom table.
DeleteFiber	Deletes an existing fiber.
DeleteFile	Deletes an existing file.
DeleteFileTransactedA	Deletes an existing file as a transacted operation.
DeleteFileTransactedW	Deletes an existing file as a transacted operation.
DeleteTimerQueue	Deletes a timer queue. Any pending timers in the queue are canceled and deleted.
DeleteUmsCompletionList	Deletes the specified user-mode scheduling (UMS) completion list. The list must be empty.
DeleteUmsThreadContext	Deletes the specified user-mode scheduling (UMS) thread context. The thread must be terminated.
DeleteVolumeMountPointA	Deletes a drive letter or mounted folder.

TITLE	DESCRIPTION
<a href="#">DequeueUmsCompletionListItems</a>	Retrieves user-mode scheduling (UMS) worker threads from the specified UMS completion list.
<a href="#">DeregisterEventSource</a>	Closes the specified event log.
<a href="#">DestroyThreadpoolEnvironment</a>	Deletes the specified callback environment. Call this function when the callback environment is no longer needed for creating new thread pool objects.
<a href="#">DisableThreadProfiling</a>	Disables thread profiling.
<a href="#">DnsHostnameToComputerNameA</a>	Converts a DNS-style host name to a NetBIOS-style computer name.
<a href="#">DnsHostnameToComputerNameW</a>	Converts a DNS-style host name to a NetBIOS-style computer name.
<a href="#">DosDateTimeToFileTime</a>	Converts MS-DOS date and time values to a file time.
<a href="#">EnableThreadProfiling</a>	Enables thread profiling on the specified thread.
<a href="#">EncryptFileA</a>	Encrypts a file or directory.
<a href="#">EncryptFileW</a>	Encrypts a file or directory.
<a href="#">EndUpdateResourceA</a>	Commits or discards changes made prior to a call to UpdateResource.
<a href="#">EndUpdateResourceW</a>	Commits or discards changes made prior to a call to UpdateResource.
<a href="#">EnterUmsSchedulingMode</a>	Converts the calling thread into a user-mode scheduling (UMS) scheduler thread.
<a href="#">EnumResourceLanguagesA</a>	Enumerates language-specific resources, of the specified type and name, associated with a binary module.
<a href="#">EnumResourceLanguagesW</a>	Enumerates language-specific resources, of the specified type and name, associated with a binary module.
<a href="#">EnumResourceNamesA</a>	Enumerates resources of a specified type within a binary module.
<a href="#">EnumResourceTypesA</a>	Enumerates resource types within a binary module.
<a href="#">EnumResourceTypesW</a>	Enumerates resource types within a binary module.
<a href="#">EraseTape</a>	Erases all or part of a tape.
<a href="#">EscapeCommFunction</a>	Directs the specified communications device to perform an extended function.

TITLE	DESCRIPTION
<a href="#">ExecuteUmsThread</a>	Runs the specified UMS worker thread.
<a href="#">FatalExit</a>	Transfers execution control to the debugger. The behavior of the debugger thereafter is specific to the type of debugger used.
<a href="#">FileEncryptionStatusA</a>	Retrieves the encryption status of the specified file.
<a href="#">FileEncryptionStatusW</a>	Retrieves the encryption status of the specified file.
<a href="#">FileTimeToDosDateTime</a>	Converts a file time to MS-DOS date and time values.
<a href="#">FindActCtxSectionGuid</a>	The FindActCtxSectionGuid function retrieves information on a specific GUID in the current activation context and returns a ACTCTX_SECTION_KEYED_DATA structure.
<a href="#">FindActCtxSectionStringA</a>	The FindActCtxSectionString function retrieves information on a specific string in the current activation context and returns a ACTCTX_SECTION_KEYED_DATA structure.
<a href="#">FindActCtxSectionStringW</a>	The FindActCtxSectionString function retrieves information on a specific string in the current activation context and returns a ACTCTX_SECTION_KEYED_DATA structure.
<a href="#">FindAtomA</a>	Searches the local atom table for the specified character string and retrieves the atom associated with that string.
<a href="#">FindAtomW</a>	Searches the local atom table for the specified character string and retrieves the atom associated with that string.
<a href="#">FindFirstFileNameTransactedW</a>	Creates an enumeration of all the hard links to the specified file as a transacted operation. The function returns a handle to the enumeration that can be used on subsequent calls to the FindNextFileNameW function.
<a href="#">FindFirstFileTransactedA</a>	Searches a directory for a file or subdirectory with a name that matches a specific name as a transacted operation.
<a href="#">FindFirstFileTransactedW</a>	Searches a directory for a file or subdirectory with a name that matches a specific name as a transacted operation.
<a href="#">FindFirstStreamTransactedW</a>	Enumerates the first stream in the specified file or directory as a transacted operation.
<a href="#">FindFirstVolumeA</a>	Retrieves the name of a volume on a computer.
<a href="#">FindFirstVolumeMountPointA</a>	Retrieves the name of a mounted folder on the specified volume.
<a href="#">FindFirstVolumeMountPointW</a>	Retrieves the name of a mounted folder on the specified volume.
<a href="#">FindNextVolumeA</a>	Continues a volume search started by a call to the FindFirstVolume function.



TITLE	DESCRIPTION
<a href="#">FindNextVolumeMountPointA</a>	Continues a mounted folder search started by a call to the FindFirstVolumeMountPoint function.
<a href="#">FindNextVolumeMountPointW</a>	Continues a mounted folder search started by a call to the FindFirstVolumeMountPoint function.
<a href="#">FindResourceA</a>	Determines the location of a resource with the specified type and name in the specified module.
<a href="#">FindResourceExA</a>	Determines the location of the resource with the specified type, name, and language in the specified module.
<a href="#">FindVolumeMountPointClose</a>	Closes the specified mounted folder search handle.
<a href="#">FormatMessage</a>	Formats a message string.
<a href="#">FormatMessageA</a>	Formats a message string.
<a href="#">FormatMessageW</a>	Formats a message string.
<a href="#">GetActiveProcessorCount</a>	Returns the number of active processors in a processor group or in the system.
<a href="#">GetActiveProcessorGroupCount</a>	Returns the number of active processor groups in the system.
<a href="#">GetApplicationRecoveryCallback</a>	Retrieves a pointer to the callback routine registered for the specified process. The address returned is in the virtual address space of the process.
<a href="#">GetApplicationRestartSettings</a>	Retrieves the restart information registered for the specified process.
<a href="#">GetAtomNameA</a>	Retrieves a copy of the character string associated with the specified local atom.
<a href="#">GetAtomNameW</a>	Retrieves a copy of the character string associated with the specified local atom.
<a href="#">GetBinaryTypeA</a>	Determines whether a file is an executable (.exe) file, and if so, which subsystem runs the executable file.
<a href="#">GetBinaryTypeW</a>	Determines whether a file is an executable (.exe) file, and if so, which subsystem runs the executable file.
<a href="#">GetCommConfig</a>	Retrieves the current configuration of a communications device.
<a href="#">GetCommMask</a>	Retrieves the value of the event mask for a specified communications device.
<a href="#">GetCommModemStatus</a>	Retrieves the modem control-register values.

TITLE	DESCRIPTION
<a href="#">GetCommPorts</a>	Gets an array that contains the well-formed COM ports.
<a href="#">GetCommProperties</a>	Retrieves information about the communications properties for a specified communications device.
<a href="#">GetCommState</a>	Retrieves the current control settings for a specified communications device.
<a href="#">GetCommTimeouts</a>	Retrieves the time-out parameters for all read and write operations on a specified communications device.
<a href="#">GetCompressedFileSizeTransactedA</a>	Retrieves the actual number of bytes of disk storage used to store a specified file as a transacted operation.
<a href="#">GetCompressedFileSizeTransactedW</a>	Retrieves the actual number of bytes of disk storage used to store a specified file as a transacted operation.
<a href="#">GetComputerNameA</a>	Retrieves the NetBIOS name of the local computer. This name is established at system startup, when the system reads it from the registry.
<a href="#">GetComputerNameW</a>	Retrieves the NetBIOS name of the local computer. This name is established at system startup, when the system reads it from the registry.
<a href="#">GetCurrentActCtx</a>	The GetCurrentActCtx function returns the handle to the active activation context of the calling thread.
<a href="#">GetCurrentDirectory</a>	Retrieves the current directory for the current process.
<a href="#">GetCurrentHwProfileA</a>	Retrieves information about the current hardware profile for the local computer.
<a href="#">GetCurrentHwProfileW</a>	Retrieves information about the current hardware profile for the local computer.
<a href="#">GetCurrentUmsThread</a>	Returns the user-mode scheduling (UMS) thread context of the calling UMS thread.
<a href="#">GetDefaultCommConfigA</a>	Retrieves the default configuration for the specified communications device.
<a href="#">GetDefaultCommConfigW</a>	Retrieves the default configuration for the specified communications device.
<a href="#">GetDevicePowerState</a>	Retrieves the current power state of the specified device.
<a href="#">GetDllDirectoryA</a>	Retrieves the application-specific portion of the search path used to locate DLLs for the application.
<a href="#">GetDllDirectoryW</a>	Retrieves the application-specific portion of the search path used to locate DLLs for the application.

TITLE	DESCRIPTION
<a href="#">GetEnabledXStateFeatures</a>	Gets a mask of enabled XState features on x86 or x64 processors.
<a href="#">GetEnvironmentVariable</a>	Retrieves the contents of the specified variable from the environment block of the calling process.
<a href="#">GetEventLogInformation</a>	Retrieves information about the specified event log.
<a href="#">GetFileAttributesTransactedA</a>	Retrieves file system attributes for a specified file or directory as a transacted operation.
<a href="#">GetFileAttributesTransactedW</a>	Retrieves file system attributes for a specified file or directory as a transacted operation.
<a href="#">GetFileBandwidthReservation</a>	Retrieves the bandwidth reservation properties of the volume on which the specified file resides.
<a href="#">GetFileInformationByHandleEx</a>	Retrieves file information for the specified file.
<a href="#">GetFileSecurityA</a>	Obtains specified information about the security of a file or directory. The information obtained is constrained by the caller's access rights and privileges.
<a href="#">GetFirmwareEnvironmentVariableA</a>	Retrieves the value of the specified firmware environment variable.
<a href="#">GetFirmwareEnvironmentVariableExA</a>	Retrieves the value of the specified firmware environment variable and its attributes.
<a href="#">GetFirmwareEnvironmentVariableExW</a>	Retrieves the value of the specified firmware environment variable and its attributes.
<a href="#">GetFirmwareEnvironmentVariableW</a>	Retrieves the value of the specified firmware environment variable.
<a href="#">GetFirmwareType</a>	Retrieves the firmware type of the local computer.
<a href="#">GetFullPathNameTransactedA</a>	Retrieves the full path and file name of the specified file as a transacted operation.
<a href="#">GetFullPathNameTransactedW</a>	Retrieves the full path and file name of the specified file as a transacted operation.
<a href="#">GetLogicalDriveStringsA</a>	Fills a buffer with strings that specify valid drives in the system.
<a href="#">GetLongPathNameTransactedA</a>	Converts the specified path to its long form as a transacted operation.
<a href="#">GetLongPathNameTransactedW</a>	Converts the specified path to its long form as a transacted operation.
<a href="#">GetMailslotInfo</a>	Retrieves information about the specified mailslot.

TITLE	DESCRIPTION
<a href="#">GetMaximumProcessorCount</a>	Returns the maximum number of logical processors that a processor group or the system can have.
<a href="#">GetMaximumProcessorGroupCount</a>	Returns the maximum number of processor groups that the system can have.
<a href="#">GetNamedPipeClientComputerNameA</a>	Retrieves the client computer name for the specified named pipe.
<a href="#">GetNamedPipeClientProcessId</a>	Retrieves the client process identifier for the specified named pipe.
<a href="#">GetNamedPipeClientSessionId</a>	Retrieves the client session identifier for the specified named pipe.
<a href="#">GetNamedPipeHandleStateA</a>	Retrieves information about a specified named pipe.
<a href="#">GetNamedPipeServerProcessId</a>	Retrieves the server process identifier for the specified named pipe.
<a href="#">GetNamedPipeServerSessionId</a>	Retrieves the server session identifier for the specified named pipe.
<a href="#">GetNextUmsListItem</a>	Returns the next user-mode scheduling (UMS) thread context in a list of thread contexts.
<a href="#">GetNumaAvailableMemoryNode</a>	Retrieves the amount of memory available in the specified node.
<a href="#">GetNumaAvailableMemoryNodeEx</a>	Retrieves the amount of memory that is available in a node specified as a USHORT value.
<a href="#">GetNumaNodeNumberFromHandle</a>	Retrieves the NUMA node associated with the file or I/O device represented by the specified file handle.
<a href="#">GetNumaNodeProcessorMask</a>	Retrieves the processor mask for the specified node.
<a href="#">GetNumaProcessorNode</a>	Retrieves the node number for the specified processor.
<a href="#">GetNumaProcessorNodeEx</a>	Retrieves the node number as a USHORT value for the specified logical processor.
<a href="#">GetNumaProximityNode</a>	Retrieves the NUMA node number that corresponds to the specified proximity domain identifier.
<a href="#">GetNumberOfEventLogRecords</a>	Retrieves the number of records in the specified event log.
<a href="#">GetOldestEventLogRecord</a>	Retrieves the absolute record number of the oldest record in the specified event log.
<a href="#">GetPrivateProfileInt</a>	Retrieves an integer associated with a key in the specified section of an initialization file.

TITLE	DESCRIPTION
<a href="#">GetPrivateProfileIntA</a>	Retrieves an integer associated with a key in the specified section of an initialization file.
<a href="#">GetPrivateProfileIntW</a>	Retrieves an integer associated with a key in the specified section of an initialization file.
<a href="#">GetPrivateProfileSection</a>	Retrieves all the keys and values for the specified section of an initialization file.
<a href="#">GetPrivateProfileSectionA</a>	Retrieves all the keys and values for the specified section of an initialization file.
<a href="#">GetPrivateProfileSectionNames</a>	Retrieves the names of all sections in an initialization file.
<a href="#">GetPrivateProfileSectionNamesA</a>	Retrieves the names of all sections in an initialization file.
<a href="#">GetPrivateProfileSectionNamesW</a>	Retrieves the names of all sections in an initialization file.
<a href="#">GetPrivateProfileSectionW</a>	Retrieves all the keys and values for the specified section of an initialization file.
<a href="#">GetPrivateProfileString</a>	Retrieves a string from the specified section in an initialization file.
<a href="#">GetPrivateProfileStringA</a>	Retrieves a string from the specified section in an initialization file.
<a href="#">GetPrivateProfileStringW</a>	Retrieves a string from the specified section in an initialization file.
<a href="#">GetPrivateProfileStruct</a>	Retrieves the data associated with a key in the specified section of an initialization file.
<a href="#">GetPrivateProfileStructA</a>	Retrieves the data associated with a key in the specified section of an initialization file.
<a href="#">GetPrivateProfileStructW</a>	Retrieves the data associated with a key in the specified section of an initialization file.
<a href="#">GetProcessAffinityMask</a>	Retrieves the process affinity mask for the specified process and the system affinity mask for the system.
<a href="#">GetProcessDEPPolicy</a>	Gets the data execution prevention (DEP) and DEP-ATL thunk emulation settings for the specified 32-bit process.Windows XP with SP3: Gets the DEP and DEP-ATL thunk emulation settings for the current process.
<a href="#">GetProcessIoCounters</a>	Retrieves accounting information for all I/O operations performed by the specified process.
<a href="#">GetProcessWorkingSetSize</a>	Retrieves the minimum and maximum working set sizes of the specified process.

TITLE	DESCRIPTION
<a href="#">GetProfileIntA</a>	Retrieves an integer from a key in the specified section of the Win.ini file.
<a href="#">GetProfileIntW</a>	Retrieves an integer from a key in the specified section of the Win.ini file.
<a href="#">GetProfileSectionA</a>	Retrieves all the keys and values for the specified section of the Win.ini file.
<a href="#">GetProfileSectionW</a>	Retrieves all the keys and values for the specified section of the Win.ini file.
<a href="#">GetProfileStringA</a>	Retrieves the string associated with a key in the specified section of the Win.ini file.
<a href="#">GetProfileStringW</a>	Retrieves the string associated with a key in the specified section of the Win.ini file.
<a href="#">GetShortPathNameA</a>	Retrieves the short path form of the specified path.
<a href="#">GetSystemDEPPolicy</a>	Gets the data execution prevention (DEP) policy setting for the system.
<a href="#">GetSystemPowerStatus</a>	Retrieves the power status of the system. The status indicates whether the system is running on AC or DC power, whether the battery is currently charging, how much battery life remains, and if battery saver is on or off.
<a href="#">GetSystemRegistryQuota</a>	Retrieves the current size of the registry and the maximum size that the registry is allowed to attain on the system.
<a href="#">GetTapeParameters</a>	Retrieves information that describes the tape or the tape drive.
<a href="#">GetTapePosition</a>	Retrieves the current address of the tape, in logical or absolute blocks.
<a href="#">GetTapeStatus</a>	Determines whether the tape device is ready to process tape commands.
<a href="#">GetTempFileName</a>	Creates a name for a temporary file. If a unique file name is generated, an empty file is created and the handle to it is released; otherwise, only a file name is generated.
<a href="#">GetThreadSelectorEntry</a>	Retrieves a descriptor table entry for the specified selector and thread.
<a href="#">GetUmsCompletionListEvent</a>	Retrieves a handle to the event associated with the specified user-mode scheduling (UMS) completion list.
<a href="#">GetUmsSystemThreadInformation</a>	Queries whether the specified thread is a UMS scheduler thread, a UMS worker thread, or a non-UMS thread.

TITLE	DESCRIPTION
<a href="#">GetUserNameA</a>	Retrieves the name of the user associated with the current thread.
<a href="#">GetUserNameW</a>	Retrieves the name of the user associated with the current thread.
<a href="#">GetVolumeNameForVolumeMountPointA</a>	Retrieves a volume GUID path for the volume that is associated with the specified volume mount point ( drive letter, volume GUID path, or mounted folder).
<a href="#">GetVolumePathNameA</a>	Retrieves the volume mount point where the specified path is mounted.
<a href="#">GetVolumePathNamesForVolumeNameA</a>	Retrieves a list of drive letters and mounted folder paths for the specified volume.
<a href="#">GetXStateFeaturesMask</a>	Returns the mask of XState features set within a CONTEXT structure.
<a href="#">GlobalAddAtomA</a>	Adds a character string to the global atom table and returns a unique value (an atom) identifying the string.
<a href="#">GlobalAddAtomExA</a>	Adds a character string to the global atom table and returns a unique value (an atom) identifying the string.
<a href="#">GlobalAddAtomExW</a>	Adds a character string to the global atom table and returns a unique value (an atom) identifying the string.
<a href="#">GlobalAddAtomW</a>	Adds a character string to the global atom table and returns a unique value (an atom) identifying the string.
<a href="#">GlobalAlloc</a>	Allocates the specified number of bytes from the heap.
<a href="#">GlobalDeleteAtom</a>	Decrements the reference count of a global string atom. If the atom's reference count reaches zero, GlobalDeleteAtom removes the string associated with the atom from the global atom table.
<a href="#">GlobalDiscard</a>	Discards the specified global memory block.
<a href="#">GlobalFindAtomA</a>	Searches the global atom table for the specified character string and retrieves the global atom associated with that string.
<a href="#">GlobalFindAtomW</a>	Searches the global atom table for the specified character string and retrieves the global atom associated with that string.
<a href="#">GlobalFlags</a>	Retrieves information about the specified global memory object.
<a href="#">GlobalFree</a>	Frees the specified global memory object and invalidates its handle.

TITLE	DESCRIPTION
<a href="#">GlobalGetAtomNameA</a>	Retrieves a copy of the character string associated with the specified global atom.
<a href="#">GlobalGetAtomNameW</a>	Retrieves a copy of the character string associated with the specified global atom.
<a href="#">GlobalHandle</a>	Retrieves the handle associated with the specified pointer to a global memory block.
<a href="#">GlobalLock</a>	Locks a global memory object and returns a pointer to the first byte of the object's memory block.
<a href="#">GlobalMemoryStatus</a>	Retrieves information about the system's current usage of both physical and virtual memory.
<a href="#">GlobalReAlloc</a>	Changes the size or attributes of a specified global memory object. The size can increase or decrease.
<a href="#">GlobalSize</a>	Retrieves the current size of the specified global memory object, in bytes.
<a href="#">GlobalUnlock</a>	Decrements the lock count associated with a memory object that was allocated with GMEM_MOVEABLE.
<a href="#">HasOverlappedIoCompleted</a>	Provides a high performance test operation that can be used to poll for the completion of an outstanding I/O operation.
<a href="#">InitAtomTable</a>	Initializes the local atom table and sets the number of hash buckets to the specified size.
<a href="#">InitializeContext</a>	Initializes a CONTEXT structure inside a buffer with the necessary size and alignment.
<a href="#">InitializeContext2</a>	Initializes a CONTEXT structure inside a buffer with the necessary size and alignment, with the option to specify an XSTATE compaction mask.
<a href="#">InitializeThreadpoolEnvironment</a>	Initializes a callback environment.
<a href="#">InterlockedExchangeSubtract</a>	Performs an atomic subtraction of two values.
<a href="#">IsBadCodePtr</a>	Determines whether the calling process has read access to the memory at the specified address.
<a href="#">IsBadReadPtr</a>	Verifies that the calling process has read access to the specified range of memory.
<a href="#">IsBadStringPtrA</a>	Verifies that the calling process has read access to the specified range of memory.
<a href="#">IsBadStringPtrW</a>	Verifies that the calling process has read access to the specified range of memory.



TITLE	DESCRIPTION
<a href="#">IsBadWritePtr</a>	Verifies that the calling process has write access to the specified range of memory.
<a href="#">IsNativeVhdBoot</a>	Indicates if the OS was booted from a VHD container.
<a href="#">IsSystemResumeAutomatic</a>	Determines the current state of the computer.
<a href="#">IsTextUnicode</a>	Determines if a buffer is likely to contain a form of Unicode text.
<a href="#">LoadModule</a>	Loads and executes an application or creates a new instance of an existing application.
<a href="#">LoadPackagedLibrary</a>	Loads the specified packaged module and its dependencies into the address space of the calling process.
<a href="#">LocalAlloc</a>	Allocates the specified number of bytes from the heap.
<a href="#">LocalFlags</a>	Retrieves information about the specified local memory object.
<a href="#">LocalFree</a>	Frees the specified local memory object and invalidates its handle.
<a href="#">LocalHandle</a>	Retrieves the handle associated with the specified pointer to a local memory object.
<a href="#">LocalLock</a>	Locks a local memory object and returns a pointer to the first byte of the object's memory block.
<a href="#">LocalReAlloc</a>	Changes the size or the attributes of a specified local memory object. The size can increase or decrease.
<a href="#">LocalSize</a>	Retrieves the current size of the specified local memory object, in bytes.
<a href="#">LocalUnlock</a>	Decrements the lock count associated with a memory object that was allocated with LMEM_MOVEABLE.
<a href="#">LocateXStateFeature</a>	Retrieves a pointer to the processor state for an XState feature within a CONTEXT structure.
<a href="#">LogonUserA</a>	The Win32 LogonUser function attempts to log a user on to the local computer. LogonUser returns a handle to a user token that you can use to impersonate user.
<a href="#">LogonUserExA</a>	The LogonUserEx function attempts to log a user on to the local computer.
<a href="#">LogonUserExW</a>	The LogonUserEx function attempts to log a user on to the local computer.

TITLE	DESCRIPTION
<a href="#">LogonUserW</a>	The Win32 LogonUser function attempts to log a user on to the local computer. LogonUser returns a handle to a user token that you can use to impersonate user.
<a href="#">LookupAccountNameA</a>	Accepts the name of a system and an account as input. It retrieves a security identifier (SID) for the account and the name of the domain on which the account was found.
<a href="#">LookupAccountNameW</a>	Accepts the name of a system and an account as input. It retrieves a security identifier (SID) for the account and the name of the domain on which the account was found.
<a href="#">LookupAccountSidA</a>	Accepts a security identifier (SID) as input. It retrieves the name of the account for this SID and the name of the first domain on which this SID is found.
<a href="#">LookupAccountSidLocalA</a>	Retrieves the name of the account for the specified SID on the local machine.
<a href="#">LookupAccountSidLocalW</a>	Retrieves the name of the account for the specified SID on the local machine.
<a href="#">LookupAccountSidW</a>	Accepts a security identifier (SID) as input. It retrieves the name of the account for this SID and the name of the first domain on which this SID is found.
<a href="#">LookupPrivilegeDisplayNameA</a>	Retrieves the display name that represents a specified privilege.
<a href="#">LookupPrivilegeDisplayNameW</a>	Retrieves the display name that represents a specified privilege.
<a href="#">LookupPrivilegeNameA</a>	Retrieves the name that corresponds to the privilege represented on a specific system by a specified locally unique identifier (LUID).
<a href="#">LookupPrivilegeNameW</a>	Retrieves the name that corresponds to the privilege represented on a specific system by a specified locally unique identifier (LUID).
<a href="#">LookupPrivilegeValueA</a>	Retrieves the locally unique identifier (LUID) used on a specified system to locally represent the specified privilege name.
<a href="#">LookupPrivilegeValueW</a>	Retrieves the locally unique identifier (LUID) used on a specified system to locally represent the specified privilege name.
<a href="#">lstrcatA</a>	Appends one string to another.Warning Do not use.
<a href="#">lstrcatW</a>	Appends one string to another.Warning Do not use.
<a href="#">lstrcmpA</a>	Compares two character strings. The comparison is case-sensitive.

TITLE	DESCRIPTION
<a href="#">lstrcmpiA</a>	Compares two character strings. The comparison is not case-sensitive.
<a href="#">lstrcmpiW</a>	Compares two character strings. The comparison is not case-sensitive.
<a href="#">lstrcmpW</a>	Compares two character strings. The comparison is case-sensitive.
<a href="#">lstrcpyA</a>	Copies a string to a buffer.
<a href="#">lstrcpynA</a>	Copies a specified number of characters from a source string into a buffer.Warning Do not use.
<a href="#">lstrcpynW</a>	Copies a specified number of characters from a source string into a buffer.Warning Do not use.
<a href="#">lstrcpyW</a>	Copies a string to a buffer.
<a href="#">lstrlenA</a>	Determines the length of the specified string (not including the terminating null character).
<a href="#">lstrlenW</a>	Determines the length of the specified string (not including the terminating null character).
<a href="#">MAKEINTATOM</a>	Converts the specified atom into a string, so it can be passed to functions which accept either atoms or strings.
<a href="#">MapUserPhysicalPagesScatter</a>	Maps previously allocated physical memory pages at a specified address in an Address Windowing Extensions (AWE) region.
<a href="#">MapViewOfFileExNuma</a>	Maps a view of a file mapping into the address space of a calling process and specifies the NUMA node for the physical memory.
<a href="#">MoveFile</a>	Moves an existing file or a directory, including its children.
<a href="#">MoveFileA</a>	Moves an existing file or a directory, including its children.
<a href="#">MoveFileExA</a>	Moves an existing file or directory, including its children, with various move options.
<a href="#">MoveFileExW</a>	Moves an existing file or directory, including its children, with various move options.
<a href="#">MoveFileTransactedA</a>	Moves an existing file or a directory, including its children, as a transacted operation.
<a href="#">MoveFileTransactedW</a>	Moves an existing file or a directory, including its children, as a transacted operation.
<a href="#">MoveFileW</a>	Moves an existing file or a directory, including its children.

TITLE	DESCRIPTION
<a href="#">MoveFileWithProgressA</a>	Moves a file or directory, including its children. You can provide a callback function that receives progress notifications.
<a href="#">MoveFileWithProgressW</a>	Moves a file or directory, including its children. You can provide a callback function that receives progress notifications.
<a href="#">MulDiv</a>	Multiplies two 32-bit values and then divides the 64-bit result by a third 32-bit value.
<a href="#">NotifyChangeEventLog</a>	Enables an application to receive notification when an event is written to the specified event log.
<a href="#">ObjectCloseAuditAlarmA</a>	Generates an audit message in the security event log when a handle to a private object is deleted.
<a href="#">ObjectDeleteAuditAlarmA</a>	Generates audit messages when an object is deleted.
<a href="#">ObjectOpenAuditAlarmA</a>	Generates audit messages when a client application attempts to gain access to an object or to create a new one.
<a href="#">ObjectPrivilegeAuditAlarmA</a>	Generates an audit message in the security event log.
<a href="#">OpenBackupEventLogA</a>	Opens a handle to a backup event log created by the BackupEventLog function.
<a href="#">OpenBackupEventLogW</a>	Opens a handle to a backup event log created by the BackupEventLog function.
<a href="#">OpenCommPort</a>	Attempts to open a communication device.
<a href="#">OpenEncryptedFileRawA</a>	Opens an encrypted file in order to backup (export) or restore (import) the file.
<a href="#">OpenEncryptedFileRawW</a>	Opens an encrypted file in order to backup (export) or restore (import) the file.
<a href="#">OpenEventLogA</a>	Opens a handle to the specified event log.
<a href="#">OpenEventLogW</a>	Opens a handle to the specified event log.
<a href="#">OpenFile</a>	Creates, opens, reopens, or deletes a file.
<a href="#">OpenFileById</a>	Opens the file that matches the specified identifier.
<a href="#">OpenFileMappingA</a>	Opens a named file mapping object.
<a href="#">OpenJobObjectA</a>	Opens an existing job object.
<a href="#">OpenPrivateNamespaceA</a>	Opens a private namespace.

TITLE	DESCRIPTION
<a href="#">OperationEnd</a>	Notifies the system that the application is about to end an operation.
<a href="#">OperationStart</a>	Notifies the system that the application is about to start an operation.
<a href="#">PowerClearRequest</a>	Decrements the count of power requests of the specified type for a power request object.
<a href="#">PowerCreateRequest</a>	Creates a new power request object.
<a href="#">PowerSetRequest</a>	Increments the count of power requests of the specified type for a power request object.
<a href="#">PrepareTape</a>	Prepares the tape to be accessed or removed.
<a href="#">PrivilegedServiceAuditAlarmA</a>	Generates an audit message in the security event log.
<a href="#">PulseEvent</a>	Sets the specified event object to the signaled state and then resets it to the nonsignaled state after releasing the appropriate number of waiting threads.
<a href="#">PurgeComm</a>	Discards all characters from the output or input buffer of a specified communications resource. It can also terminate pending read or write operations on the resource.
<a href="#">QueryActCtxSettingsW</a>	The QueryActCtxSettingsW function specifies the activation context, and the namespace and name of the attribute that is to be queried.
<a href="#">QueryActCtxW</a>	The QueryActCtxW function queries the activation context.
<a href="#">QueryDosDeviceA</a>	Retrieves information about MS-DOS device names.
<a href="#">QueryFullProcessImageNameA</a>	Retrieves the full name of the executable image for the specified process.
<a href="#">QueryFullProcessImageNameW</a>	Retrieves the full name of the executable image for the specified process.
<a href="#">QueryThreadProfiling</a>	Determines whether thread profiling is enabled for the specified thread.
<a href="#">QueryUmsThreadInformation</a>	Retrieves information about the specified user-mode scheduling (UMS) worker thread.
<a href="#">ReadDirectoryChangesExW</a>	Retrieves information that describes the changes within the specified directory, which can include extended information if that information type is specified.
<a href="#">ReadDirectoryChangesW</a>	Retrieves information that describes the changes within the specified directory.

TITLE	DESCRIPTION
<a href="#">ReadEncryptedFileRaw</a>	Backs up (export) encrypted files.
<a href="#">ReadEventLogA</a>	Reads the specified number of entries from the specified event log.
<a href="#">ReadEventLogW</a>	Reads the specified number of entries from the specified event log.
<a href="#">ReadThreadProfilingData</a>	Reads the specified profiling data associated with the thread.
<a href="#">RegisterApplicationRecoveryCallback</a>	Registers the active instance of an application for recovery.
<a href="#">RegisterApplicationRestart</a>	Registers the active instance of an application for restart.
<a href="#">RegisterEventSourceA</a>	Retrieves a registered handle to the specified event log.
<a href="#">RegisterEventSourceW</a>	Retrieves a registered handle to the specified event log.
<a href="#">RegisterWaitForSingleObject</a>	Directs a wait thread in the thread pool to wait on the object.
<a href="#">ReleaseActCtx</a>	The ReleaseActCtx function decrements the reference count of the specified activation context.
<a href="#">RemoveDirectoryTransactedA</a>	Deletes an existing empty directory as a transacted operation.
<a href="#">RemoveDirectoryTransactedW</a>	Deletes an existing empty directory as a transacted operation.
<a href="#">RemoveSecureMemoryCacheCallback</a>	Unregisters a callback function that was previously registered with the AddSecureMemoryCacheCallback function.
<a href="#">ReOpenFile</a>	Reopens the specified file system object with different access rights, sharing mode, and flags.
<a href="#">ReplaceFileA</a>	Replaces one file with another file, with the option of creating a backup copy of the original file.
<a href="#">ReplaceFileW</a>	Replaces one file with another file, with the option of creating a backup copy of the original file.
<a href="#">ReportEventA</a>	Writes an entry at the end of the specified event log.
<a href="#">ReportEventW</a>	Writes an entry at the end of the specified event log.
<a href="#">RequestWakeupLatency</a>	Has no effect and returns STATUS_NOT_SUPPORTED. This function is provided only for compatibility with earlier versions of Windows.Windows Server 2008 and Windows Vista: Has no effect and always returns success.

TITLE	DESCRIPTION
<a href="#">SetCommBreak</a>	Suspends character transmission for a specified communications device and places the transmission line in a break state until the ClearCommBreak function is called.
<a href="#">SetCommConfig</a>	Sets the current configuration of a communications device.
<a href="#">SetCommMask</a>	Specifies a set of events to be monitored for a communications device.
<a href="#">SetCommState</a>	Configures a communications device according to the specifications in a device-control block (a DCB structure). The function reinitializes all hardware and control settings, but it does not empty output or input queues.
<a href="#">SetCommTimeouts</a>	Sets the time-out parameters for all read and write operations on a specified communications device.
<a href="#">SetCurrentDirectory</a>	Changes the current directory for the current process.
<a href="#">SetDefaultCommConfigA</a>	Sets the default configuration for a communications device.
<a href="#">SetDefaultCommConfigW</a>	Sets the default configuration for a communications device.
<a href="#">SetDllDirectoryA</a>	Adds a directory to the search path used to locate DLLs for the application.
<a href="#">SetDllDirectoryW</a>	Adds a directory to the search path used to locate DLLs for the application.
<a href="#">SetEnvironmentVariable</a>	Sets the contents of the specified environment variable for the current process.
<a href="#">SetFileAttributesTransactedA</a>	Sets the attributes for a file or directory as a transacted operation.
<a href="#">SetFileAttributesTransactedW</a>	Sets the attributes for a file or directory as a transacted operation.
<a href="#">SetFileBandwidthReservation</a>	Requests that bandwidth for the specified file stream be reserved. The reservation is specified as a number of bytes in a period of milliseconds for I/O requests on the specified file handle.
<a href="#">SetFileCompletionNotificationModes</a>	Sets the notification modes for a file handle, allowing you to specify how completion notifications work for the specified file.
<a href="#">SetFileSecurityA</a>	Sets the security of a file or directory object.
<a href="#">SetFileShortNameA</a>	Sets the short name for the specified file.
<a href="#">SetFileShortNameW</a>	Sets the short name for the specified file.

TITLE	DESCRIPTION
<a href="#">SetFirmwareEnvironmentVariableA</a>	Sets the value of the specified firmware environment variable.
<a href="#">SetFirmwareEnvironmentVariableExA</a>	Sets the value of the specified firmware environment variable as the attributes that indicate how this variable is stored and maintained.
<a href="#">SetFirmwareEnvironmentVariableExW</a>	Sets the value of the specified firmware environment variable and the attributes that indicate how this variable is stored and maintained.
<a href="#">SetFirmwareEnvironmentVariableW</a>	Sets the value of the specified firmware environment variable.
<a href="#">SetHandleCount</a>	
<a href="#">SetMailslotInfo</a>	Sets the time-out value used by the specified mailslot for a read operation.
<a href="#">SetProcessAffinityMask</a>	Sets a processor affinity mask for the threads of the specified process.
<a href="#">SetProcessDEPPolicy</a>	Changes data execution prevention (DEP) and DEP-ATL thunk emulation settings for a 32-bit process.
<a href="#">SetProcessWorkingSetSize</a>	Sets the minimum and maximum working set sizes for the specified process.
<a href="#">SetSearchPathMode</a>	Sets the per-process mode that the SearchPath function uses when locating files.
<a href="#">SetSystemPowerState</a>	Suspends the system by shutting power down. Depending on the ForceFlag parameter, the function either suspends operation immediately or requests permission from all applications and device drivers before doing so.
<a href="#">SetTapeParameters</a>	Specifies the block size of a tape or configures the tape device.
<a href="#">SetTapePosition</a>	Sets the tape position on the specified device.
<a href="#">SetThreadAffinityMask</a>	Sets a processor affinity mask for the specified thread.
<a href="#">SetThreadExecutionState</a>	Enables an application to inform the system that it is in use, thereby preventing the system from entering sleep or turning off the display while the application is running.
<a href="#">SetThreadpoolCallbackCleanupGroup</a>	Associates the specified cleanup group with the specified callback environment.
<a href="#">SetThreadpoolCallbackLibrary</a>	Ensures that the specified DLL remains loaded as long as there are outstanding callbacks.
<a href="#">SetThreadpoolCallbackPersistent</a>	Specifies that the callback should run on a persistent thread.



TITLE	DESCRIPTION
<a href="#">SetThreadPoolCallbackPool</a>	Sets the thread pool to be used when generating callbacks.
<a href="#">SetThreadPoolCallbackPriority</a>	Specifies the priority of a callback function relative to other work items in the same thread pool.
<a href="#">SetThreadPoolCallbackRunsLong</a>	Indicates that callbacks associated with this callback environment may not return quickly.
<a href="#">SetUmsThreadInformation</a>	Sets application-specific context information for the specified user-mode scheduling (UMS) worker thread.
<a href="#">SetupComm</a>	Initializes the communications parameters for a specified communications device.
<a href="#">SetVolumeLabelA</a>	Sets the label of a file system volume.
<a href="#">SetVolumeLabelW</a>	Sets the label of a file system volume.
<a href="#">SetVolumeMountPointA</a>	Associates a volume with a drive letter or a directory on another volume.
<a href="#">SetVolumeMountPointW</a>	Associates a volume with a drive letter or a directory on another volume.
<a href="#">SetXStateFeaturesMask</a>	Sets the mask of XState features set within a CONTEXT structure.
<a href="#">SwitchToFiber</a>	Schedules a fiber. The function must be called on a fiber.
<a href="#">TransmitCommChar</a>	Transmits a specified character ahead of any pending data in the output buffer of the specified communications device.
<a href="#">UmsThreadYield</a>	Yields control to the user-mode scheduling (UMS) scheduler thread on which the calling UMS worker thread is running.
<a href="#">UnregisterApplicationRecoveryCallback</a>	Removes the active instance of an application from the recovery list.
<a href="#">UnregisterApplicationRestart</a>	Removes the active instance of an application from the restart list.
<a href="#">UnregisterWait</a>	Cancels a registered wait operation issued by the RegisterWaitForSingleObject function.
<a href="#">UpdateResourceA</a>	Adds, deletes, or replaces a resource in a portable executable (PE) file.
<a href="#">UpdateResourceW</a>	Adds, deletes, or replaces a resource in a portable executable (PE) file.
<a href="#">VerifyVersionInfoA</a>	Compares a set of operating system version requirements to the corresponding values for the currently running version of the system.

TITLE	DESCRIPTION
<a href="#">VerifyVersionInfoW</a>	Compares a set of operating system version requirements to the corresponding values for the currently running version of the system.
<a href="#">WaitCommEvent</a>	Waits for an event to occur for a specified communications device. The set of events that are monitored by this function is contained in the event mask associated with the device handle.
<a href="#">WaitNamedPipeA</a>	Waits until either a time-out interval elapses or an instance of the specified named pipe is available for connection (that is, the pipe's server process has a pending ConnectNamedPipe operation on the pipe).
<a href="#">WinExec</a>	Runs the specified application.
<a href="#">WinMain</a>	The user-provided entry point for a graphical Windows-based application.
<a href="#">Wow64EnableWow64FsRedirection</a>	Enables or disables file system redirection for the calling thread.
<a href="#">Wow64GetThreadSelectorEntry</a>	Retrieves a descriptor table entry for the specified selector and WOW64 thread.
<a href="#">WriteEncryptedFileRaw</a>	Restores (import) encrypted files.
<a href="#">WritePrivateProfileSectionA</a>	Replaces the keys and values for the specified section in an initialization file.
<a href="#">WritePrivateProfileSectionW</a>	Replaces the keys and values for the specified section in an initialization file.
<a href="#">WritePrivateProfileStringA</a>	Copies a string into the specified section of an initialization file.
<a href="#">WritePrivateProfileStringW</a>	Copies a string into the specified section of an initialization file.
<a href="#">WritePrivateProfileStructA</a>	Copies data into a key in the specified section of an initialization file. As it copies the data, the function calculates a checksum and appends it to the end of the data.
<a href="#">WritePrivateProfileStructW</a>	Copies data into a key in the specified section of an initialization file. As it copies the data, the function calculates a checksum and appends it to the end of the data.
<a href="#">WriteProfileSectionA</a>	Replaces the contents of the specified section in the Win.ini file with specified keys and values.
<a href="#">WriteProfileSectionW</a>	Replaces the contents of the specified section in the Win.ini file with specified keys and values.
<a href="#">WriteProfileStringA</a>	Copies a string into the specified section of the Win.ini file.

TITLE	DESCRIPTION
<a href="#">WriteProfileStringW</a>	Copies a string into the specified section of the Win.ini file.
<a href="#">WriteTapemark</a>	Writes a specified number of filemarks, setmarks, short filemarks, or long filemarks to a tape device.
<a href="#">WTSGetActiveConsoleSessionId</a>	Retrieves the session identifier of the console session.
<a href="#">ZombifyActCtx</a>	The ZombifyActCtx function deactivates the specified activation context, but does not deallocate it.

## Callback functions

TITLE	DESCRIPTION
<a href="#">LPPROGRESS_ROUTINE</a>	An application-defined callback function used with the CopyFileEx, MoveFileTransacted, and MoveFileWithProgress functions.
<a href="#">PCOPYFILE2_PROGRESS_ROUTINE</a>	An application-defined callback function used with the CopyFile2 function.
<a href="#">PFE_EXPORT_FUNC</a>	An application-defined callback function used with ReadEncryptedFileRaw.
<a href="#">PFE_IMPORT_FUNC</a>	An application-defined callback function used with WriteEncryptedFileRaw. The system calls ImportCallback one or more times, each time to retrieve a portion of a backup file's data.
<a href="#">PFIBER_START_ROUTINE</a>	An application-defined function used with the CreateFiber function. It serves as the starting address for a fiber.

## Structures

TITLE	DESCRIPTION
<a href="#">ACTCTX_SECTION_KEYED_DATA</a>	The ACTCTX_SECTION_KEYED_DATA structure is used by the FindActCtxSectionString and FindActCtxSectionGuid functions to return the activation context information along with either the GUID or 32-bit integer-tagged activation context section.
<a href="#">ACTCTXA</a>	The ACTCTX structure is used by the CreateActCtx function to create the activation context.
<a href="#">ACTCTXW</a>	The ACTCTX structure is used by the CreateActCtx function to create the activation context.
<a href="#">COMMCONFIG</a>	Contains information about the configuration state of a communications device.
<a href="#">COMMPROP</a>	Contains information about a communications driver.

TITLE	DESCRIPTION
COMMTIMEOUTS	Contains the time-out parameters for a communications device.
COMSTAT	Contains information about a communications device.
COPYFILE2_EXTENDED_PARAMETERS	Contains extended parameters for the CopyFile2 function.
COPYFILE2_MESSAGE	Passed to the CopyFile2ProgressRoutine callback function with information about a pending copy operation.
DCB	Defines the control setting for a serial communications device.
EVENTLOG_FULL_INFORMATION	Indicates whether the event log is full.
FILE_ALIGNMENT_INFO	Contains alignment information for a file.
FILE_ALLOCATION_INFO	Contains the total number of bytes that should be allocated for a file.
FILE_ATTRIBUTE_TAG_INFO	Receives the requested file attribute information. Used for any handles.
FILE_BASIC_INFO	Contains the basic information for a file. Used for file handles.
FILE_COMPRESSION_INFO	Receives file compression information.
FILE_DISPOSITION_INFO	Indicates whether a file should be deleted. Used for any handles.
FILE_END_OF_FILE_INFO	Contains the specified value to which the end of the file should be set.
FILE_FULL_DIR_INFO	Contains directory information for a file.
FILE_ID_BOTH_DIR_INFO	Contains information about files in the specified directory.
FILE_ID_DESCRIPTOR	Specifies the type of ID that is being used.
FILE_ID_EXTD_DIR_INFO	Contains identification information for a file.
FILE_ID_INFO	Contains identification information for a file.
FILE_IO_PRIORITY_HINT_INFO	Specifies the priority hint for a file I/O operation.
FILE_NAME_INFO	Receives the file name.
FILE_REMOTE_PROTOCOL_INFO	Contains file remote protocol information.
FILE_RENAME_INFO	Contains the name to which the file should be renamed.

TITLE	DESCRIPTION
FILE_STANDARD_INFO	Receives extended information for the file.
FILE_STORAGE_INFO	Contains directory information for a file.
FILE_STREAM_INFO	Receives file stream information for the specified file.
HW_PROFILE_INFOA	Contains information about a hardware profile.
HW_PROFILE_INFOW	Contains information about a hardware profile.
MEMORYSTATUS	Contains information about the current state of both physical and virtual memory.
OFSTRUCT	Contains information about a file that the OpenFile function opened or attempted to open.
OPERATION_END_PARAMETERS	This structure is used by the OperationEnd function.
OPERATION_START_PARAMETERS	This structure is used by the OperationStart function.
STARTUPINFOEXA	Specifies the window station, desktop, standard handles, and attributes for a new process. It is used with the CreateProcess and CreateProcessAsUser functions.
STARTUPINFOEXW	Specifies the window station, desktop, standard handles, and attributes for a new process. It is used with the CreateProcess and CreateProcessAsUser functions.
SYSTEM_POWER_STATUS	Contains information about the power status of the system.
UMS_SCHEDULER_STARTUP_INFO	Specifies attributes for a user-mode scheduling (UMS) scheduler thread.
UMS_SYSTEM_THREAD_INFORMATION	Specifies a UMS scheduler thread, UMS worker thread, or non-UMS thread. The GetUmsSystemThreadInformation function uses this structure.
WIN32_STREAM_ID	Contains stream data.

## Enumerations

TITLE	DESCRIPTION
COPYFILE2_COPY_PHASE	Indicates the phase of a copy at the time of an error.
COPYFILE2_MESSAGE_ACTION	Returned by the CopyFile2ProgressRoutine callback function to indicate what action should be taken for the pending copy operation.
COPYFILE2_MESSAGE_TYPE	Indicates the type of message passed in the COPYFILE2_MESSAGE structure to the CopyFile2ProgressRoutine callback function.

TITLE	DESCRIPTION
FILE_ID_TYPE	Discriminator for the union in the FILE_ID_DESCRIPTOR structure.
PRIORITY_HINT	Defines values that are used with the FILE_IO_PRIORITY_HINT_INFO structure to specify the priority hint for a file I/O operation.

# AddAtomA function (winbase.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Adds a character string to the local atom table and returns a unique value (an atom) identifying the string.

## Syntax

```
ATOM AddAtomA(  
    LPCSTR lpString  
);
```

## Parameters

`lpString`

Type: LPCTSTR

The null-terminated string to be added. The string can have a maximum size of 255 bytes. Strings differing only in case are considered identical. The case of the first string added is preserved and returned by the [GetAtomName](#) function.

Alternatively, you can use an integer atom that has been converted using the [MAKEINTATOM](#) macro. See the Remarks for more information.

## Return value

Type: ATOM

If the function succeeds, the return value is the newly created atom.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

## Remarks

The **AddAtom** function stores no more than one copy of a given string in the atom table. If the string is already in the table, the function returns the existing atom and, in the case of a string atom, increments the string's reference count.

If *lpString* has the form "#1234", **AddAtom** returns an integer atom whose value is the 16-bit representation of the decimal number specified in the string (0x04D2, in this example). If the decimal value specified is 0x0000 or is greater than or equal to 0xC000, the return value is zero, indicating an error. If *lpString* was created by the [MAKEINTATOM](#) macro, the low-order word must be in the range 0x0001 through 0xBFFF. If the low-order word is not in this range, the function fails.

If *lpString* has any other form, **AddAtom** returns a string atom.

#### NOTE

The winbase.h header defines AddAtom 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

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

## See also

[DeleteAtom](#)

[FindAtom](#)

[GetAtomName](#)

[GlobalAddAtom](#)

[GlobalDeleteAtom](#)

[GlobalFindAtom](#)

[GlobalGetAtomName](#)

[MAKEINTATOM](#)

### Reference



# AddAtomW function (winbase.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Adds a character string to the local atom table and returns a unique value (an atom) identifying the string.

## Syntax

```
ATOM AddAtomW(  
    LPCWSTR lpString  
);
```

## Parameters

`lpString`

Type: LPCTSTR

The null-terminated string to be added. The string can have a maximum size of 255 bytes. Strings differing only in case are considered identical. The case of the first string added is preserved and returned by the [GetAtomName](#) function.

Alternatively, you can use an integer atom that has been converted using the [MAKEINTATOM](#) macro. See the Remarks for more information.

## Return value

Type: ATOM

If the function succeeds, the return value is the newly created atom.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

## Remarks

The **AddAtom** function stores no more than one copy of a given string in the atom table. If the string is already in the table, the function returns the existing atom and, in the case of a string atom, increments the string's reference count.

If *lpString* has the form "#1234", **AddAtom** returns an integer atom whose value is the 16-bit representation of the decimal number specified in the string (0x04D2, in this example). If the decimal value specified is 0x0000 or is greater than or equal to 0xC000, the return value is zero, indicating an error. If *lpString* was created by the [MAKEINTATOM](#) macro, the low-order word must be in the range 0x0001 through 0xBFFF. If the low-order word is not in this range, the function fails.

If *lpString* has any other form, **AddAtom** returns a string atom.

#### NOTE

The winbase.h header defines AddAtom 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

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

## See also

[DeleteAtom](#)

[FindAtom](#)

[GetAtomName](#)

[GlobalAddAtom](#)

[GlobalDeleteAtom](#)

[GlobalFindAtom](#)

[GlobalGetAtomName](#)

[MAKEINTATOM](#)

### Reference

# DeleteAtom function (winbase.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Decrements the reference count of a local string atom. If the atom's reference count is reduced to zero, **DeleteAtom** removes the string associated with the atom from the local atom table.

## Syntax

```
ATOM DeleteAtom(  
    ATOM nAtom  
);
```

## Parameters

**nAtom**

Type: **ATOM**

The atom to be deleted.

## Return value

Type: **ATOM**

If the function succeeds, the return value is zero.

If the function fails, the return value is the *nAtom* parameter. To get extended error information, call [GetLastError](#).

## Remarks

A string atom's reference count specifies the number of times the atom has been added to the atom table. The [AddAtom](#) function increments the count on each call. The **DeleteAtom** function decrements the count on each call but removes the string only if the atom's reference count is zero.

Each call to [AddAtom](#) should have a corresponding call to **DeleteAtom**. Do not call **DeleteAtom** more times than you call **AddAtom**, or you may delete the atom while other clients are using it.

The **DeleteAtom** function has no effect on an integer atom (an atom whose value is in the range 0x0001 to 0xBFFF). The function always returns zero for an integer atom.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows

<b>Header</b>	winbase.h (include Windows.h)
<b>Library</b>	Kernel32.lib
<b>DLL</b>	Kernel32.dll

# See also

[AddAtom](#)

[FindAtom](#)

[GlobalAddAtom](#)

[GlobalDeleteAtom](#)

[GlobalFindAtom](#)

[MAKEINTATOM](#)

## Reference

# FindAtomA function (winbase.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Searches the local atom table for the specified character string and retrieves the atom associated with that string.

## Syntax

```
ATOM FindAtomA(  
    LPCSTR lpString  
);
```

## Parameters

`lpString`

Type: LPCTSTR

The character string for which to search.

Alternatively, you can use an integer atom that has been converted using the [MAKEINTATOM](#) macro. See [Remarks](#) for more information.

## Return value

Type: **ATOM**

If the function succeeds, the return value is the atom associated with the given string.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

## Remarks

Even though the system preserves the case of a string in an atom table, the search performed by the **FindAtom** function is not case sensitive.

If *lpString* was created by the [MAKEINTATOM](#) macro, the low-order word must be in the range 0x0001 through 0xBFFF. If the low-order word is not in this range, the function fails.

### NOTE

The winbase.h header defines FindAtom 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

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

[AddAtom](#)

[DeleteAtom](#)

[GlobalAddAtom](#)

[GlobalDeleteAtom](#)

[GlobalFindAtom](#)

## Reference

# FindAtomW function (winbase.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Searches the local atom table for the specified character string and retrieves the atom associated with that string.

## Syntax

```
ATOM FindAtomW(  
    LPCWSTR lpString  
);
```

## Parameters

`lpString`

Type: LPCTSTR

The character string for which to search.

Alternatively, you can use an integer atom that has been converted using the [MAKEINTATOM](#) macro. See [Remarks](#) for more information.

## Return value

Type: **ATOM**

If the function succeeds, the return value is the atom associated with the given string.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

## Remarks

Even though the system preserves the case of a string in an atom table, the search performed by the **FindAtom** function is not case sensitive.

If *lpString* was created by the [MAKEINTATOM](#) macro, the low-order word must be in the range 0x0001 through 0xBFFF. If the low-order word is not in this range, the function fails.

### NOTE

The winbase.h header defines FindAtom 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

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

[AddAtom](#)

[DeleteAtom](#)

[GlobalAddAtom](#)

[GlobalDeleteAtom](#)

[GlobalFindAtom](#)

## Reference



# GetAtomNameA function (winbase.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Retrieves a copy of the character string associated with the specified local atom.

## Syntax

```
UINT GetAtomNameA(  
    ATOM  nAtom,  
    LPSTR lpBuffer,  
    int   nSize  
);
```

## Parameters

**nAtom**

Type: **ATOM**

The local atom that identifies the character string to be retrieved.

**lpBuffer**

Type: **LPTSTR**

The character string.

**nSize**

Type: **int**

The size, in characters, of the buffer.

## Return value

Type: **UINT**

If the function succeeds, the return value is the length of the string copied to the buffer, in characters, not including the terminating null character.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

## Remarks

The string returned for an integer atom (an atom whose value is in the range 0x0001 to 0xBFFF) is a null-terminated string in which the first character is a pound sign (#) and the remaining characters represent the unsigned integer atom value.

### Security Considerations

Using this function incorrectly might compromise the security of your program. Incorrect use of this function includes not correctly specifying the size of the *lpBuffer* parameter.

#### NOTE

The winbase.h header defines GetAtomName 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

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

## See also

[AddAtom](#)

[DeleteAtom](#)

[FindAtom](#)

[GlobalAddAtom](#)

[GlobalDeleteAtom](#)

[GlobalFindAtom](#)

[GlobalGetAtomName](#)

[MAKEINTATOM](#)

### Reference

# GetAtomNameW function (winbase.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Retrieves a copy of the character string associated with the specified local atom.

## Syntax

```
UINT GetAtomNameW(  
    ATOM    nAtom,  
    LPWSTR  lpBuffer,  
    int     nSize  
);
```

## Parameters

**nAtom**

Type: **ATOM**

The local atom that identifies the character string to be retrieved.

**lpBuffer**

Type: **LPTSTR**

The character string.

**nSize**

Type: **int**

The size, in characters, of the buffer.

## Return value

Type: **UINT**

If the function succeeds, the return value is the length of the string copied to the buffer, in characters, not including the terminating null character.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

## Remarks

The string returned for an integer atom (an atom whose value is in the range 0x0001 to 0xBFFF) is a null-terminated string in which the first character is a pound sign (#) and the remaining characters represent the unsigned integer atom value.

### Security Considerations

Using this function incorrectly might compromise the security of your program. Incorrect use of this function includes not correctly specifying the size of the *lpBuffer* parameter.

#### NOTE

The winbase.h header defines GetAtomName 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

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

## See also

[AddAtom](#)

[DeleteAtom](#)

[FindAtom](#)

[GlobalAddAtom](#)

[GlobalDeleteAtom](#)

[GlobalFindAtom](#)

[GlobalGetAtomName](#)

[MAKEINTATOM](#)

### Reference

# GlobalAddAtomA function (winbase.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Adds a character string to the global atom table and returns a unique value (an atom) identifying the string.

## Syntax

```
ATOM GlobalAddAtomA(  
    LPCSTR lpString  
);
```

## Parameters

`lpString`

Type: LPCTSTR

The null-terminated string to be added. The string can have a maximum size of 255 bytes. Strings that differ only in case are considered identical. The case of the first string of this name added to the table is preserved and returned by the [GlobalGetAtomName](#) function.

Alternatively, you can use an integer atom that has been converted using the [MAKEINTATOM](#) macro. See the Remarks for more information.

## Return value

Type: ATOM

If the function succeeds, the return value is the newly created atom.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

## Remarks

If the string already exists in the global atom table, the atom for the existing string is returned and the atom's reference count is incremented.

The string associated with the atom is not deleted from memory until its reference count is zero. For more information, see the [GlobalDeleteAtom](#) function.

Global atoms are not deleted automatically when the application terminates. For every call to the **GlobalAddAtom** function, there must be a corresponding call to the [GlobalDeleteAtom](#) function.

If the *lpString* parameter has the form "#1234", **GlobalAddAtom** returns an integer atom whose value is the 16-bit representation of the decimal number specified in the string (0x04D2, in this example). If the decimal value specified is 0x0000 or is greater than or equal to 0xC000, the return value is zero, indicating an error. If *lpString* was created by the [MAKEINTATOM](#) macro, the low-order word must be in the range 0x0001 through 0xBFFF. If the low-order word is not in this range, the function fails.

If *lpString* has any other form, **GlobalAddAtom** returns a string atom.

#### NOTE

The winbase.h header defines GlobalAddAtom 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

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

## See also

[AddAtom](#)

[DeleteAtom](#)

[FindAtom](#)

[GetAtomName](#)

[GlobalDeleteAtom](#)

[GlobalFindAtom](#)

[GlobalGetAtomName](#)

[MAKEINTATOM](#)

### Reference

# GlobalAddAtomExA function (winbase.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Adds a character string to the global atom table and returns a unique value (an atom) identifying the string.

## Syntax

```
ATOM GlobalAddAtomExA(  
    LPCSTR lpString,  
    DWORD  Flags  
);
```

## Parameters

**lpString**

The null-terminated string to be added. The string can have a maximum size of 255 bytes. Strings that differ only in case are considered identical. The case of the first string of this name added to the table is preserved and returned by the [GlobalGetAtomName](#) function.

Alternatively, you can use an integer atom that has been converted using the [MAKEINTATOM](#) macro. See the Remarks for more information.

**Flags**

## Return value

If the function succeeds, the return value is the newly created atom.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

## Remarks

### NOTE

The winbase.h header defines GlobalAddAtomEx 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

<b>Target Platform</b>	Windows
<b>Header</b>	winbase.h (include Windows.h)
<b>Library</b>	Kernel32.lib

DLL	Kernel32.dll

## See also

[GlobalAddAtom](#)



# GlobalAddAtomExW function (winbase.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Adds a character string to the global atom table and returns a unique value (an atom) identifying the string.

## Syntax

```
ATOM GlobalAddAtomExW(  
    LPCWSTR lpString,  
    DWORD   Flags  
);
```

## Parameters

**lpString**

The null-terminated string to be added. The string can have a maximum size of 255 bytes. Strings that differ only in case are considered identical. The case of the first string of this name added to the table is preserved and returned by the [GlobalGetAtomName](#) function.

Alternatively, you can use an integer atom that has been converted using the [MAKEINTATOM](#) macro. See the Remarks for more information.

**Flags**

## Return value

If the function succeeds, the return value is the newly created atom.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

## Remarks

### NOTE

The winbase.h header defines GlobalAddAtomEx 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

<b>Target Platform</b>	Windows
<b>Header</b>	winbase.h (include Windows.h)
<b>Library</b>	Kernel32.lib

DLL	Kernel32.dll

## See also

[GlobalAddAtom](#)

# GlobalAddAtomW function (winbase.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Adds a character string to the global atom table and returns a unique value (an atom) identifying the string.

## Syntax

```
ATOM GlobalAddAtomW(  
    LPCWSTR lpString  
);
```

## Parameters

`lpString`

Type: LPCTSTR

The null-terminated string to be added. The string can have a maximum size of 255 bytes. Strings that differ only in case are considered identical. The case of the first string of this name added to the table is preserved and returned by the [GlobalGetAtomName](#) function.

Alternatively, you can use an integer atom that has been converted using the [MAKEINTATOM](#) macro. See the Remarks for more information.

## Return value

Type: ATOM

If the function succeeds, the return value is the newly created atom.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

## Remarks

If the string already exists in the global atom table, the atom for the existing string is returned and the atom's reference count is incremented.

The string associated with the atom is not deleted from memory until its reference count is zero. For more information, see the [GlobalDeleteAtom](#) function.

Global atoms are not deleted automatically when the application terminates. For every call to the **GlobalAddAtom** function, there must be a corresponding call to the [GlobalDeleteAtom](#) function.

If the *lpString* parameter has the form "#1234", **GlobalAddAtom** returns an integer atom whose value is the 16-bit representation of the decimal number specified in the string (0x04D2, in this example). If the decimal value specified is 0x0000 or is greater than or equal to 0xC000, the return value is zero, indicating an error. If *lpString* was created by the [MAKEINTATOM](#) macro, the low-order word must be in the range 0x0001 through 0xBFFF. If the low-order word is not in this range, the function fails.

If *lpString* has any other form, **GlobalAddAtom** returns a string atom.

#### NOTE

The winbase.h header defines GlobalAddAtom 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

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

## See also

[AddAtom](#)

[DeleteAtom](#)

[FindAtom](#)

[GetAtomName](#)

[GlobalDeleteAtom](#)

[GlobalFindAtom](#)

[GlobalGetAtomName](#)

[MAKEINTATOM](#)

### Reference

# GlobalDeleteAtom function (winbase.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Decrements the reference count of a global string atom. If the atom's reference count reaches zero, **GlobalDeleteAtom** removes the string associated with the atom from the global atom table.

## Syntax

```
ATOM GlobalDeleteAtom(  
    ATOM nAtom  
);
```

## Parameters

nAtom

Type: **ATOM**

The atom and character string to be deleted.

## Return value

Type: **ATOM**

The function always returns (**ATOM**) 0.

To determine whether the function has failed, call [SetLastError](#) with **ERROR\_SUCCESS** before calling **GlobalDeleteAtom**, then call [GetLastError](#). If the last error code is still **ERROR\_SUCCESS**, **GlobalDeleteAtom** has succeeded.

## Remarks

A string atom's reference count specifies the number of times the string has been added to the atom table. The [GlobalAddAtom](#) function increments the reference count of a string that already exists in the global atom table each time it is called.

Each call to [GlobalAddAtom](#) should have a corresponding call to **GlobalDeleteAtom**. Do not call **GlobalDeleteAtom** more times than you call **GlobalAddAtom**, or you may delete the atom while other clients are using it. Applications using Dynamic Data Exchange (DDE) should follow the rules on global atom management to prevent leaks and premature deletion.

**GlobalDeleteAtom** has no effect on an integer atom (an atom whose value is in the range 0x0001 to 0xBFFF). The function always returns zero for an integer atom.

### Examples

For an example, see [Initiating a Conversation](#).

## Requirements

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

[AddAtom](#)

[DeleteAtom](#)

[FindAtom](#)

[GlobalAddAtom](#)

[GlobalFindAtom](#)

[MAKEINTATOM](#)

### Reference

# GlobalFindAtomA function (winbase.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Searches the global atom table for the specified character string and retrieves the global atom associated with that string.

## Syntax

```
ATOM GlobalFindAtomA(  
    LPCSTR lpString  
);
```

## Parameters

`lpString`

Type: LPCTSTR

The null-terminated character string for which to search.

Alternatively, you can use an integer atom that has been converted using the [MAKEINTATOM](#) macro. See the Remarks for more information.

## Return value

Type: **ATOM**

If the function succeeds, the return value is the global atom associated with the given string.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

## Remarks

Even though the system preserves the case of a string in an atom table as it was originally entered, the search performed by **GlobalFindAtom** is not case sensitive.

If *lpString* was created by the [MAKEINTATOM](#) macro, the low-order word must be in the range 0x0001 through 0xBFFF. If the low-order word is not in this range, the function fails.

### NOTE

The winbase.h header defines GlobalFindAtom 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

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

[AddAtom](#)

[DeleteAtom](#)

[FindAtom](#)

[GetAtomName](#)

[GlobalAddAtom](#)

[GlobalDeleteAtom](#)

[GlobalGetAtomName](#)

### Reference



# GlobalFindAtomW function (winbase.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Searches the global atom table for the specified character string and retrieves the global atom associated with that string.

## Syntax

```
ATOM GlobalFindAtomW(  
    LPCWSTR lpString  
);
```

## Parameters

`lpString`

Type: LPCTSTR

The null-terminated character string for which to search.

Alternatively, you can use an integer atom that has been converted using the [MAKEINTATOM](#) macro. See the Remarks for more information.

## Return value

Type: **ATOM**

If the function succeeds, the return value is the global atom associated with the given string.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

## Remarks

Even though the system preserves the case of a string in an atom table as it was originally entered, the search performed by **GlobalFindAtom** is not case sensitive.

If *lpString* was created by the [MAKEINTATOM](#) macro, the low-order word must be in the range 0x0001 through 0xBFFF. If the low-order word is not in this range, the function fails.

### NOTE

The winbase.h header defines GlobalFindAtom 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

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

[AddAtom](#)

[DeleteAtom](#)

[FindAtom](#)

[GetAtomName](#)

[GlobalAddAtom](#)

[GlobalDeleteAtom](#)

[GlobalGetAtomName](#)

## Reference

# GlobalGetAtomNameA function (winbase.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Retrieves a copy of the character string associated with the specified global atom.

## Syntax

```
UINT GlobalGetAtomNameA(  
    ATOM    nAtom,  
    LPSTR   lpBuffer,  
    int     nSize  
);
```

## Parameters

**nAtom**

Type: **ATOM**

The global atom associated with the character string to be retrieved.

**lpBuffer**

Type: **LPTSTR**

The buffer for the character string.

**nSize**

Type: **int**

The size, in characters, of the buffer.

## Return value

Type: **UINT**

If the function succeeds, the return value is the length of the string copied to the buffer, in characters, not including the terminating null character.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

## Remarks

The string returned for an integer atom (an atom whose value is in the range 0x0001 to 0xBFFF) is a null-terminated string in which the first character is a pound sign (#) and the remaining characters represent the unsigned integer atom value.

### Security Considerations

Using this function incorrectly might compromise the security of your program. Incorrect use of this function includes not correctly specifying the size of the *lpBuffer* parameter. Also, note that a global atom is accessible by anyone; thus, privacy and the integrity of its contents is not assured.

#### NOTE

The winbase.h header defines GlobalGetAtomName 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

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

## See also

[AddAtom](#)

[DeleteAtom](#)

[FindAtom](#)

[GlobalAddAtom](#)

[GlobalDeleteAtom](#)

[GlobalFindAtom](#)

[MAKEINTATOM](#)

### Reference

# GlobalGetAtomNameW function (winbase.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Retrieves a copy of the character string associated with the specified global atom.

## Syntax

```
UINT GlobalGetAtomNameW(  
    ATOM    nAtom,  
    LPWSTR  lpBuffer,  
    int     nSize  
);
```

## Parameters

nAtom

Type: **ATOM**

The global atom associated with the character string to be retrieved.

lpBuffer

Type: **LPTSTR**

The buffer for the character string.

nSize

Type: **int**

The size, in characters, of the buffer.

## Return value

Type: **UINT**

If the function succeeds, the return value is the length of the string copied to the buffer, in characters, not including the terminating null character.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

## Remarks

The string returned for an integer atom (an atom whose value is in the range 0x0001 to 0xBFFF) is a null-terminated string in which the first character is a pound sign (#) and the remaining characters represent the unsigned integer atom value.

### Security Considerations

Using this function incorrectly might compromise the security of your program. Incorrect use of this function includes not correctly specifying the size of the *lpBuffer* parameter. Also, note that a global atom is accessible by anyone; thus, privacy and the integrity of its contents is not assured.

#### NOTE

The winbase.h header defines GlobalGetAtomName 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

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

## See also

[AddAtom](#)

[DeleteAtom](#)

[FindAtom](#)

[GlobalAddAtom](#)

[GlobalDeleteAtom](#)

[GlobalFindAtom](#)

[MAKEINTATOM](#)

### Reference

# InitAtomTable function (winbase.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Initializes the local atom table and sets the number of hash buckets to the specified size.

## Syntax

```
BOOL InitAtomTable(  
    DWORD nSize  
);
```

## Parameters

**nSize**

Type: **DWORD**

The number of hash buckets to use for the atom table. If this parameter is zero, the default number of hash buckets are created.

To achieve better performance, specify a prime number in *nSize*.

## Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

## Remarks

An application need not use this function to use a local atom table. The default number of hash buckets used is 37. If an application does use **InitAtomTable**, however, it should call the function before any other atom-management function.

If an application uses a large number of local atoms, it can reduce the time required to add an atom to the local atom table or to find an atom in the table by increasing the size of the table. However, this increases the amount of memory required to maintain the table.

The number of buckets in the global atom table cannot be changed. If the atom table has already been initialized, either explicitly by a prior call to **InitAtomTable**, or implicitly by the use of any atom-management function, **InitAtomTable** returns success without changing the number of hash buckets.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]

<b>Target Platform</b>	Windows
<b>Header</b>	winbase.h (include Windows.h)
<b>Library</b>	Kernel32.lib
<b>DLL</b>	Kernel32.dll

## See also

[AddAtom](#)

[DeleteAtom](#)

[FindAtom](#)

[GetAtomName](#)

[GlobalAddAtom](#)

[GlobalDeleteAtom](#)

[GlobalFindAtom](#)

[GlobalGetAtomName](#)

## Reference



# MAKEINTATOM macro (winbase.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Converts the specified atom into a string, so it can be passed to functions which accept either atoms or strings.

## Syntax

```
void MAKEINTATOM(  
    i  
);
```

## Parameters

i

The numeric value to be made into an integer atom. This parameter can be either an integer atom or a string atom.

## Return value

None

## Remarks

Although the return value of the **MAKEINTATOM** macro is cast as an **LPTSTR** value, it cannot be used as a string pointer except when it is passed to atom-management functions that require an **LPTSTR** argument.

## Requirements

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)

## See also

[AddAtom](#)

[DeleteAtom](#)

[GetAtomName](#)

[GlobalAddAtom](#)

[GlobalDeleteAtom](#)

GlobalGetAtomName

**Reference**

# wingdi.h header

1/15/2021 • 53 minutes to read • [Edit Online](#)

This header is used by Data Exchange. For more information, see:

- [Data Exchange](#) wingdi.h contains the following programming interfaces:

## Functions

TITLE	DESCRIPTION
<a href="#">AbortDoc</a>	The AbortDoc function stops the current print job and erases everything drawn since the last call to the StartDoc function.
<a href="#">AbortPath</a>	The AbortPath function closes and discards any paths in the specified device context.
<a href="#">AddFontMemResourceEx</a>	The AddFontMemResourceEx function adds the font resource from a memory image to the system.
<a href="#">AddFontResourceA</a>	The AddFontResource function adds the font resource from the specified file to the system font table. The font can subsequently be used for text output by any application.
<a href="#">AddFontResourceExA</a>	The AddFontResourceEx function adds the font resource from the specified file to the system. Fonts added with the AddFontResourceEx function can be marked as private and not enumerable.
<a href="#">AddFontResourceExW</a>	The AddFontResourceEx function adds the font resource from the specified file to the system. Fonts added with the AddFontResourceEx function can be marked as private and not enumerable.
<a href="#">AddFontResourceW</a>	The AddFontResource function adds the font resource from the specified file to the system font table. The font can subsequently be used for text output by any application.
<a href="#">AlphaBlend</a>	The AlphaBlend function displays bitmaps that have transparent or semitransparent pixels.
<a href="#">AngleArc</a>	The AngleArc function draws a line segment and an arc.
<a href="#">AnimatePalette</a>	The AnimatePalette function replaces entries in the specified logical palette.
<a href="#">Arc</a>	The Arc function draws an elliptical arc.
<a href="#">ArcTo</a>	The ArcTo function draws an elliptical arc.

TITLE	DESCRIPTION
<a href="#">BeginPath</a>	The BeginPath function opens a path bracket in the specified device context.
<a href="#">BitBlt</a>	The BitBlt function performs a bit-block transfer of the color data corresponding to a rectangle of pixels from the specified source device context into a destination device context.
<a href="#">CancelDC</a>	The CancelDC function cancels any pending operation on the specified device context (DC).
<a href="#">CheckColorsInGamut</a>	The CheckColorsInGamut function determines whether a specified set of RGB triples lies in the output gamut of a specified device. The RGB triples are interpreted in the input logical color space.
<a href="#">ChoosePixelFormat</a>	The ChoosePixelFormat function attempts to match an appropriate pixel format supported by a device context to a given pixel format specification.
<a href="#">Chord</a>	The Chord function draws a chord (a region bounded by the intersection of an ellipse and a line segment, called a secant). The chord is outlined by using the current pen and filled by using the current brush.
<a href="#">CloseEnhMetaFile</a>	The CloseEnhMetaFile function closes an enhanced-metafile device context and returns a handle that identifies an enhanced-format metafile.
<a href="#">CloseFigure</a>	The CloseFigure function closes an open figure in a path.
<a href="#">CloseMetaFile</a>	The CloseMetaFile function closes a metafile device context and returns a handle that identifies a Windows-format metafile.
<a href="#">CMYK</a>	The CMYK macro creates a CMYK color value by combining the specified cyan, magenta, yellow, and black values.
<a href="#">ColorCorrectPalette</a>	The ColorCorrectPalette function corrects the entries of a palette using the WCS 1.0 parameters in the specified device context.
<a href="#">ColorMatchToTarget</a>	The ColorMatchToTarget function enables you to preview colors as they would appear on the target device.
<a href="#">CombineRgn</a>	The CombineRgn function combines two regions and stores the result in a third region. The two regions are combined according to the specified mode.
<a href="#">CombineTransform</a>	The CombineTransform function concatenates two world-space to page-space transformations.
<a href="#">CopyEnhMetaFileA</a>	The CopyEnhMetaFile function copies the contents of an enhanced-format metafile to a specified file.

TITLE	DESCRIPTION
<a href="#">CopyEnhMetaFileW</a>	The CopyEnhMetaFile function copies the contents of an enhanced-format metafile to a specified file.
<a href="#">CopyMetaFileA</a>	The CopyMetaFile function copies the content of a Windows-format metafile to the specified file.
<a href="#">CopyMetaFileW</a>	The CopyMetaFile function copies the content of a Windows-format metafile to the specified file.
<a href="#">CreateBitmap</a>	The CreateBitmap function creates a bitmap with the specified width, height, and color format (color planes and bits-per-pixel).
<a href="#">CreateBitmapIndirect</a>	The CreateBitmapIndirect function creates a bitmap with the specified width, height, and color format (color planes and bits-per-pixel).
<a href="#">CreateBrushIndirect</a>	The CreateBrushIndirect function creates a logical brush that has the specified style, color, and pattern.
<a href="#">CreateColorSpaceA</a>	The CreateColorSpace function creates a logical color space.
<a href="#">CreateColorSpaceW</a>	The CreateColorSpace function creates a logical color space.
<a href="#">CreateCompatibleBitmap</a>	The CreateCompatibleBitmap function creates a bitmap compatible with the device that is associated with the specified device context.
<a href="#">CreateCompatibleDC</a>	The CreateCompatibleDC function creates a memory device context (DC) compatible with the specified device.
<a href="#">CreateDCA</a>	The CreateDC function creates a device context (DC) for a device using the specified name.
<a href="#">CreateDCW</a>	The CreateDC function creates a device context (DC) for a device using the specified name.
<a href="#">CreateDIBitmap</a>	The CreateDIBitmap function creates a compatible bitmap (DDB) from a DIB and, optionally, sets the bitmap bits.
<a href="#">CreateDIBPatternBrush</a>	The CreateDIBPatternBrush function creates a logical brush that has the pattern specified by the specified device-independent bitmap (DIB).
<a href="#">CreateDIBPatternBrushPt</a>	The CreateDIBPatternBrushPt function creates a logical brush that has the pattern specified by the device-independent bitmap (DIB).
<a href="#">CreateDIBSection</a>	The CreateDIBSection function creates a DIB that applications can write to directly.
<a href="#">CreateDiscardableBitmap</a>	The CreateDiscardableBitmap function creates a discardable bitmap that is compatible with the specified device.

TITLE	DESCRIPTION
CreateEllipticRgn	The CreateEllipticRgn function creates an elliptical region.
CreateEllipticRgnIndirect	The CreateEllipticRgnIndirect function creates an elliptical region.
CreateEnhMetaFileA	The CreateEnhMetaFile function creates a device context for an enhanced-format metafile. This device context can be used to store a device-independent picture.
CreateEnhMetaFileW	The CreateEnhMetaFile function creates a device context for an enhanced-format metafile. This device context can be used to store a device-independent picture.
CreateFontA	The CreateFont function creates a logical font with the specified characteristics. The logical font can subsequently be selected as the font for any device.
CreateFontIndirectA	The CreateFontIndirect function creates a logical font that has the specified characteristics. The font can subsequently be selected as the current font for any device context.
CreateFontIndirectExA	The CreateFontIndirectEx function specifies a logical font that has the characteristics in the specified structure. The font can subsequently be selected as the current font for any device context.
CreateFontIndirectExW	The CreateFontIndirectEx function specifies a logical font that has the characteristics in the specified structure. The font can subsequently be selected as the current font for any device context.
CreateFontIndirectW	The CreateFontIndirect function creates a logical font that has the specified characteristics. The font can subsequently be selected as the current font for any device context.
CreateFontW	The CreateFont function creates a logical font with the specified characteristics. The logical font can subsequently be selected as the font for any device.
CreateHalftonePalette	The CreateHalftonePalette function creates a halftone palette for the specified device context (DC).
CreateHatchBrush	The CreateHatchBrush function creates a logical brush that has the specified hatch pattern and color.
CreateICA	The CreateIC function creates an information context for the specified device.
CreateICW	The CreateIC function creates an information context for the specified device.
CreateMetaFileA	The CreateMetaFile function creates a device context for a Windows-format metafile.

TITLE	DESCRIPTION
CreateMetaFileW	The CreateMetaFile function creates a device context for a Windows-format metafile.
CreatePalette	The CreatePalette function creates a logical palette.
CreatePatternBrush	The CreatePatternBrush function creates a logical brush with the specified bitmap pattern. The bitmap can be a DIB section bitmap, which is created by the CreateDIBSection function, or it can be a device-dependent bitmap.
CreatePen	The CreatePen function creates a logical pen that has the specified style, width, and color. The pen can subsequently be selected into a device context and used to draw lines and curves.
CreatePenIndirect	The CreatePenIndirect function creates a logical cosmetic pen that has the style, width, and color specified in a structure.
CreatePolygonRgn	The CreatePolygonRgn function creates a polygonal region.
CreatePolyPolygonRgn	The CreatePolyPolygonRgn function creates a region consisting of a series of polygons. The polygons can overlap.
CreateRectRgn	The CreateRectRgn function creates a rectangular region.
CreateRectRgnIndirect	The CreateRectRgnIndirect function creates a rectangular region.
CreateRoundRectRgn	The CreateRoundRectRgn function creates a rectangular region with rounded corners.
CreateScalableFontResourceA	The CreateScalableFontResource function creates a font resource file for a scalable font.
CreateScalableFontResourceW	The CreateScalableFontResource function creates a font resource file for a scalable font.
CreateSolidBrush	The CreateSolidBrush function creates a logical brush that has the specified solid color.
DeleteColorSpace	The DeleteColorSpace function removes and destroys a specified color space.
DeleteDC	The DeleteDC function deletes the specified device context (DC).
DeleteEnhMetaFile	The DeleteEnhMetaFile function deletes an enhanced-format metafile or an enhanced-format metafile handle.
DeleteMetaFile	The DeleteMetaFile function deletes a Windows-format metafile or Windows-format metafile handle.

TITLE	DESCRIPTION
<a href="#">DeleteObject</a>	The DeleteObject function deletes a logical pen, brush, font, bitmap, region, or palette, freeing all system resources associated with the object. After the object is deleted, the specified handle is no longer valid.
<a href="#">DescribePixelFormat</a>	The DescribePixelFormat function obtains information about the pixel format identified by iPixelFormat of the device associated with hdc. The function sets the members of the PIXELFORMATDESCRIPTOR structure pointed to by ppfd with that pixel format data.
<a href="#">DeviceCapabilitiesA</a>	The DeviceCapabilities function retrieves the capabilities of a printer driver.
<a href="#">DeviceCapabilitiesW</a>	The DeviceCapabilities function retrieves the capabilities of a printer driver.
<a href="#">DPtoLP</a>	The DPtoLP function converts device coordinates into logical coordinates. The conversion depends on the mapping mode of the device context, the settings of the origins and extents for the window and viewport, and the world transformation.
<a href="#">DrawEscape</a>	The DrawEscape function provides drawing capabilities of the specified video display that are not directly available through the graphics device interface (GDI).
<a href="#">Ellipse</a>	The Ellipse function draws an ellipse. The center of the ellipse is the center of the specified bounding rectangle. The ellipse is outlined by using the current pen and is filled by using the current brush.
<a href="#">EndDoc</a>	The EndDoc function ends a print job.
<a href="#">EndPage</a>	The EndPage function notifies the device that the application has finished writing to a page. This function is typically used to direct the device driver to advance to a new page.
<a href="#">EndPath</a>	The EndPath function closes a path bracket and selects the path defined by the bracket into the specified device context.
<a href="#">EnumEnhMetaFile</a>	The EnumEnhMetaFile function enumerates the records within an enhanced-format metafile by retrieving each record and passing it to the specified callback function.
<a href="#">EnumFontFamiliesA</a>	The EnumFontFamilies function enumerates the fonts in a specified font family that are available on a specified device.
<a href="#">EnumFontFamiliesExA</a>	The EnumFontFamiliesEx function enumerates all uniquely-named fonts in the system that match the font characteristics specified by the LOGFONT structure. EnumFontFamiliesEx enumerates fonts based on typeface name, character set, or both.



TITLE	DESCRIPTION
<a href="#">EnumFontFamiliesExW</a>	The EnumFontFamiliesEx function enumerates all uniquely-named fonts in the system that match the font characteristics specified by the LOGFONT structure. EnumFontFamiliesEx enumerates fonts based on typeface name, character set, or both.
<a href="#">EnumFontFamiliesW</a>	The EnumFontFamilies function enumerates the fonts in a specified font family that are available on a specified device.
<a href="#">EnumFontsA</a>	The EnumFonts function enumerates the fonts available on a specified device.
<a href="#">EnumFontsW</a>	The EnumFonts function enumerates the fonts available on a specified device.
<a href="#">EnumICMProfilesA</a>	The EnumICMProfiles function enumerates the different output color profiles that the system supports for a given device context.
<a href="#">EnumICMProfilesW</a>	The EnumICMProfiles function enumerates the different output color profiles that the system supports for a given device context.
<a href="#">EnumMetaFile</a>	The EnumMetaFile function enumerates the records within a Windows-format metafile by retrieving each record and passing it to the specified callback function.
<a href="#">EnumObjects</a>	The EnumObjects function enumerates the pens or brushes available for the specified device context (DC).
<a href="#">EqualRgn</a>	The EqualRgn function checks the two specified regions to determine whether they are identical. The function considers two regions identical if they are equal in size and shape.
<a href="#">Escape</a>	Enables an application to access the system-defined device capabilities that are not available through GDI.
<a href="#">ExcludeClipRect</a>	The ExcludeClipRect function creates a new clipping region that consists of the existing clipping region minus the specified rectangle.
<a href="#">ExtCreatePen</a>	The ExtCreatePen function creates a logical cosmetic or geometric pen that has the specified style, width, and brush attributes.
<a href="#">ExtCreateRegion</a>	The ExtCreateRegion function creates a region from the specified region and transformation data.
<a href="#">ExtEscape</a>	The ExtEscape function enables an application to access device capabilities that are not available through GDI.
<a href="#">ExtFloodFill</a>	The ExtFloodFill function fills an area of the display surface with the current brush.

TITLE	DESCRIPTION
<a href="#">ExtSelectClipRgn</a>	The ExtSelectClipRgn function combines the specified region with the current clipping region using the specified mode.
<a href="#">ExtTextOutA</a>	The ExtTextOut function draws text using the currently selected font, background color, and text color. You can optionally provide dimensions to be used for clipping, opaquing, or both.
<a href="#">ExtTextOutW</a>	The ExtTextOut function draws text using the currently selected font, background color, and text color. You can optionally provide dimensions to be used for clipping, opaquing, or both.
<a href="#">FillPath</a>	The FillPath function closes any open figures in the current path and fills the path's interior by using the current brush and polygon-filling mode.
<a href="#">FillRgn</a>	The FillRgn function fills a region by using the specified brush.
<a href="#">FlattenPath</a>	The FlattenPath function transforms any curves in the path that is selected into the current device context (DC), turning each curve into a sequence of lines.
<a href="#">FloodFill</a>	The FloodFill function fills an area of the display surface with the current brush. The area is assumed to be bounded as specified by the crFill parameter.
<a href="#">FrameRgn</a>	The FrameRgn function draws a border around the specified region by using the specified brush.
<a href="#">GdiAlphaBlend</a>	The GdiAlphaBlend function displays bitmaps that have transparent or semitransparent pixels.
<a href="#">GdiComment</a>	The GdiComment function copies a comment from a buffer into a specified enhanced-format metafile.
<a href="#">GdiFlush</a>	The GdiFlush function flushes the calling thread's current batch.
<a href="#">GdiGetBatchLimit</a>	The GdiGetBatchLimit function returns the maximum number of function calls that can be accumulated in the calling thread's current batch. The system flushes the current batch whenever this limit is exceeded.
<a href="#">GdiGradientFill</a>	The GdiGradientFill function fills rectangle and triangle structures.
<a href="#">GdiSetBatchLimit</a>	The GdiSetBatchLimit function sets the maximum number of function calls that can be accumulated in the calling thread's current batch. The system flushes the current batch whenever this limit is exceeded.

TITLE	DESCRIPTION
<a href="#">GdiTransparentBlt</a>	The GdiTransparentBlt function performs a bit-block transfer of the color data corresponding to a rectangle of pixels from the specified source device context into a destination device context.
<a href="#">GetArcDirection</a>	The GetArcDirection function retrieves the current arc direction for the specified device context. Arc and rectangle functions use the arc direction.
<a href="#">GetAspectRatioFilterEx</a>	The GetAspectRatioFilterEx function retrieves the setting for the current aspect-ratio filter.
<a href="#">GetBitmapBits</a>	The GetBitmapBits function copies the bitmap bits of a specified device-dependent bitmap into a buffer.
<a href="#">GetBitmapDimensionEx</a>	The GetBitmapDimensionEx function retrieves the dimensions of a compatible bitmap. The retrieved dimensions must have been set by the SetBitmapDimensionEx function.
<a href="#">GetBkColor</a>	The GetBkColor function returns the current background color for the specified device context.
<a href="#">GetBkMode</a>	The GetBkMode function returns the current background mix mode for a specified device context. The background mix mode of a device context affects text, hatched brushes, and pen styles that are not solid lines.
<a href="#">GetBoundsRect</a>	The GetBoundsRect function obtains the current accumulated bounding rectangle for a specified device context.
<a href="#">GetBrushOrgEx</a>	The GetBrushOrgEx function retrieves the current brush origin for the specified device context. This function replaces the GetBrushOrg function.
<a href="#">GetBValue</a>	The GetBValue macro retrieves an intensity value for the blue component of a red, green, blue (RGB) value.
<a href="#">GetCharABCWidthsA</a>	The GetCharABCWidths function retrieves the widths, in logical units, of consecutive characters in a specified range from the current TrueType font. This function succeeds only with TrueType fonts.
<a href="#">GetCharABCWidthsFloatA</a>	The GetCharABCWidthsFloat function retrieves the widths, in logical units, of consecutive characters in a specified range from the current font.
<a href="#">GetCharABCWidthsFloatW</a>	The GetCharABCWidthsFloat function retrieves the widths, in logical units, of consecutive characters in a specified range from the current font.

TITLE	DESCRIPTION
<a href="#">GetCharABCWidthsI</a>	The <code>GetCharABCWidthsI</code> function retrieves the widths, in logical units, of consecutive glyph indices in a specified range from the current TrueType font. This function succeeds only with TrueType fonts.
<a href="#">GetCharABCWidthsW</a>	The <code>GetCharABCWidths</code> function retrieves the widths, in logical units, of consecutive characters in a specified range from the current TrueType font. This function succeeds only with TrueType fonts.
<a href="#">GetCharacterPlacementA</a>	The <code>GetCharacterPlacement</code> function retrieves information about a character string, such as character widths, caret positioning, ordering within the string, and glyph rendering.
<a href="#">GetCharacterPlacementW</a>	The <code>GetCharacterPlacement</code> function retrieves information about a character string, such as character widths, caret positioning, ordering within the string, and glyph rendering.
<a href="#">GetCharWidth32A</a>	The <code>GetCharWidth32</code> function retrieves the widths, in logical coordinates, of consecutive characters in a specified range from the current font.
<a href="#">GetCharWidth32W</a>	The <code>GetCharWidth32</code> function retrieves the widths, in logical coordinates, of consecutive characters in a specified range from the current font.
<a href="#">GetCharWidthA</a>	The <code>GetCharWidth</code> function retrieves the widths, in logical coordinates, of consecutive characters in a specified range from the current font.
<a href="#">GetCharWidthFloatA</a>	The <code>GetCharWidthFloat</code> function retrieves the fractional widths of consecutive characters in a specified range from the current font.
<a href="#">GetCharWidthFloatW</a>	The <code>GetCharWidthFloat</code> function retrieves the fractional widths of consecutive characters in a specified range from the current font.
<a href="#">GetCharWidthI</a>	The <code>GetCharWidthI</code> function retrieves the widths, in logical coordinates, of consecutive glyph indices in a specified range from the current font.
<a href="#">GetCharWidthW</a>	The <code>GetCharWidth</code> function retrieves the widths, in logical coordinates, of consecutive characters in a specified range from the current font.
<a href="#">GetClipBox</a>	The <code>GetClipBox</code> function retrieves the dimensions of the tightest bounding rectangle that can be drawn around the current visible area on the device.
<a href="#">GetClipRgn</a>	The <code>GetClipRgn</code> function retrieves a handle identifying the current application-defined clipping region for the specified device context.

TITLE	DESCRIPTION
<a href="#">GetColorAdjustment</a>	The GetColorAdjustment function retrieves the color adjustment values for the specified device context (DC).
<a href="#">GetColorSpace</a>	The GetColorSpace function retrieves the handle to the input color space from a specified device context.
<a href="#">GetCurrentObject</a>	The GetCurrentObject function retrieves a handle to an object of the specified type that has been selected into the specified device context (DC).
<a href="#">GetCurrentPositionEx</a>	The GetCurrentPositionEx function retrieves the current position in logical coordinates.
<a href="#">GetCValue</a>	The GetCValue macro retrieves the cyan color value from a CMYK color value.
<a href="#">GetDCBrushColor</a>	The GetDCBrushColor function retrieves the current brush color for the specified device context (DC).
<a href="#">GetDCOrgEx</a>	The GetDCOrgEx function retrieves the final translation origin for a specified device context (DC).
<a href="#">GetDCPenColor</a>	The GetDCPenColor function retrieves the current pen color for the specified device context (DC).
<a href="#">GetDeviceCaps</a>	The GetDeviceCaps function retrieves device-specific information for the specified device.
<a href="#">GetDeviceGammaRamp</a>	The GetDeviceGammaRamp function gets the gamma ramp on direct color display boards having drivers that support downloadable gamma ramps in hardware.
<a href="#">GetDIBColorTable</a>	The GetDIBColorTable function retrieves RGB (red, green, blue) color values from a range of entries in the color table of the DIB section bitmap that is currently selected into a specified device context.
<a href="#">GetDIBits</a>	The GetDIBits function retrieves the bits of the specified compatible bitmap and copies them into a buffer as a DIB using the specified format.
<a href="#">GetEnhMetaFileA</a>	The GetEnhMetaFile function creates a handle that identifies the enhanced-format metafile stored in the specified file.
<a href="#">GetEnhMetaFileBits</a>	The GetEnhMetaFileBits function retrieves the contents of the specified enhanced-format metafile and copies them into a buffer.
<a href="#">GetEnhMetaFileDescriptionA</a>	The GetEnhMetaFileDescription function retrieves an optional text description from an enhanced-format metafile and copies the string to the specified buffer.
<a href="#">GetEnhMetaFileDescriptionW</a>	The GetEnhMetaFileDescription function retrieves an optional text description from an enhanced-format metafile and copies the string to the specified buffer.

TITLE	DESCRIPTION
<a href="#">GetEnhMetaFileHeader</a>	The GetEnhMetaFileHeader function retrieves the record containing the header for the specified enhanced-format metafile.
<a href="#">GetEnhMetaFilePaletteEntries</a>	The GetEnhMetaFilePaletteEntries function retrieves optional palette entries from the specified enhanced metafile.
<a href="#">GetEnhMetaFilePixelFormat</a>	The GetEnhMetaFilePixelFormat function retrieves pixel format information for an enhanced metafile.
<a href="#">GetEnhMetaFileW</a>	The GetEnhMetaFile function creates a handle that identifies the enhanced-format metafile stored in the specified file.
<a href="#">GetFontData</a>	The GetFontData function retrieves font metric data for a TrueType font.
<a href="#">GetFontLanguageInfo</a>	The GetFontLanguageInfo function returns information about the currently selected font for the specified display context. Applications typically use this information and the GetCharacterPlacement function to prepare a character string for display.
<a href="#">GetFontUnicodeRanges</a>	The GetFontUnicodeRanges function returns information about which Unicode characters are supported by a font. The information is returned as a GLYPHSET structure.
<a href="#">GetGlyphIndicesA</a>	The GetGlyphIndices function translates a string into an array of glyph indices. The function can be used to determine whether a glyph exists in a font.
<a href="#">GetGlyphIndicesW</a>	The GetGlyphIndices function translates a string into an array of glyph indices. The function can be used to determine whether a glyph exists in a font.
<a href="#">GetGlyphOutlineA</a>	The GetGlyphOutline function retrieves the outline or bitmap for a character in the TrueType font that is selected into the specified device context.
<a href="#">GetGlyphOutlineW</a>	The GetGlyphOutline function retrieves the outline or bitmap for a character in the TrueType font that is selected into the specified device context.
<a href="#">GetGraphicsMode</a>	The GetGraphicsMode function retrieves the current graphics mode for the specified device context.
<a href="#">GetGValue</a>	The GetGValue macro retrieves an intensity value for the green component of a red, green, blue (RGB) value.
<a href="#">GetICMProfileA</a>	The GetICMProfile function retrieves the file name of the current output color profile for a specified device context.

TITLE	DESCRIPTION
<a href="#">GetICMProfileW</a>	The GetICMProfile function retrieves the file name of the current output color profile for a specified device context.
<a href="#">GetKerningPairsA</a>	The GetKerningPairs function retrieves the character-kerning pairs for the currently selected font for the specified device context.
<a href="#">GetKerningPairsW</a>	The GetKerningPairs function retrieves the character-kerning pairs for the currently selected font for the specified device context.
<a href="#">GetKValue</a>	The GetKValue macro retrieves the black color value from a CMYK color value.
<a href="#">GetLayout</a>	The GetLayout function returns the layout of a device context (DC).
<a href="#">GetLogColorSpaceA</a>	The GetLogColorSpace function retrieves the color space definition identified by a specified handle.
<a href="#">GetLogColorSpaceW</a>	The GetLogColorSpace function retrieves the color space definition identified by a specified handle.
<a href="#">GetMapMode</a>	The GetMapMode function retrieves the current mapping mode.
<a href="#">GetMetaFileA</a>	The GetMetaFile function creates a handle that identifies the metafile stored in the specified file.
<a href="#">GetMetaFileBitsEx</a>	The GetMetaFileBitsEx function retrieves the contents of a Windows-format metafile and copies them into the specified buffer.
<a href="#">GetMetaFileW</a>	The GetMetaFile function creates a handle that identifies the metafile stored in the specified file.
<a href="#">GetMetaRgn</a>	The GetMetaRgn function retrieves the current metaregion for the specified device context.
<a href="#">GetMiterLimit</a>	The GetMiterLimit function retrieves the miter limit for the specified device context.
<a href="#">GetMValue</a>	The GetMValue macro retrieves the magenta color value from a CMYK color value.
<a href="#">GetNearestColor</a>	The GetNearestColor function retrieves a color value identifying a color from the system palette that will be displayed when the specified color value is used.
<a href="#">GetNearestPaletteIndex</a>	The GetNearestPaletteIndex function retrieves the index for the entry in the specified logical palette most closely matching a specified color value.

TITLE	DESCRIPTION
<a href="#">GetObject</a>	The GetObject function retrieves information for the specified graphics object.
<a href="#">GetObjectA</a>	The GetObject function retrieves information for the specified graphics object.
<a href="#">GetObjectType</a>	The GetObjectType retrieves the type of the specified object.
<a href="#">GetObjectW</a>	The GetObject function retrieves information for the specified graphics object.
<a href="#">GetOutlineTextMetricsA</a>	The GetOutlineTextMetrics function retrieves text metrics for TrueType fonts.
<a href="#">GetOutlineTextMetricsW</a>	The GetOutlineTextMetrics function retrieves text metrics for TrueType fonts.
<a href="#">GetPaletteEntries</a>	The GetPaletteEntries function retrieves a specified range of palette entries from the given logical palette.
<a href="#">GetPath</a>	The GetPath function retrieves the coordinates defining the endpoints of lines and the control points of curves found in the path that is selected into the specified device context.
<a href="#">GetPixel</a>	The GetPixel function retrieves the red, green, blue (RGB) color value of the pixel at the specified coordinates.
<a href="#">GetPixelFormat</a>	The GetPixelFormat function obtains the index of the currently selected pixel format of the specified device context.
<a href="#">GetPolyFillMode</a>	The GetPolyFillMode function retrieves the current polygon fill mode.
<a href="#">GetRandomRgn</a>	The GetRandomRgn function copies the system clipping region of a specified device context to a specific region.
<a href="#">GetRasterizerCaps</a>	The GetRasterizerCaps function returns flags indicating whether TrueType fonts are installed in the system.
<a href="#">GetRegionData</a>	The GetRegionData function fills the specified buffer with data describing a region. This data includes the dimensions of the rectangles that make up the region.
<a href="#">GetRgnBox</a>	The GetRgnBox function retrieves the bounding rectangle of the specified region.
<a href="#">GetROP2</a>	The GetROP2 function retrieves the foreground mix mode of the specified device context. The mix mode specifies how the pen or interior color and the color already on the screen are combined to yield a new color.
<a href="#">GetRValue</a>	The GetRValue macro retrieves an intensity value for the red component of a red, green, blue (RGB) value.



TITLE	DESCRIPTION
<a href="#">GetStockObject</a>	The GetStockObject function retrieves a handle to one of the stock pens, brushes, fonts, or palettes.
<a href="#">GetStretchBltMode</a>	The GetStretchBltMode function retrieves the current stretching mode. The stretching mode defines how color data is added to or removed from bitmaps that are stretched or compressed when the StretchBlt function is called.
<a href="#">GetSystemPaletteEntries</a>	The GetSystemPaletteEntries function retrieves a range of palette entries from the system palette that is associated with the specified device context (DC).
<a href="#">GetSystemPaletteUse</a>	The GetSystemPaletteUse function retrieves the current state of the system (physical) palette for the specified device context (DC).
<a href="#">GetTextAlign</a>	The GetTextAlign function retrieves the text-alignment setting for the specified device context.
<a href="#">GetTextCharacterExtra</a>	The GetTextCharacterExtra function retrieves the current intercharacter spacing for the specified device context.
<a href="#">GetTextCharset</a>	Retrieves a character set identifier for the font that is currently selected into a specified device context.
<a href="#">GetTextCharsetInfo</a>	Retrieves information about the character set of the font that is currently selected into a specified device context.
<a href="#">GetTextColor</a>	The GetTextColor function retrieves the current text color for the specified device context.
<a href="#">GetTextExtentExPointA</a>	The GetTextExtentExPoint function retrieves the number of characters in a specified string that will fit within a specified space and fills an array with the text extent for each of those characters.
<a href="#">GetTextExtentExPointI</a>	The GetTextExtentExPointI function retrieves the number of characters in a specified string that will fit within a specified space and fills an array with the text extent for each of those characters.
<a href="#">GetTextExtentExPointW</a>	The GetTextExtentExPoint function retrieves the number of characters in a specified string that will fit within a specified space and fills an array with the text extent for each of those characters.
<a href="#">GetTextExtentPoint32A</a>	The GetTextExtentPoint32 function computes the width and height of the specified string of text.
<a href="#">GetTextExtentPoint32W</a>	The GetTextExtentPoint32 function computes the width and height of the specified string of text.
<a href="#">GetTextExtentPointA</a>	The GetTextExtentPoint function computes the width and height of the specified string of text.

TITLE	DESCRIPTION
<a href="#">GetTextExtentPointI</a>	The GetTextExtentPointI function computes the width and height of the specified array of glyph indices.
<a href="#">GetTextExtentPointW</a>	The GetTextExtentPoint function computes the width and height of the specified string of text.
<a href="#">GetTextFaceA</a>	The GetTextFace function retrieves the typeface name of the font that is selected into the specified device context.
<a href="#">GetTextFaceW</a>	The GetTextFace function retrieves the typeface name of the font that is selected into the specified device context.
<a href="#">GetTextMetrics</a>	The GetTextMetrics function fills the specified buffer with the metrics for the currently selected font.
<a href="#">GetTextMetricsA</a>	The GetTextMetrics function fills the specified buffer with the metrics for the currently selected font.
<a href="#">GetTextMetricsW</a>	The GetTextMetrics function fills the specified buffer with the metrics for the currently selected font.
<a href="#">GetViewportExtEx</a>	The GetViewportExtEx function retrieves the x-extent and y-extent of the current viewport for the specified device context.
<a href="#">GetViewportOrgEx</a>	The GetViewportOrgEx function retrieves the x-coordinates and y-coordinates of the viewport origin for the specified device context.
<a href="#">GetWindowExtEx</a>	This function retrieves the x-extent and y-extent of the window for the specified device context.
<a href="#">GetWindowOrgEx</a>	The GetWindowOrgEx function retrieves the x-coordinates and y-coordinates of the window origin for the specified device context.
<a href="#">GetWinMetaFileBits</a>	The GetWinMetaFileBits function converts the enhanced-format records from a metafile into Windows-format records and stores the converted records in the specified buffer.
<a href="#">GetWorldTransform</a>	The GetWorldTransform function retrieves the current world-space to page-space transformation.
<a href="#">GetYValue</a>	The GetYValue macro retrieves the yellow color value from a CMYK color value.
<a href="#">GradientFill</a>	The GradientFill function fills rectangle and triangle structures.
<a href="#">IntersectClipRect</a>	The IntersectClipRect function creates a new clipping region from the intersection of the current clipping region and the specified rectangle.

TITLE	DESCRIPTION
<a href="#">InvertRgn</a>	The InvertRgn function inverts the colors in the specified region.
<a href="#">LineDDA</a>	The LineDDA function determines which pixels should be highlighted for a line defined by the specified starting and ending points.
<a href="#">LineTo</a>	The LineTo function draws a line from the current position up to, but not including, the specified point.
<a href="#">LPtoDP</a>	The LPtoDP function converts logical coordinates into device coordinates. The conversion depends on the mapping mode of the device context, the settings of the origins and extents for the window and viewport, and the world transformation.
<a href="#">MAKEPOINTS</a>	The MAKEPOINTS macro converts a value that contains the x- and y-coordinates of a point into a POINTS structure.
<a href="#">MAKEROP4</a>	The MAKEROP4 macro creates a quaternary raster operation code for use with the MaskBlt function.
<a href="#">MaskBlt</a>	The MaskBlt function combines the color data for the source and destination bitmaps using the specified mask and raster operation.
<a href="#">ModifyWorldTransform</a>	The ModifyWorldTransform function changes the world transformation for a device context using the specified mode.
<a href="#">MoveToEx</a>	The MoveToEx function updates the current position to the specified point and optionally returns the previous position.
<a href="#">OffsetClipRgn</a>	The OffsetClipRgn function moves the clipping region of a device context by the specified offsets.
<a href="#">OffsetRgn</a>	The OffsetRgn function moves a region by the specified offsets.
<a href="#">OffsetViewportOrgEx</a>	The OffsetViewportOrgEx function modifies the viewport origin for a device context using the specified horizontal and vertical offsets.
<a href="#">OffsetWindowOrgEx</a>	The OffsetWindowOrgEx function modifies the window origin for a device context using the specified horizontal and vertical offsets.
<a href="#">PaintRgn</a>	The PaintRgn function paints the specified region by using the brush currently selected into the device context.
<a href="#">PALETTEINDEX</a>	The PALETTEINDEX macro accepts an index to a logical-color palette entry and returns a palette-entry specifier consisting of a COLORREF value that specifies the color associated with the given index.

TITLE	DESCRIPTION
<a href="#">PALETTE_RGB</a>	The PALETTE_RGB macro accepts three values that represent the relative intensities of red, green, and blue and returns a palette-relative red, green, blue (RGB) specifier consisting of 2 in the high-order byte and an RGB value in the three low-order bytes. An application using a color palette can pass this specifier, instead of an explicit RGB value, to functions that expect a color.
<a href="#">PatBlt</a>	The PatBlt function paints the specified rectangle using the brush that is currently selected into the specified device context. The brush color and the surface color or colors are combined by using the specified raster operation.
<a href="#">PathToRegion</a>	The PathToRegion function creates a region from the path that is selected into the specified device context. The resulting region uses device coordinates.
<a href="#">Pie</a>	The Pie function draws a pie-shaped wedge bounded by the intersection of an ellipse and two radials. The pie is outlined by using the current pen and filled by using the current brush.
<a href="#">PlayEnhMetaFile</a>	The PlayEnhMetaFile function displays the picture stored in the specified enhanced-format metafile.
<a href="#">PlayEnhMetaFileRecord</a>	The PlayEnhMetaFileRecord function plays an enhanced-metafile record by executing the graphics device interface (GDI) functions identified by the record.
<a href="#">PlayMetaFile</a>	The PlayMetaFile function displays the picture stored in the given Windows-format metafile on the specified device.
<a href="#">PlayMetaFileRecord</a>	The PlayMetaFileRecord function plays a Windows-format metafile record by executing the graphics device interface (GDI) function contained within that record.
<a href="#">PlgBlt</a>	The PlgBlt function performs a bit-block transfer of the bits of color data from the specified rectangle in the source device context to the specified parallelogram in the destination device context.
<a href="#">PolyBezier</a>	The PolyBezier function draws one or more Bézier curves.
<a href="#">PolyBezierTo</a>	The PolyBezierTo function draws one or more Bézier curves.
<a href="#">PolyDraw</a>	The PolyDraw function draws a set of line segments and Bézier curves.
<a href="#">Polygon</a>	The Polygon function draws a polygon consisting of two or more vertices connected by straight lines. The polygon is outlined by using the current pen and filled by using the current brush and polygon fill mode.
<a href="#">Polyline</a>	The Polyline function draws a series of line segments by connecting the points in the specified array.

TITLE	DESCRIPTION
<a href="#">PolylineTo</a>	The PolylineTo function draws one or more straight lines.
<a href="#">PolyPolygon</a>	The PolyPolygon function draws a series of closed polygons. Each polygon is outlined by using the current pen and filled by using the current brush and polygon fill mode. The polygons drawn by this function can overlap.
<a href="#">PolyPolyline</a>	The PolyPolyline function draws multiple series of connected line segments.
<a href="#">PolyTextOutA</a>	The PolyTextOut function draws several strings using the font and text colors currently selected in the specified device context.
<a href="#">PolyTextOutW</a>	The PolyTextOut function draws several strings using the font and text colors currently selected in the specified device context.
<a href="#">PtInRegion</a>	The PtInRegion function determines whether the specified point is inside the specified region.
<a href="#">PtVisible</a>	The PtVisible function determines whether the specified point is within the clipping region of a device context.
<a href="#">RealizePalette</a>	The RealizePalette function maps palette entries from the current logical palette to the system palette.
<a href="#">Rectangle</a>	The Rectangle function draws a rectangle. The rectangle is outlined by using the current pen and filled by using the current brush.
<a href="#">RectInRegion</a>	The RectInRegion function determines whether any part of the specified rectangle is within the boundaries of a region.
<a href="#">RectVisible</a>	The RectVisible function determines whether any part of the specified rectangle lies within the clipping region of a device context.
<a href="#">RemoveFontMemResourceEx</a>	The RemoveFontMemResourceEx function removes the fonts added from a memory image file.
<a href="#">RemoveFontResourceA</a>	The RemoveFontResource function removes the fonts in the specified file from the system font table.
<a href="#">RemoveFontResourceExA</a>	The RemoveFontResourceEx function removes the fonts in the specified file from the system font table.
<a href="#">RemoveFontResourceExW</a>	The RemoveFontResourceEx function removes the fonts in the specified file from the system font table.
<a href="#">RemoveFontResourceW</a>	The RemoveFontResource function removes the fonts in the specified file from the system font table.

TITLE	DESCRIPTION
<a href="#">ResetDCA</a>	The ResetDC function updates the specified printer or plotter device context (DC) using the specified information.
<a href="#">ResetDCW</a>	The ResetDC function updates the specified printer or plotter device context (DC) using the specified information.
<a href="#">ResizePalette</a>	The ResizePalette function increases or decreases the size of a logical palette based on the specified value.
<a href="#">RestoreDC</a>	The RestoreDC function restores a device context (DC) to the specified state. The DC is restored by popping state information off a stack created by earlier calls to the SaveDC function.
<a href="#">RGB</a>	The RGB macro selects a red, green, blue (RGB) color based on the arguments supplied and the color capabilities of the output device.
<a href="#">RoundRect</a>	The RoundRect function draws a rectangle with rounded corners. The rectangle is outlined by using the current pen and filled by using the current brush.
<a href="#">SaveDC</a>	The SaveDC function saves the current state of the specified device context (DC) by copying data describing selected objects and graphic modes (such as the bitmap, brush, palette, font, pen, region, drawing mode, and mapping mode) to a context stack.
<a href="#">ScaleViewportExtEx</a>	The ScaleViewportExtEx function modifies the viewport for a device context using the ratios formed by the specified multiplicands and divisors.
<a href="#">ScaleWindowExtEx</a>	The ScaleWindowExtEx function modifies the window for a device context using the ratios formed by the specified multiplicands and divisors.
<a href="#">SelectClipPath</a>	The SelectClipPath function selects the current path as a clipping region for a device context, combining the new region with any existing clipping region using the specified mode.
<a href="#">SelectClipRgn</a>	The SelectClipRgn function selects a region as the current clipping region for the specified device context.
<a href="#">SelectObject</a>	The SelectObject function selects an object into the specified device context (DC). The new object replaces the previous object of the same type.
<a href="#">SelectPalette</a>	The SelectPalette function selects the specified logical palette into a device context.
<a href="#">SetAbortProc</a>	The SetAbortProc function sets the application-defined abort function that allows a print job to be canceled during spooling.

TITLE	DESCRIPTION
<a href="#">SetArcDirection</a>	The SetArcDirection sets the drawing direction to be used for arc and rectangle functions.
<a href="#">SetBitmapBits</a>	The SetBitmapBits function sets the bits of color data for a bitmap to the specified values.
<a href="#">SetBitmapDimensionEx</a>	The SetBitmapDimensionEx function assigns preferred dimensions to a bitmap. These dimensions can be used by applications; however, they are not used by the system.
<a href="#">SetBkColor</a>	The SetBkColor function sets the current background color to the specified color value, or to the nearest physical color if the device cannot represent the specified color value.
<a href="#">SetBkMode</a>	The SetBkMode function sets the background mix mode of the specified device context. The background mix mode is used with text, hatched brushes, and pen styles that are not solid lines.
<a href="#">SetBoundsRect</a>	The SetBoundsRect function controls the accumulation of bounding rectangle information for the specified device context.
<a href="#">SetBrushOrgEx</a>	The SetBrushOrgEx function sets the brush origin that GDI assigns to the next brush an application selects into the specified device context.
<a href="#">SetColorAdjustment</a>	The SetColorAdjustment function sets the color adjustment values for a device context (DC) using the specified values.
<a href="#">SetColorSpace</a>	The SetColorSpace function defines the input color space for a given device context.
<a href="#">SetDCBrushColor</a>	SetDCBrushColor function sets the current device context (DC) brush color to the specified color value. If the device cannot represent the specified color value, the color is set to the nearest physical color.
<a href="#">SetDCPenColor</a>	SetDCPenColor function sets the current device context (DC) pen color to the specified color value. If the device cannot represent the specified color value, the color is set to the nearest physical color.
<a href="#">SetDeviceGammaRamp</a>	The SetDeviceGammaRamp function sets the gamma ramp on direct color display boards having drivers that support downloadable gamma ramps in hardware.
<a href="#">SetDIBColorTable</a>	The SetDIBColorTable function sets RGB (red, green, blue) color values in a range of entries in the color table of the DIB that is currently selected into a specified device context.
<a href="#">SetDIBits</a>	The SetDIBits function sets the pixels in a compatible bitmap (DDB) using the color data found in the specified DIB.

TITLE	DESCRIPTION
<a href="#">SetDIBitsToDevice</a>	The SetDIBitsToDevice function sets the pixels in the specified rectangle on the device that is associated with the destination device context using color data from a DIB, JPEG, or PNG image.
<a href="#">SetEnhMetaFileBits</a>	The SetEnhMetaFileBits function creates a memory-based enhanced-format metafile from the specified data.
<a href="#">SetGraphicsMode</a>	The SetGraphicsMode function sets the graphics mode for the specified device context.
<a href="#">SetICMMode</a>	The SetICMMode function causes Image Color Management to be enabled, disabled, or queried on a given device context (DC).
<a href="#">SetICMProfileA</a>	The SetICMProfile function sets a specified color profile as the output profile for a specified device context (DC).
<a href="#">SetICMProfileW</a>	The SetICMProfile function sets a specified color profile as the output profile for a specified device context (DC).
<a href="#">SetLayout</a>	The SetLayout function changes the layout of a device context (DC).
<a href="#">SetMapMode</a>	The SetMapMode function sets the mapping mode of the specified device context. The mapping mode defines the unit of measure used to transform page-space units into device-space units, and also defines the orientation of the device's x and y axes.
<a href="#">SetMapperFlags</a>	The SetMapperFlags function alters the algorithm the font mapper uses when it maps logical fonts to physical fonts.
<a href="#">SetMetaFileBitsEx</a>	The SetMetaFileBitsEx function creates a memory-based Windows-format metafile from the supplied data.
<a href="#">SetMetaRgn</a>	The SetMetaRgn function intersects the current clipping region for the specified device context with the current metaregion and saves the combined region as the new metaregion for the specified device context.
<a href="#">SetMiterLimit</a>	The SetMiterLimit function sets the limit for the length of miter joins for the specified device context.
<a href="#">SetPaletteEntries</a>	The SetPaletteEntries function sets RGB (red, green, blue) color values and flags in a range of entries in a logical palette.
<a href="#">SetPixel</a>	The SetPixel function sets the pixel at the specified coordinates to the specified color.
<a href="#">SetPixelFormat</a>	The SetPixelFormat function sets the pixel format of the specified device context to the format specified by the iPixelFormat index.



TITLE	DESCRIPTION
<a href="#">SetPixelV</a>	The SetPixelV function sets the pixel at the specified coordinates to the closest approximation of the specified color. The point must be in the clipping region and the visible part of the device surface.
<a href="#">SetPolyFillMode</a>	The SetPolyFillMode function sets the polygon fill mode for functions that fill polygons.
<a href="#">SetRectRgn</a>	The SetRectRgn function converts a region into a rectangular region with the specified coordinates.
<a href="#">SetROP2</a>	The SetROP2 function sets the current foreground mix mode.
<a href="#">SetStretchBltMode</a>	The SetStretchBltMode function sets the bitmap stretching mode in the specified device context.
<a href="#">SetSystemPaletteUse</a>	The SetSystemPaletteUse function allows an application to specify whether the system palette contains 2 or 20 static colors.
<a href="#">SetTextAlign</a>	The SetTextAlign function sets the text-alignment flags for the specified device context.
<a href="#">SetTextCharacterExtra</a>	The SetTextCharacterExtra function sets the intercharacter spacing. Intercharacter spacing is added to each character, including break characters, when the system writes a line of text.
<a href="#">SetTextColor</a>	The SetTextColor function sets the text color for the specified device context to the specified color.
<a href="#">SetTextJustification</a>	The SetTextJustification function specifies the amount of space the system should add to the break characters in a string of text. The space is added when an application calls the TextOut or ExtTextOut functions.
<a href="#">SetViewportExtEx</a>	Sets the horizontal and vertical extents of the viewport for a device context by using the specified values.
<a href="#">SetViewportOrgEx</a>	The SetViewportOrgEx function specifies which device point maps to the window origin (0,0).
<a href="#">SetWindowExtEx</a>	The SetWindowExtEx function sets the horizontal and vertical extents of the window for a device context by using the specified values.
<a href="#">SetWindowOrgEx</a>	The SetWindowOrgEx function specifies which window point maps to the viewport origin (0,0).
<a href="#">SetWinMetaFileBits</a>	The SetWinMetaFileBits function converts a metafile from the older Windows format to the new enhanced format and stores the new metafile in memory.

TITLE	DESCRIPTION
<a href="#">SetWorldTransform</a>	The SetWorldTransform function sets a two-dimensional linear transformation between world space and page space for the specified device context. This transformation can be used to scale, rotate, shear, or translate graphics output.
<a href="#">StartDocA</a>	The StartDoc function starts a print job.
<a href="#">StartDocW</a>	The StartDoc function starts a print job.
<a href="#">StartPage</a>	The StartPage function prepares the printer driver to accept data.
<a href="#">StretchBlt</a>	The StretchBlt function copies a bitmap from a source rectangle into a destination rectangle, stretching or compressing the bitmap to fit the dimensions of the destination rectangle, if necessary.
<a href="#">StretchDIBits</a>	The StretchDIBits function copies the color data for a rectangle of pixels in a DIB, JPEG, or PNG image to the specified destination rectangle.
<a href="#">StrokeAndFillPath</a>	The StrokeAndFillPath function closes any open figures in a path, strokes the outline of the path by using the current pen, and fills its interior by using the current brush.
<a href="#">StrokePath</a>	The StrokePath function renders the specified path by using the current pen.
<a href="#">SwapBuffers</a>	The SwapBuffers function exchanges the front and back buffers if the current pixel format for the window referenced by the specified device context includes a back buffer.
<a href="#">TextOutA</a>	The TextOut function writes a character string at the specified location, using the currently selected font, background color, and text color.
<a href="#">TextOutW</a>	The TextOut function writes a character string at the specified location, using the currently selected font, background color, and text color.
<a href="#">TranslateCharsetInfo</a>	Translates character set information and sets all members of a destination structure to appropriate values.
<a href="#">TransparentBlt</a>	The TransparentBlt function performs a bit-block transfer of the color data corresponding to a rectangle of pixels from the specified source device context into a destination device context.
<a href="#">UnrealizeObject</a>	The UnrealizeObject function resets the origin of a brush or resets a logical palette.
<a href="#">UpdateColors</a>	The UpdateColors function updates the client area of the specified device context by remapping the current colors in the client area to the currently realized logical palette.

TITLE	DESCRIPTION
<a href="#">UpdateCMRegKeyA</a>	The UpdateCMRegKey function manages color profiles and Color Management Modules in the system.
<a href="#">UpdateCMRegKeyW</a>	The UpdateCMRegKey function manages color profiles and Color Management Modules in the system.
<a href="#">wglCopyContext</a>	The wglCopyContext function copies selected groups of rendering states from one OpenGL rendering context to another.
<a href="#">wglCreateContext</a>	The wglCreateContext function creates a new OpenGL rendering context, which is suitable for drawing on the device referenced by hdc. The rendering context has the same pixel format as the device context.
<a href="#">wglCreateLayerContext</a>	The wglCreateLayerContext function creates a new OpenGL rendering context for drawing to a specified layer plane on a device context.
<a href="#">wglDeleteContext</a>	The wglDeleteContext function deletes a specified OpenGL rendering context.
<a href="#">wglDescribeLayerPlane</a>	The wglDescribeLayerPlane function obtains information about the layer planes of a given pixel format.
<a href="#">wglGetCurrentContext</a>	The wglGetCurrentContext function obtains a handle to the current OpenGL rendering context of the calling thread.
<a href="#">wglGetCurrentDC</a>	The wglGetCurrentDC function obtains a handle to the device context that is associated with the current OpenGL rendering context of the calling thread.
<a href="#">wglGetLayerPaletteEntries</a>	Retrieves the palette entries from a given color-index layer plane for a specified device context.
<a href="#">wglGetProcAddress</a>	The wglGetProcAddress function returns the address of an OpenGL extension function for use with the current OpenGL rendering context.
<a href="#">wglMakeCurrent</a>	The wglMakeCurrent function makes a specified OpenGL rendering context the calling thread's current rendering context.
<a href="#">wglRealizeLayerPalette</a>	The wglRealizeLayerPalette function maps palette entries from a given color-index layer plane into the physical palette or initializes the palette of an RGBA layer plane.
<a href="#">wglSetLayerPaletteEntries</a>	Sets the palette entries in a given color-index layer plane for a specified device context.
<a href="#">wglShareLists</a>	The wglShareLists function enables multiple OpenGL rendering contexts to share a single display-list space.

TITLE	DESCRIPTION
<a href="#">wglSwapLayerBuffers</a>	The wglSwapLayerBuffers function swaps the front and back buffers in the overlay, underlay, and main planes of the window referenced by a specified device context.
<a href="#">wglUseFontBitmapsA</a>	The wglUseFontBitmaps function creates a set of bitmap display lists for use in the current OpenGL rendering context.
<a href="#">wglUseFontBitmapsW</a>	The wglUseFontBitmaps function creates a set of bitmap display lists for use in the current OpenGL rendering context.
<a href="#">wglUseFontOutlinesA</a>	The wglUseFontOutlines function creates a set of display lists, one for each glyph of the currently selected outline font of a device context, for use with the current rendering context.
<a href="#">wglUseFontOutlinesW</a>	The wglUseFontOutlines function creates a set of display lists, one for each glyph of the currently selected outline font of a device context, for use with the current rendering context.
<a href="#">WidenPath</a>	The WidenPath function redefines the current path as the area that would be painted if the path were stroked using the pen currently selected into the given device context.

## Callback functions

TITLE	DESCRIPTION
<a href="#">ABORTPROC</a>	The AbortProc function is an application-defined callback function used with the SetAbortProc function.
<a href="#">ENHMFENUMPROC</a>	The EnhMetaFileProc function is an application-defined callback function used with the EnumEnhMetaFile function.
<a href="#">GOBJENUMPROC</a>	The EnumObjectsProc function is an application-defined callback function used with the EnumObjects function.
<a href="#">ICMENUMPROCA</a>	The EnumICMProfilesProcCallback callback is an application-defined callback function that processes color profile data from EnumICMProfiles .
<a href="#">ICMENUMPROCW</a>	The EnumICMProfilesProcCallback callback is an application-defined callback function that processes color profile data from EnumICMProfiles .
<a href="#">LINEDDAPROC</a>	The LineDDAProc function is an application-defined callback function used with the LineDDA function.
<a href="#">MFENUMPROC</a>	The EnumMetaFileProc function is an application-defined callback function that processes Windows-format metafile records.

## Structures



TITLE	DESCRIPTION
<a href="#">ABC</a>	The ABC structure contains the width of a character in a TrueType font.
<a href="#">ABCFLOAT</a>	The ABCFLOAT structure contains the A, B, and C widths of a font character.
<a href="#">AXESLISTA</a>	The AXESLIST structure contains information on all the axes of a multiple master font.
<a href="#">AXESLISTW</a>	The AXESLIST structure contains information on all the axes of a multiple master font.
<a href="#">AXISINFOA</a>	The AXISINFO structure contains information about an axis of a multiple master font.
<a href="#">AXISINFOW</a>	The AXISINFO structure contains information about an axis of a multiple master font.
<a href="#">BITMAP</a>	The BITMAP structure defines the type, width, height, color format, and bit values of a bitmap.
<a href="#">BITMAPCOREHEADER</a>	The BITMAPCOREHEADER structure contains information about the dimensions and color format of a DIB.
<a href="#">BITMAPCOREINFO</a>	The BITMAPCOREINFO structure defines the dimensions and color information for a DIB.
<a href="#">BITMAPFILEHEADER</a>	The BITMAPFILEHEADER structure contains information about the type, size, and layout of a file that contains a DIB.
<a href="#">BITMAPINFO</a>	The BITMAPINFO structure defines the dimensions and color information for a DIB.
<a href="#">BITMAPINFOHEADER</a>	The BITMAPINFOHEADER structure contains information about the dimensions and color format of a device-independent bitmap (DIB).
<a href="#">BITMAPV4HEADER</a>	The BITMAPV4HEADER structure is the bitmap information header file. It is an extended version of the BITMAPINFOHEADER structure. Applications can use the BITMAPV5HEADER structure for added functionality.
<a href="#">BITMAPV5HEADER</a>	The BITMAPV5HEADER structure is the bitmap information header file. It is an extended version of the BITMAPINFOHEADER structure.
<a href="#">BLENDFUNCTION</a>	The BLENDFUNCTION structure controls blending by specifying the blending functions for source and destination bitmaps.
<a href="#">CHARSETINFO</a>	Contains information about a character set.
<a href="#">CIEXYZ</a>	The CIEXYZ structure contains the x,y, and z coordinates of a specific color in a specified color space.

TITLE	DESCRIPTION
CIEXYZTRIPLE	The CIEXYZTRIPLE structure contains the x,y, and z coordinates of the three colors that correspond to the red, green, and blue endpoints for a specified logical color space.
COLORADJUSTMENT	The COLORADJUSTMENT structure defines the color adjustment values used by the StretchBlt and StretchDIBits functions when the stretch mode is HALFTONE. You can set the color adjustment values by calling the SetColorAdjustment function.
DESIGNVECTOR	The DESIGNVECTOR structure is used by an application to specify values for the axes of a multiple master font.
DEVMODEA	The DEVMODE data structure contains information about the initialization and environment of a printer or a display device.
DEVMODEW	The DEVMODEW structure is used for specifying characteristics of display and print devices in the Unicode (wide) character set.
DIBSECTION	The DIBSECTION structure contains information about a DIB created by calling the CreateDIBSection function.
DISPLAY_DEVICEA	The DISPLAY_DEVICE structure receives information about the display device specified by the iDevNum parameter of the EnumDisplayDevices function.
DISPLAY_DEVICEW	The DISPLAY_DEVICE structure receives information about the display device specified by the iDevNum parameter of the EnumDisplayDevices function.
DISPLAYCONFIG_2DREGION	The DISPLAYCONFIG_2DREGION structure represents a point or an offset in a two-dimensional space.
DISPLAYCONFIG_ADAPTER_NAME	The DISPLAYCONFIG_ADAPTER_NAME structure contains information about the display adapter.
DISPLAYCONFIG_DESKTOP_IMAGE_INFO	The DISPLAYCONFIG_DESKTOP_IMAGE_INFO structure contains information about the image displayed on the desktop.
DISPLAYCONFIG_DEVICE_INFO_HEADER	The DISPLAYCONFIG_DEVICE_INFO_HEADER structure contains display information about the device.
DISPLAYCONFIG_MODE_INFO	The DISPLAYCONFIG_MODE_INFO structure contains either source mode or target mode information.
DISPLAYCONFIG_PATH_INFO	The DISPLAYCONFIG_PATH_INFO structure is used to describe a single path from a target to a source.
DISPLAYCONFIG_PATH_SOURCE_INFO	The DISPLAYCONFIG_PATH_SOURCE_INFO structure contains source information for a single path.

TITLE	DESCRIPTION
DISPLAYCONFIG_PATH_TARGET_INFO	The DISPLAYCONFIG_PATH_TARGET_INFO structure contains target information for a single path.
DISPLAYCONFIG_RATIONAL	The DISPLAYCONFIG_RATIONAL structure describes a fractional value that represents vertical and horizontal frequencies of a video mode (that is, vertical sync and horizontal sync).
DISPLAYCONFIG_SET_TARGET_PERSISTENCE	The DISPLAYCONFIG_SET_TARGET_PERSISTENCE structure contains information about setting the display.
DISPLAYCONFIG_SOURCE_DEVICE_NAME	The DISPLAYCONFIG_SOURCE_DEVICE_NAME structure contains the GDI device name for the source or view.
DISPLAYCONFIG_SOURCE_MODE	The DISPLAYCONFIG_SOURCE_MODE structure represents a point or an offset in a two-dimensional space.
DISPLAYCONFIG_SUPPORT_VIRTUAL_RESOLUTION	The DISPLAYCONFIG_SUPPORT_VIRTUAL_RESOLUTION structure contains information on the state of virtual resolution support for the monitor.
DISPLAYCONFIG_TARGET_BASE_TYPE	Specifies base output technology info for a given target ID.
DISPLAYCONFIG_TARGET_DEVICE_NAME	The DISPLAYCONFIG_TARGET_DEVICE_NAME structure contains information about the target.
DISPLAYCONFIG_TARGET_DEVICE_NAME_FLAGS	The DISPLAYCONFIG_TARGET_DEVICE_NAME_FLAGS structure contains information about a target device.
DISPLAYCONFIG_TARGET_MODE	The DISPLAYCONFIG_TARGET_MODE structure describes a display path target mode.
DISPLAYCONFIG_TARGET_PREFERRED_MODE	The DISPLAYCONFIG_TARGET_PREFERRED_MODE structure contains information about the preferred mode of a display.
DISPLAYCONFIG_VIDEO_SIGNAL_INFO	The DISPLAYCONFIG_VIDEO_SIGNAL_INFO structure contains information about the video signal for a display.
DOCINFOA	The DOCINFO structure contains the input and output file names and other information used by the StartDoc function.
DOCINFOW	The DOCINFO structure contains the input and output file names and other information used by the StartDoc function.
DRAWPATRECT	The DRAWPATRECT structure defines a rectangle to be created.
EMR	The EMR structure provides the base structure for all enhanced metafile records. An enhanced metafile record contains the parameters for a specific GDI function used to create part of a picture in an enhanced format metafile.

TITLE	DESCRIPTION
EMRABORTPATH	Contains data for the AbortPath, BeginPath, EndPath, CloseFigure, FlattenPath, WidenPath, SetMetaRgn, SaveDC, and RealizePalette enhanced metafile records.
EMRALPHABLEND	The EMRALPHABLEND structure contains members for the AlphaBlend enhanced metafile record.
EMRANGLEARC	The EMRANGLEARC structure contains members for the AngleArc enhanced metafile record.
EMRARC	The EMRARC, EMRARCTO, EMRCHORD, and EMRPIE structures contain members for the Arc, ArcTo, Chord, and Pie enhanced metafile records.
EMRBITBLT	The EMRBITBLT structure contains members for the BitBlt enhanced metafile record. Note that graphics device interface (GDI) converts the device-dependent bitmap into a device-independent bitmap (DIB) before storing it in the metafile record.
EMRCOLORCORRECTPALETTE	The EMRCOLORCORRECTPALETTE structure contains members for the ColorCorrectPalette enhanced metafile record.
EMRCOLOMATCHTOTARGET	The EMRCOLOMATCHTOTARGET structure contains members for the ColorMatchToTarget enhanced metafile record.
EMRCREATEBRUSHINDIRECT	The EMRCREATEBRUSHINDIRECT structure contains members for the CreateBrushIndirect enhanced metafile record.
EMRCREATECOLORSPACE	The EMRCREATECOLORSPACE structure contains members for the CreateColorSpace enhanced metafile record.
EMRCREATECOLORSPACEW	The EMRCREATECOLORSPACEW structure contains members for the CreateColorSpace enhanced metafile record. It differs from EMRCREATECOLORSPACE in that it has a Unicode logical color space and also has an optional array containing raw source profile data.
EMRCREATEDIBPATTERNBRUSHPT	The EMRCREATEDIBPATTERNBRUSHPT structure contains members for the CreateDIBPatternBrushPt enhanced metafile record. The BITMAPINFO structure is followed by the bitmap bits that form a packed device-independent bitmap (DIB).
EMRCREATEMONOBRUSH	The EMRCREATEMONOBRUSH structure contains members for the CreatePatternBrush (when passed a monochrome bitmap) or CreateDIBPatternBrush (when passed a monochrome DIB) enhanced metafile records.
EMRCREATEPALETTE	The EMRCREATEPALETTE structure contains members for the CreatePalette enhanced metafile record.



TITLE	DESCRIPTION
EMRCREATEPEN	The EMRCREATEPEN structure contains members for the CreatePen enhanced metafile record.
EMRELLIPSE	The EMRELLIPSE and EMRECTANGLE structures contain members for the Ellipse and Rectangle enhanced metafile records.
EMREOF	The EMREOF structure contains data for the enhanced metafile record that indicates the end of the metafile.
EMREXCLUDECLIPRECT	The EMREXCLUDECLIPRECT and EMRINTERSECTCLIPRECT structures contain members for the ExcludeClipRect and IntersectClipRect enhanced metafile records.
EMREXTCREATEFONTINDIRECTW	The EMREXTCREATEFONTINDIRECTW structure contains members for the CreateFontIndirect enhanced metafile record.
EMREXTCREATEPEN	The EMREXTCREATEPEN structure contains members for the ExtCreatePen enhanced metafile record. If the record contains a BITMAPINFO structure, it is followed by the bitmap bits that form a packed device-independent bitmap (DIB).
EMREXTFLOODFILL	The EMREXTFLOODFILL structure contains members for the ExtFloodFill enhanced metafile record.
EMREXTSELECTCLIPRGN	The EMREXTSELECTCLIPRGN structure contains members for the ExtSelectClipRgn enhanced metafile record.
EMREXTTEXTOUTA	The EMREXTTEXTOUTA and EMREXTTEXTOUTW structures contain members for the ExtTextOut, TextOut, or DrawText enhanced metafile records.
EMRFILLPATH	The EMRFILLPATH,  EMRSTROKEANDFILLPATH,  and EMRSTROKEPATH structures contain members for the FillPath, StrokeAndFillPath, and StrokePath enhanced metafile records.
EMRFILLRGN	The EMRFILLRGN structure contains members for the FillRgn enhanced metafile record.
EMRFORMAT	The EMRFORMAT structure contains information that identifies graphics data in an enhanced metafile. A GDICOMMENT_MULTIFORMATS enhanced metafile public comment contains an array of EMRFORMAT structures.
EMRFRAMERGN	The EMRFRAMERGN structure contains members for the FrameRgn enhanced metafile record.
EMRGDICOMMENT	The EMRGDICOMMENT structure contains application-specific data.

TITLE	DESCRIPTION
EMRGLSBOUNDEDRECORD	The EMRGLSBOUNDEDRECORD structure contains members for an enhanced metafile record generated by OpenGL functions. It contains data for OpenGL functions with information in pixel units that must be scaled when playing the metafile.
EMRGLSRECORD	The EMRGLSRECORD structure contains members for an enhanced metafile record generated by OpenGL functions. It contains data for OpenGL functions that scale automatically to the OpenGL viewport.
EMRGRADIENTFILL	The EMRGRADIENTFILL structure contains members for the GradientFill enhanced metafile record.
EMRINVERTRGN	The EMRINVERTRGN and EMRPAINTRGN structures contain members for the InvertRgn and PaintRgn enhanced metafile records.
EMRLINETO	The EMRLINETO and EMRMOVETOEX structures contains members for the LineTo and MoveToEx enhanced metafile records.
EMRMASKBLT	The EMRMASKBLT structure contains members for the MaskBlt enhanced metafile record. Note that graphics device interface (GDI) converts the device-dependent bitmap into a device-independent bitmap (DIB) before storing it in the metafile record.
EMRMODIFYWORLDTRANSFORM	The EMRMODIFYWORLDTRANSFORM structure contains members for the ModifyWorldTransform enhanced metafile record.
EMROFFSETCLIPRGN	The EMROFFSETCLIPRGN structure contains members for the OffsetClipRgn enhanced metafile record.
EMRPIXELFORMAT	The EMRPIXELFORMAT structure contains the members for the SetPixelFormat enhanced metafile record. The pixel format information in ENHMETAHEADER refers to this structure.
EMRPLGBLT	The EMRPLGBLT structure contains members for the PlgBlt enhanced metafile record. Note that graphics device interface (GDI) converts the device-dependent bitmap into a device-independent bitmap (DIB) before storing it in the metafile record.
EMRPOLYDRAW	The EMRPOLYDRAW structure contains members for the PolyDraw enhanced metafile record.
EMRPOLYDRAW16	The EMRPOLYDRAW16 structure contains members for the PolyDraw enhanced metafile record.
EMRPOLYLINE	The EMRPOLYLINE, EMRPOLYBEZIER, EMRPOLYGON, EMRPOLYBEZIERTO, and EMRPOLYLINETO structures contain members for the Polyline, PolyBezier, Polygon, PolyBezierTo, and PolylineTo enhanced metafile records.

TITLE	DESCRIPTION
EMRPOLYLINE16	The EMRPOLYLINE16, EMRPOLYBEZIER16, EMRPOLYGON16, EMRPOLYBEZIERTO16, and EMRPOLYLINETO16 structures contain members for the Polyline, PolyBezier, Polygon, PolyBezierTo, and PolylineTo enhanced metafile records.
EMRPOLYPOLYLINE	The EMRPOLYPOLYLINE and EMRPOLYPOLYGON structures contain members for the PolyPolyline and PolyPolygon enhanced metafile records.
EMRPOLYPOLYLINE16	The EMRPOLYPOLYLINE16 and EMRPOLYPOLYGON16 structures contain members for the PolyPolyline and PolyPolygon enhanced metafile records.
EMRPOLYTEXTOUTA	The EMRPOLYTEXTOUTA and EMRPOLYTEXTOUTW structures contain members for the PolyTextOut enhanced metafile record.
EMRRESIZEPALETTE	The EMRRESIZEPALETTE structure contains members for the ResizePalette enhanced metafile record.
EMRRESTOREDC	The EMRRESTOREDC structure contains members for the RestoreDC enhanced metafile record.
EMRROUNDRECT	The EMRROUNDRECT structure contains members for the RoundRect enhanced metafile record.
EMRSCALEVIEWPORTEXT	The EMRSCALEVIEWPORTEXT and EMRSCALEWINDOWEXT structures contain members for the ScaleViewportExtEx and ScaleWindowExtEx enhanced metafile records.
EMRSELECTCLIPPATH	Contains parameters for the SelectClipPath, SetBkMode, SetMapMode, SetPolyFillMode, SetROP2, SetStretchBltMode, SetTextAlign, SetICMMode, and SetLayout enhanced metafile records.
EMRSELECTOBJECT	The EMRSELECTOBJECT and EMRDELETEOBJECT structures contain members for the SelectObject and DeleteObject enhanced metafile records.
EMRSELECTPALETTE	The EMRSELECTPALETTE structure contains members for the SelectPalette enhanced metafile record. Note that the bForceBackground parameter in SelectPalette is always recorded as TRUE, which causes the palette to be realized as a background palette.
EMRSETARCDIRECTION	The EMRSETARCDIRECTION structure contains members for the SetArcDirection enhanced metafile record.
EMRSETBKCOLOR	The EMRSETBKCOLOR and EMRSETTEXTCOLOR structures contain members for the SetBkColor and SetTextColor enhanced metafile records.

TITLE	DESCRIPTION
EMRSETCOLORADJUSTMENT	The EMRSETCOLORADJUSTMENT structure contains members for the SetColorAdjustment enhanced metafile record.
EMRSETCOLORSPACE	The EMRSETCOLORSPACE, EMRSELECTCOLORSPACE, and EMRDELETIColorSPACE structures contain members for the SetColorSpace and DeleteColorSpace enhanced metafile records.
EMRSETDIBITSTODEVICE	The EMRSETDIBITSTODEVICE structure contains members for the SetDIBitsToDevice enhanced metafile record.
EMRSETICMPROFILE	The EMRSETICMPROFILE structure contains members for the SetICMProfile enhanced metafile record.
EMRSETMAPPERFLAGS	The EMRSETMAPPERFLAGS structure contains members for the SetMapperFlags enhanced metafile record.
EMRSETMITERLIMIT	The EMRSETMITERLIMIT structure contains members for the SetMiterLimit enhanced metafile record.
EMRSETPALETTEENTRIES	The EMRSETPALETTEENTRIES structure contains members for the SetPaletteEntries enhanced metafile record.
EMRSETPIXELV	The EMRSETPIXELV structure contains members for the SetPixelV enhanced metafile record. When an enhanced metafile is created, calls to SetPixel are also recorded in this record.
EMRSETVIEWPORTEXTEX	The EMRSETVIEWPORTEXTEX and EMRSETWINDOWEXTEX structures contain members for the SetViewportExtEx and SetWindowExtEx enhanced metafile records.
EMRSETVIEWPORTORGEX	The EMRSETVIEWPORTORGEX, EMRSETWINDOWORGEX, and EMRSETBRUSHORGEX structures contain members for the SetViewportOrgEx, SetWindowOrgEx, and SetBrushOrgEx enhanced metafile records.
EMRSETWORLDTRANSFORM	The EMRSETWORLDTRANSFORM structure contains members for the SetWorldTransform enhanced metafile record.
EMRSTRETCHBLT	The EMRSTRETCHBLT structure contains members for the StretchBlt enhanced metafile record. Note that graphics device interface (GDI) converts the device-dependent bitmap into a device-independent bitmap (DIB) before storing it in the metafile record.
EMRSTRETCHDIBITS	The EMRSTRETCHDIBITS structure contains members for the StretchDIBits enhanced metafile record.
EMRTEXT	The EMRTEXT structure contains members for text output.
EMRTRANSPARENTBLT	The EMRTRANSPARENTBLT structure contains members for the TransparentBLT enhanced metafile record.

TITLE	DESCRIPTION
ENHMETAHEADER	The ENHMETAHEADER structure contains enhanced-metafile data such as the dimensions of the picture stored in the enhanced metafile, the count of records in the enhanced metafile, the resolution of the device on which the picture was created, and so on. This structure is always the first record in an enhanced metafile.
ENHMETARECORD	The ENHMETARECORD structure contains data that describes a graphics device interface (GDI) function used to create part of a picture in an enhanced-format metafile.
ENUMLOGFONTA	The ENUMLOGFONT structure defines the attributes of a font, the complete name of a font, and the style of a font.
ENUMLOGFONTEXA	The ENUMLOGFONTEX structure contains information about an enumerated font.
ENUMLOGFONTEXDVA	The ENUMLOGFONTEXDV structure contains the information used to create a font.
ENUMLOGFONTEXDVW	The ENUMLOGFONTEXDV structure contains the information used to create a font.
ENUMLOGFONTEXW	The ENUMLOGFONTEX structure contains information about an enumerated font.
ENUMLOGFONTW	The ENUMLOGFONT structure defines the attributes of a font, the complete name of a font, and the style of a font.
ENUMTEXTMETRICA	The ENUMTEXTMETRIC structure contains information about a physical font.
ENUMTEXTMETRICW	The ENUMTEXTMETRIC structure contains information about a physical font.
EXTLOGFONTA	The EXTLOGFONT structure defines the attributes of a font.
EXTLOGFONTW	The EXTLOGFONT structure defines the attributes of a font.
EXTLOGPEN	The EXTLOGPEN structure defines the pen style, width, and brush attributes for an extended pen.
FIXED	The FIXED structure contains the integral and fractional parts of a fixed-point real number.
FONTSIGNATURE	Contains information identifying the code pages and Unicode subranges for which a given font provides glyphs.
GCP_RESULTSA	The GCP_RESULTS structure contains information about characters in a string. This structure receives the results of the GetCharacterPlacement function. For some languages, the first element in the arrays may contain more, language-dependent information.

TITLE	DESCRIPTION
GCP_RESULTSW	The GCP_RESULTS structure contains information about characters in a string. This structure receives the results of the GetCharacterPlacement function. For some languages, the first element in the arrays may contain more, language-dependent information.
GLYPHMETRICS	The GLYPHMETRICS structure contains information about the placement and orientation of a glyph in a character cell.
GLYPHMETRICSFLOAT	The GLYPHMETRICSFLOAT structure contains information about the placement and orientation of a glyph in a character cell.
GLYPHSET	The GLYPHSET structure contains information about a range of Unicode code points.
GRADIENT_RECT	The GRADIENT_RECT structure specifies the index of two vertices in the pVertex array in the GradientFill function. These two vertices form the upper-left and lower-right boundaries of a rectangle.
GRADIENT_TRIANGLE	The GRADIENT_TRIANGLE structure specifies the index of three vertices in the pVertex array in the GradientFill function. These three vertices form one triangle.
HANDLETABLE	The HANDLETABLE structure is an array of handles, each of which identifies a graphics device interface (GDI) object.
KERNINGPAIR	The KERNINGPAIR structure defines a kerning pair.
LAYERPLANEDESCRIPTOR	The LAYERPLANEDESCRIPTOR structure describes the pixel format of a drawing surface.
LOCALESIGNATURE	Contains extended font signature information, including two code page bitfields (CPBs) that define the default and supported character sets and code pages. This structure is typically used to represent the relationships between font coverage and locales.
LOGBRUSH	The LOGBRUSH structure defines the style, color, and pattern of a physical brush. It is used by the CreateBrushIndirect and ExtCreatePen functions.
LOGBRUSH32	The LOGBRUSH32 structure defines the style, color, and pattern of a physical brush.
LOGCOLORSPACEA	The LOGCOLORSPACE structure contains information that defines a logical color space.
LOGCOLORSPACEW	The LOGCOLORSPACE structure contains information that defines a logical color space.
LOGFONTA	The LOGFONT structure defines the attributes of a font.
LOGFONTW	The LOGFONT structure defines the attributes of a font.

TITLE	DESCRIPTION
LOGPALETTE	The LOGPALETTE structure defines a logical palette.
LOGPEN	The LOGPEN structure defines the style, width, and color of a pen. The CreatePenIndirect function uses the LOGPEN structure.
MAT2	The MAT2 structure contains the values for a transformation matrix used by the GetGlyphOutline function.
METAFILEPICT	Defines the metafile picture format used for exchanging metafile data through the clipboard.
METAHEADER	The METAHEADER structure contains information about a Windows-format metafile.
METARECORD	The METARECORD structure contains a Windows-format metafile record.
NEWTEXTMETRICA	The NEWTEXTMETRIC structure contains data that describes a physical font.
NEWTEXTMETRICEXA	The NEWTEXTMETRICEX structure contains information about a physical font.
NEWTEXTMETRICEXW	The NEWTEXTMETRICEX structure contains information about a physical font.
NEWTEXTMETRICW	The NEWTEXTMETRIC structure contains data that describes a physical font.
OUTLINETEXTMETRICA	The OUTLINETEXTMETRIC structure contains metrics describing a TrueType font.
OUTLINETEXTMETRICW	The OUTLINETEXTMETRIC structure contains metrics describing a TrueType font.
PALETTEENTRY	Specifies the color and usage of an entry in a logical palette.
PANOSE	The PANOSE structure describes the PANOSE font-classification values for a TrueType font. These characteristics are then used to associate the font with other fonts of similar appearance but different names.
PIXELFORMATDESCRIPTOR	The PIXELFORMATDESCRIPTOR structure describes the pixel format of a drawing surface.
POINTFLOAT	The POINTFLOAT structure contains the x and y coordinates of a point.
POINTFX	The POINTFX structure contains the coordinates of points that describe the outline of a character in a TrueType font.

TITLE	DESCRIPTION
POLYTEXTA	The POLYTEXT structure describes how the PolyTextOut function should draw a string of text.
POLYTEXTW	The POLYTEXT structure describes how the PolyTextOut function should draw a string of text.
PSFEATURE_CUSTPAPER	The PSFEATURE_CUSTPAPER structure contains information about a custom paper size for a PostScript driver. This structure is used with the GET_PS_FEATURESETTING printer escape function.
PSFEATURE_OUTPUT	The PSFEATURE_OUTPUT structure contains information about PostScript driver output options. This structure is used with the GET_PS_FEATURESETTING printer escape function.
PSINJECTDATA	The PSINJECTDATA structure is a header for the input buffer used with the POSTSCRIPT_INJECTION printer escape function.
RASTERIZER_STATUS	The RASTERIZER_STATUS structure contains information about whether TrueType is installed. This structure is filled when an application calls the GetRasterizerCaps function.
RGBQUAD	The RGBQUAD structure describes a color consisting of relative intensities of red, green, and blue.
RGBTRIPLE	The RGBTRIPLE structure describes a color consisting of relative intensities of red, green, and blue. The bmciColors member of the BITMAPCOREINFO structure consists of an array of RGBTRIPLE structures.
RGNDATA	The RGNDATA structure contains a header and an array of rectangles that compose a region. The rectangles are sorted top to bottom, left to right. They do not overlap.
RGNDATAHEADER	The RGNDATAHEADER structure describes the data returned by the GetRegionData function.
TEXTMETRICA	The TEXTMETRIC structure contains basic information about a physical font. All sizes are specified in logical units; that is, they depend on the current mapping mode of the display context.
TEXTMETRICW	The TEXTMETRIC structure contains basic information about a physical font. All sizes are specified in logical units; that is, they depend on the current mapping mode of the display context.
TRIVERTEX	The TRIVERTEX structure contains color information and position information.
TPOLYCURVE	The TPOLYCURVE structure contains information about a curve in the outline of a TrueType character.



TITLE	DESCRIPTION
<a href="#">TTPOLYGONHEADER</a>	The TTPOLYGONHEADER structure specifies the starting position and type of a contour in a TrueType character outline.
<a href="#">WCRANGE</a>	The WCRANGE structure specifies a range of Unicode characters.
<a href="#">XFORM</a>	The XFORM structure specifies a world-space to page-space transformation.

## Enumerations

TITLE	DESCRIPTION
<a href="#">DISPLAYCONFIG_DEVICE_INFO_TYPE</a>	The DISPLAYCONFIG_DEVICE_INFO_TYPE enumeration specifies the type of display device info to configure or obtain through the DisplayConfigSetDeviceInfo or DisplayConfigGetDeviceInfo function.
<a href="#">DISPLAYCONFIG_MODE_INFO_TYPE</a>	The DISPLAYCONFIG_MODE_INFO_TYPE enumeration specifies that the information that is contained within the DISPLAYCONFIG_MODE_INFO structure is either source or target mode.
<a href="#">DISPLAYCONFIG_PIXELFORMAT</a>	The DISPLAYCONFIG_PIXELFORMAT enumeration specifies pixel format in various bits per pixel (BPP) values.
<a href="#">DISPLAYCONFIG_ROTATION</a>	The DISPLAYCONFIG_ROTATION enumeration specifies the clockwise rotation of the display.
<a href="#">DISPLAYCONFIG_SCALING</a>	The DISPLAYCONFIG_SCALING enumeration specifies the scaling transformation applied to content displayed on a video present network (VidPN) present path.
<a href="#">DISPLAYCONFIG_SCANLINE_ORDERING</a>	The DISPLAYCONFIG_SCANLINE_ORDERING enumeration specifies the method that the display uses to create an image on a screen.
<a href="#">DISPLAYCONFIG_TOPOLOGY_ID</a>	The DISPLAYCONFIG_TOPOLOGY_ID enumeration specifies the type of display topology.
<a href="#">DISPLAYCONFIG_VIDEO_OUTPUT_TECHNOLOGY</a>	The DISPLAYCONFIG_VIDEO_OUTPUT_TECHNOLOGY enumeration specifies the target's connector type.

# METAFILEPICT structure (wingdi.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Defines the metafile picture format used for exchanging metafile data through the clipboard.

## Syntax

```
typedef struct tagMETAFILEPICT {  
    LONG        mm;  
    LONG        xExt;  
    LONG        yExt;  
    HMETAFILE    hMF;  
} METAFILEPICT, *LPMETAFILEPICT;
```

## Members

mm

Type: **LONG**

The mapping mode in which the picture is drawn.

xExt

Type: **LONG**

The size of the metafile picture for all modes except the **MM\_ISOTROPIC** and **MM\_ANISOTROPIC** modes. (For more information about these modes, see the **yExt** member.) The x-extent specifies the width of the rectangle within which the picture is drawn. The coordinates are in units that correspond to the mapping mode.

yExt

Type: **LONG**

The size of the metafile picture for all modes except the **MM\_ISOTROPIC** and **MM\_ANISOTROPIC** modes. The y-extent specifies the height of the rectangle within which the picture is drawn. The coordinates are in units that correspond to the mapping mode. For **MM\_ISOTROPIC** and **MM\_ANISOTROPIC** modes, which can be scaled, the **xExt** and **yExt** members contain an optional suggested size in **MM\_HIMETRIC** units. For **MM\_ANISOTROPIC** pictures, **xExt** and **yExt** can be zero when no suggested size is supplied. For **MM\_ISOTROPIC** pictures, an aspect ratio must be supplied even when no suggested size is given. (If a suggested size is given, the aspect ratio is implied by the size.) To give an aspect ratio without implying a suggested size, set **xExt** and **yExt** to negative values whose ratio is the appropriate aspect ratio. The magnitude of the negative **xExt** and **yExt** values is ignored; only the ratio is used.

hMF

Type: **HEMETAFILE**

A handle to a memory metafile.

## Requirements

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

# See also

[Clipboard](#)

Conceptual

Reference

[SetClipboardData](#)

# winuser.h header

1/15/2021 • 78 minutes to read • [Edit Online](#)

This header is used by Windows Controls. For more information, see:

- [Windows Controls](#) winuser.h contains the following programming interfaces:

## Functions

TITLE	DESCRIPTION
<a href="#">ActivateKeyboardLayout</a>	Sets the input locale identifier (formerly called the keyboard layout handle) for the calling thread or the current process. The input locale identifier specifies a locale as well as the physical layout of the keyboard.
<a href="#">AddClipboardFormatListener</a>	Places the given window in the system-maintained clipboard format listener list.
<a href="#">AdjustWindowRect</a>	Calculates the required size of the window rectangle, based on the desired client-rectangle size. The window rectangle can then be passed to the <code>CreateWindow</code> function to create a window whose client area is the desired size.
<a href="#">AdjustWindowRectEx</a>	Calculates the required size of the window rectangle, based on the desired size of the client rectangle. The window rectangle can then be passed to the <code>CreateWindowEx</code> function to create a window whose client area is the desired size.
<a href="#">AdjustWindowRectExForDpi</a>	Calculates the required size of the window rectangle, based on the desired size of the client rectangle and the provided DPI.
<a href="#">AllowSetForegroundWindow</a>	Enables the specified process to set the foreground window using the <code>SetForegroundWindow</code> function. The calling process must already be able to set the foreground window. For more information, see Remarks later in this topic.
<a href="#">AnimateWindow</a>	Enables you to produce special effects when showing or hiding windows. There are four types of animation: <code>_roll</code> , <code>slide</code> , <code>collapse</code> or <code>expand</code> , and <code>alpha-blended fade</code> .
<a href="#">AnyPopup</a>	Indicates whether an owned, visible, top-level pop-up, or overlapped window exists on the screen. The function searches the entire screen, not just the calling application's client area.
<a href="#">AppendMenuA</a>	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.

TITLE	DESCRIPTION
<a href="#">AppendMenuW</a>	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.
<a href="#">AreDpiAwarenessContextsEqual</a>	Determines whether two DPI_AWARENESS_CONTEXT values are identical.
<a href="#">ArrangeIconicWindows</a>	Arranges all the minimized (iconic) child windows of the specified parent window.
<a href="#">AttachThreadInput</a>	Attaches or detaches the input processing mechanism of one thread to that of another thread.
<a href="#">BeginDeferWindowPos</a>	Allocates memory for a multiple-window- position structure and returns the handle to the structure.
<a href="#">BeginPaint</a>	The BeginPaint function prepares the specified window for painting and fills a PAINTSTRUCT structure with information about the painting.
<a href="#">BlockInput</a>	Blocks keyboard and mouse input events from reaching applications.
<a href="#">BringWindowToTop</a>	Brings the specified window to the top of the Z order. If the window is a top-level window, it is activated. If the window is a child window, the top-level parent window associated with the child window is activated.
<a href="#">BroadcastSystemMessage</a>	Sends a message to the specified recipients.
<a href="#">BroadcastSystemMessageExA</a>	Sends a message to the specified recipients.
<a href="#">BroadcastSystemMessageExW</a>	Sends a message to the specified recipients.
<a href="#">BroadcastSystemMessageW</a>	Sends a message to the specified recipients.
<a href="#">CalculatePopupWindowPosition</a>	Calculates an appropriate pop-up window position using the specified anchor point, pop-up window size, flags, and the optional exclude rectangle.
<a href="#">CallMsgFilterA</a>	Passes the specified message and hook code to the hook procedures associated with the WH_SYSMSGFILTER and WH_MSGFILTER hooks.
<a href="#">CallMsgFilterW</a>	Passes the specified message and hook code to the hook procedures associated with the WH_SYSMSGFILTER and WH_MSGFILTER hooks.
<a href="#">CallNextHookEx</a>	Passes the hook information to the next hook procedure in the current hook chain. A hook procedure can call this function either before or after processing the hook information.


TITLE	DESCRIPTION
<a href="#">CallWindowProcA</a>	Passes message information to the specified window procedure.
<a href="#">CallWindowProcW</a>	Passes message information to the specified window procedure.
<a href="#">CascadeWindows</a>	Cascades the specified child windows of the specified parent window.
<a href="#">ChangeClipboardChain</a>	Removes a specified window from the chain of clipboard viewers.
<a href="#">ChangeDisplaySettingsA</a>	The ChangeDisplaySettings function changes the settings of the default display device to the specified graphics mode.
<a href="#">ChangeDisplaySettingsExA</a>	The ChangeDisplaySettingsEx function changes the settings of the specified display device to the specified graphics mode.
<a href="#">ChangeDisplaySettingsExW</a>	The ChangeDisplaySettingsEx function changes the settings of the specified display device to the specified graphics mode.
<a href="#">ChangeDisplaySettingsW</a>	The ChangeDisplaySettings function changes the settings of the default display device to the specified graphics mode.
<a href="#">ChangeWindowMessageFilter</a>	Adds or removes a message from the User Interface Privilege Isolation (UIPI) message filter.
<a href="#">ChangeWindowMessageFilterEx</a>	Modifies the User Interface Privilege Isolation (UIPI) message filter for a specified window.
<a href="#">CharLowerA</a>	Converts a character string or a single character to lowercase. If the operand is a character string, the function converts the characters in place.
<a href="#">CharLowerBuffA</a>	Converts uppercase characters in a buffer to lowercase characters. The function converts the characters in place.
<a href="#">CharLowerBuffW</a>	Converts uppercase characters in a buffer to lowercase characters. The function converts the characters in place.
<a href="#">CharLowerW</a>	Converts a character string or a single character to lowercase. If the operand is a character string, the function converts the characters in place.
<a href="#">CharNextA</a>	Retrieves a pointer to the next character in a string. This function can handle strings consisting of either single- or multi-byte characters.
<a href="#">CharNextExA</a>	Retrieves the pointer to the next character in a string. This function can handle strings consisting of either single- or multi-byte characters.

TITLE	DESCRIPTION
<a href="#">CharNextW</a>	Retrieves a pointer to the next character in a string. This function can handle strings consisting of either single- or multi-byte characters.
<a href="#">CharPrevA</a>	Retrieves a pointer to the preceding character in a string. This function can handle strings consisting of either single- or multi-byte characters.
<a href="#">CharPrevExA</a>	Retrieves the pointer to the preceding character in a string. This function can handle strings consisting of either single- or multi-byte characters.
<a href="#">CharPrevW</a>	Retrieves a pointer to the preceding character in a string. This function can handle strings consisting of either single- or multi-byte characters.
<a href="#">CharToOemA</a>	Translates a string into the OEM-defined character set. Warning Do not use.
<a href="#">CharToOemBuffA</a>	Translates a specified number of characters in a string into the OEM-defined character set.
<a href="#">CharToOemBuffW</a>	Translates a specified number of characters in a string into the OEM-defined character set.
<a href="#">CharToOemW</a>	Translates a string into the OEM-defined character set. Warning Do not use.
<a href="#">CharUpperA</a>	Converts a character string or a single character to uppercase. If the operand is a character string, the function converts the characters in place.
<a href="#">CharUpperBuffA</a>	Converts lowercase characters in a buffer to uppercase characters. The function converts the characters in place.
<a href="#">CharUpperBuffW</a>	Converts lowercase characters in a buffer to uppercase characters. The function converts the characters in place.
<a href="#">CharUpperW</a>	Converts a character string or a single character to uppercase. If the operand is a character string, the function converts the characters in place.
<a href="#">CheckDlgButton</a>	Changes the check state of a button control.
<a href="#">CheckMenuItem</a>	Sets the state of the specified menu item's check-mark attribute to either selected or clear.
<a href="#">CheckMenuRadioItem</a>	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.
<a href="#">CheckRadioButton</a>	Adds a check mark to (checks) a specified radio button in a group and removes a check mark from (clears) all other radio buttons in the group.

TITLE	DESCRIPTION
<a href="#">ChildWindowFromPoint</a>	Determines which, if any, of the child windows belonging to a parent window contains the specified point. The search is restricted to immediate child windows. Grandchildren, and deeper descendant windows are not searched.
<a href="#">ChildWindowFromPointEx</a>	Determines which, if any, of the child windows belonging to the specified parent window contains the specified point.
<a href="#">ClientToScreen</a>	The ClientToScreen function converts the client-area coordinates of a specified point to screen coordinates.
<a href="#">ClipCursor</a>	Confines the cursor to a rectangular area on the screen.
<a href="#">CloseClipboard</a>	Closes the clipboard.
<a href="#">CloseDesktop</a>	Closes an open handle to a desktop object.
<a href="#">CloseGestureInfoHandle</a>	Closes resources associated with a gesture information handle.
<a href="#">CloseTouchInputHandle</a>	Closes a touch input handle, frees process memory associated with it, and invalidates the handle.
<a href="#">CloseWindow</a>	Minimizes (but does not destroy) the specified window.
<a href="#">CloseWindowStation</a>	Closes an open window station handle.
<a href="#">CopyAcceleratorTableA</a>	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.
<a href="#">CopyAcceleratorTableW</a>	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.
<a href="#">CopyCursor</a>	Copies the specified cursor.
<a href="#">CopyIcon</a>	Copies the specified icon from another module to the current module.
<a href="#">CopyImage</a>	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.
<a href="#">CopyRect</a>	The CopyRect function copies the coordinates of one rectangle to another.



TITLE	DESCRIPTION
<a href="#">CountClipboardFormats</a>	Retrieves the number of different data formats currently on the clipboard.
<a href="#">CreateAcceleratorTableA</a>	Creates an accelerator table.
<a href="#">CreateAcceleratorTableW</a>	Creates an accelerator table.
<a href="#">CreateCaret</a>	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.
<a href="#">CreateCursor</a>	Creates a cursor having the specified size, bit patterns, and hot spot.
<a href="#">CreateDesktopA</a>	Creates a new desktop, associates it with the current window station of the calling process, and assigns it to the calling thread.
<a href="#">CreateDesktopExA</a>	Creates a new desktop with the specified heap, associates it with the current window station of the calling process, and assigns it to the calling thread.
<a href="#">CreateDesktopExW</a>	Creates a new desktop with the specified heap, associates it with the current window station of the calling process, and assigns it to the calling thread.
<a href="#">CreateDesktopW</a>	Creates a new desktop, associates it with the current window station of the calling process, and assigns it to the calling thread.
<a href="#">CreateDialogA</a>	Creates a modeless dialog box from a dialog box template resource. The CreateDialog macro uses the CreateDialogParam function.
<a href="#">CreateDialogIndirectA</a>	Creates a modeless dialog box from a dialog box template in memory. The CreateDialogIndirect macro uses the CreateDialogIndirectParam function.
<a href="#">CreateDialogIndirectParamA</a>	Creates a modeless dialog box from a dialog box template in memory.
<a href="#">CreateDialogIndirectParamW</a>	Creates a modeless dialog box from a dialog box template in memory.
<a href="#">CreateDialogIndirectW</a>	Creates a modeless dialog box from a dialog box template in memory. The CreateDialogIndirect macro uses the CreateDialogIndirectParam function.
<a href="#">CreateDialogParamA</a>	Creates a modeless dialog box from a dialog box template resource.
<a href="#">CreateDialogParamW</a>	Creates a modeless dialog box from a dialog box template resource.

TITLE	DESCRIPTION
<a href="#">CreateDialogW</a>	Creates a modeless dialog box from a dialog box template resource. The CreateDialog macro uses the CreateDialogParam function.
<a href="#">CreateIcon</a>	Creates an icon that has the specified size, colors, and bit patterns.
<a href="#">CreateIconFromResource</a>	Creates an icon or cursor from resource bits describing the icon.
<a href="#">CreateIconFromResourceEx</a>	Creates an icon or cursor from resource bits describing the icon.
<a href="#">CreateIconIndirect</a>	Creates an icon or cursor from an ICONINFO structure.
<a href="#">CreateMDIWindowA</a>	Creates a multiple-document interface (MDI) child window.
<a href="#">CreateMDIWindowW</a>	Creates a multiple-document interface (MDI) child window.
<a href="#">CreateMenu</a>	Creates a menu. The menu is initially empty, but it can be filled with menu items by using the InsertMenuItem, AppendMenu, and InsertMenu functions.
<a href="#">CreatePopupMenu</a>	Creates a drop-down menu, submenu, or shortcut menu.
<a href="#">CreateSyntheticPointerDevice</a>	Configures the pointer injection device for the calling application, and initializes the maximum number of simultaneous pointers that the app can inject.
<a href="#">CreateWindowA</a>	Creates an overlapped, pop-up, or child window.
<a href="#">CreateWindowExA</a>	Creates an overlapped, pop-up, or child window with an extended window style; otherwise, this function is identical to the CreateWindow function.
<a href="#">CreateWindowExW</a>	Creates an overlapped, pop-up, or child window with an extended window style; otherwise, this function is identical to the CreateWindow function.
<a href="#">CreateWindowStationA</a>	Creates a window station object, associates it with the calling process, and assigns it to the current session.
<a href="#">CreateWindowStationW</a>	Creates a window station object, associates it with the calling process, and assigns it to the current session.
<a href="#">CreateWindowW</a>	Creates an overlapped, pop-up, or child window.
<a href="#">DefDlgProcW</a>	Calls the default dialog box window procedure to provide default processing for any window messages that a dialog box with a private window class does not process.
<a href="#">DeferWindowPos</a>	Updates the specified multiple-window  position structure for the specified window.

TITLE	DESCRIPTION
<a href="#">DefFrameProcA</a>	Provides default processing for any window messages that the window procedure of a multiple-document interface (MDI) frame window does not process.
<a href="#">DefFrameProcW</a>	Provides default processing for any window messages that the window procedure of a multiple-document interface (MDI) frame window does not process.
<a href="#">DefMDIChildProcA</a>	Provides default processing for any window message that the window procedure of a multiple-document interface (MDI) child window does not process.
<a href="#">DefMDIChildProcW</a>	Provides default processing for any window message that the window procedure of a multiple-document interface (MDI) child window does not process.
<a href="#">DefRawInputProc</a>	Verifies that the size of the RAWINPUTHEADER structure is correct.
<a href="#">DefWindowProcA</a>	Calls the default window procedure to provide default processing for any window messages that an application does not process.
<a href="#">DefWindowProcW</a>	Calls the default window procedure to provide default processing for any window messages that an application does not process.
<a href="#">DeleteMenu</a>	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.
<a href="#">DeregisterShellHookWindow</a>	Unregisters a specified Shell window that is registered to receive Shell hook messages.
<a href="#">DestroyAcceleratorTable</a>	Destroys an accelerator table.
<a href="#">DestroyCaret</a>	Destroys the caret's current shape, frees the caret from the window, and removes the caret from the screen.
<a href="#">DestroyCursor</a>	Destroys a cursor and frees any memory the cursor occupied. Do not use this function to destroy a shared cursor.
<a href="#">DestroyIcon</a>	Destroys an icon and frees any memory the icon occupied.
<a href="#">DestroyMenu</a>	Destroys the specified menu and frees any memory that the menu occupies.
<a href="#">DestroySyntheticPointerDevice</a>	Destroys the specified pointer injection device.
<a href="#">DestroyWindow</a>	Destroys the specified window.

TITLE	DESCRIPTION
<a href="#">DialogBoxA</a>	Creates a modal dialog box from a dialog box template resource. DialogBox does not return control until the specified callback function terminates the modal dialog box by calling the EndDialog function.
<a href="#">DialogBoxIndirectA</a>	Creates a modal dialog box from a dialog box template in memory. DialogBoxIndirect does not return control until the specified callback function terminates the modal dialog box by calling the EndDialog function.
<a href="#">DialogBoxIndirectParamA</a>	Creates a modal dialog box from a dialog box template in memory.
<a href="#">DialogBoxIndirectParamW</a>	Creates a modal dialog box from a dialog box template in memory.
<a href="#">DialogBoxIndirectW</a>	Creates a modal dialog box from a dialog box template in memory. DialogBoxIndirect does not return control until the specified callback function terminates the modal dialog box by calling the EndDialog function.
<a href="#">DialogBoxParamA</a>	Creates a modal dialog box from a dialog box template resource.
<a href="#">DialogBoxParamW</a>	Creates a modal dialog box from a dialog box template resource.
<a href="#">DialogBoxW</a>	Creates a modal dialog box from a dialog box template resource. DialogBox does not return control until the specified callback function terminates the modal dialog box by calling the EndDialog function.
<a href="#">DisableProcessWindowsGhosting</a>	Disables the window ghosting feature for the calling GUI process. Window ghosting is a Windows Manager feature that lets the user minimize, move, or close the main window of an application that is not responding.
<a href="#">DispatchMessage</a>	Dispatches a message to a window procedure. It is typically used to dispatch a message retrieved by the GetMessage function.
<a href="#">DispatchMessageA</a>	Dispatches a message to a window procedure. It is typically used to dispatch a message retrieved by the GetMessage function.
<a href="#">DispatchMessageW</a>	Dispatches a message to a window procedure. It is typically used to dispatch a message retrieved by the GetMessage function.
<a href="#">DisplayConfigGetDeviceInfo</a>	The DisplayConfigGetDeviceInfo function retrieves display configuration information about the device.
<a href="#">DisplayConfigSetDeviceInfo</a>	The DisplayConfigSetDeviceInfo function sets the properties of a target.

TITLE	DESCRIPTION
<a href="#">DlGDirListA</a>	Replaces the contents of a list box with the names of the subdirectories and files in a specified directory. You can filter the list of names by specifying a set of file attributes. The list can optionally include mapped drives.
<a href="#">DlGDirListComboBoxA</a>	Replaces the contents of a combo box with the names of the subdirectories and files in a specified directory. You can filter the list of names by specifying a set of file attributes. The list of names can include mapped drive letters.
<a href="#">DlGDirListComboBoxW</a>	Replaces the contents of a combo box with the names of the subdirectories and files in a specified directory. You can filter the list of names by specifying a set of file attributes. The list of names can include mapped drive letters.
<a href="#">DlGDirListW</a>	Replaces the contents of a list box with the names of the subdirectories and files in a specified directory. You can filter the list of names by specifying a set of file attributes. The list can optionally include mapped drives.
<a href="#">DlGDirSelectComboBoxExA</a>	Retrieves the current selection from a combo box filled by using the <a href="#">DlGDirListComboBox</a> function. The selection is interpreted as a drive letter, a file, or a directory name.
<a href="#">DlGDirSelectComboBoxExW</a>	Retrieves the current selection from a combo box filled by using the <a href="#">DlGDirListComboBox</a> function. The selection is interpreted as a drive letter, a file, or a directory name.
<a href="#">DlGDirSelectExA</a>	Retrieves the current selection from a single-selection list box. It assumes that the list box has been filled by the <a href="#">DlGDirList</a> function and that the selection is a drive letter, filename, or directory name.
<a href="#">DlGDirSelectExW</a>	Retrieves the current selection from a single-selection list box. It assumes that the list box has been filled by the <a href="#">DlGDirList</a> function and that the selection is a drive letter, filename, or directory name.
<a href="#">DragDetect</a>	Captures the mouse and tracks its movement until the user releases the left button, presses the ESC key, or moves the mouse outside the drag rectangle around the specified point.
<a href="#">DrawAnimatedRects</a>	Animates the caption of a window to indicate the opening of an icon or the minimizing or maximizing of a window.
<a href="#">DrawCaption</a>	The <a href="#">DrawCaption</a> function draws a window caption.
<a href="#">DrawEdge</a>	The <a href="#">DrawEdge</a> function draws one or more edges of rectangle.
<a href="#">DrawFocusRect</a>	The <a href="#">DrawFocusRect</a> function draws a rectangle in the style used to indicate that the rectangle has the focus.

TITLE	DESCRIPTION
<a href="#">DrawFrameControl</a>	The DrawFrameControl function draws a frame control of the specified type and style.
<a href="#">DrawIcon</a>	Draws an icon or cursor into the specified device context.
<a href="#">DrawIconEx</a>	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.
<a href="#">DrawMenuBar</a>	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.
<a href="#">DrawStateA</a>	The DrawState function displays an image and applies a visual effect to indicate a state, such as a disabled or default state.
<a href="#">DrawStateW</a>	The DrawState function displays an image and applies a visual effect to indicate a state, such as a disabled or default state.
<a href="#">DrawText</a>	The DrawText function draws formatted text in the specified rectangle. It formats the text according to the specified method (expanding tabs, justifying characters, breaking lines, and so forth).
<a href="#">DrawTextA</a>	The DrawText function draws formatted text in the specified rectangle. It formats the text according to the specified method (expanding tabs, justifying characters, breaking lines, and so forth).
<a href="#">DrawTextExA</a>	The DrawTextEx function draws formatted text in the specified rectangle.
<a href="#">DrawTextExW</a>	The DrawTextEx function draws formatted text in the specified rectangle.
<a href="#">DrawTextW</a>	The DrawText function draws formatted text in the specified rectangle. It formats the text according to the specified method (expanding tabs, justifying characters, breaking lines, and so forth).
<a href="#">EmptyClipboard</a>	Empties the clipboard and frees handles to data in the clipboard. The function then assigns ownership of the clipboard to the window that currently has the clipboard open.
<a href="#">EnableMenuItem</a>	Enables, disables, or grays the specified menu item.
<a href="#">EnableMouseInPointer</a>	Enables the mouse to act as a pointer input device and send WM_POINTER messages.
<a href="#">EnableNonClientDpiScaling</a>	In high-DPI displays, enables automatic display scaling of the non-client area portions of the specified top-level window. Must be called during the initialization of that window.

TITLE	DESCRIPTION
<a href="#">EnableScrollBar</a>	The EnableScrollBar function enables or disables one or both scroll bar arrows.
<a href="#">EnableWindow</a>	Enables or disables mouse and keyboard input to the specified window or control. When input is disabled, the window does not receive input such as mouse clicks and key presses. When input is enabled, the window receives all input.
<a href="#">EndDeferWindowPos</a>	Simultaneously updates the position and size of one or more windows in a single screen-refreshing cycle.
<a href="#">EndDialog</a>	Destroys a modal dialog box, causing the system to end any processing for the dialog box.
<a href="#">EndMenu</a>	Ends the calling thread's active menu.
<a href="#">EndPaint</a>	The EndPaint function marks the end of painting in the specified window. This function is required for each call to the BeginPaint function, but only after painting is complete.
<a href="#">EndTask</a>	Forcibly closes the specified window.
<a href="#">EnumChildWindows</a>	Enumerates the child windows that belong to the specified parent window by passing the handle to each child window, in turn, to an application-defined callback function.
<a href="#">EnumClipboardFormats</a>	Enumerates the data formats currently available on the clipboard.
<a href="#">EnumDesktopsA</a>	Enumerates all desktops associated with the specified window station of the calling process. The function passes the name of each desktop, in turn, to an application-defined callback function.
<a href="#">EnumDesktopsW</a>	Enumerates all desktops associated with the specified window station of the calling process. The function passes the name of each desktop, in turn, to an application-defined callback function.
<a href="#">EnumDesktopWindows</a>	Enumerates all top-level windows associated with the specified desktop. It passes the handle to each window, in turn, to an application-defined callback function.
<a href="#">EnumDisplayDevicesA</a>	The EnumDisplayDevices function lets you obtain information about the display devices in the current session.
<a href="#">EnumDisplayDevicesW</a>	The EnumDisplayDevices function lets you obtain information about the display devices in the current session.

TITLE	DESCRIPTION
<a href="#">EnumDisplayMonitors</a>	The EnumDisplayMonitors function enumerates display monitors (including invisible pseudo-monitors associated with the mirroring drivers) that intersect a region formed by the intersection of a specified clipping rectangle and the visible region of a device context. EnumDisplayMonitors calls an application-defined MonitorEnumProc callback function once for each monitor that is enumerated. Note that GetSystemMetrics (SM_CMONITORS) counts only the display monitors.
<a href="#">EnumDisplaySettingsA</a>	The EnumDisplaySettings function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes of a display device, make a series of calls to this function.
<a href="#">EnumDisplaySettingsExA</a>	The EnumDisplaySettingsEx function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes for a display device, make a series of calls to this function.
<a href="#">EnumDisplaySettingsExW</a>	The EnumDisplaySettingsEx function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes for a display device, make a series of calls to this function.
<a href="#">EnumDisplaySettingsW</a>	The EnumDisplaySettings function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes of a display device, make a series of calls to this function.
<a href="#">EnumPropsA</a>	Enumerates all entries in the property list of a window by passing them, one by one, to the specified callback function. EnumProps continues until the last entry is enumerated or the callback function returns FALSE.
<a href="#">EnumPropsExA</a>	Enumerates all entries in the property list of a window by passing them, one by one, to the specified callback function. EnumPropsEx continues until the last entry is enumerated or the callback function returns FALSE.
<a href="#">EnumPropsExW</a>	Enumerates all entries in the property list of a window by passing them, one by one, to the specified callback function. EnumPropsEx continues until the last entry is enumerated or the callback function returns FALSE.
<a href="#">EnumPropsW</a>	Enumerates all entries in the property list of a window by passing them, one by one, to the specified callback function. EnumProps continues until the last entry is enumerated or the callback function returns FALSE.
<a href="#">EnumThreadWindows</a>	Enumerates all nonchild windows associated with a thread by passing the handle to each window, in turn, to an application-defined callback function.



TITLE	DESCRIPTION
<a href="#">EnumWindows</a>	Enumerates all top-level windows on the screen by passing the handle to each window, in turn, to an application-defined callback function. EnumWindows continues until the last top-level window is enumerated or the callback function returns FALSE.
<a href="#">EnumWindowStationsA</a>	Enumerates all window stations in the current session. The function passes the name of each window station, in turn, to an application-defined callback function.
<a href="#">EnumWindowStationsW</a>	Enumerates all window stations in the current session. The function passes the name of each window station, in turn, to an application-defined callback function.
<a href="#">EqualRect</a>	The EqualRect function determines whether the two specified rectangles are equal by comparing the coordinates of their upper-left and lower-right corners.
<a href="#">EvaluateProximityToPolygon</a>	Returns the score of a polygon as the probable touch target (compared to all other polygons that intersect the touch contact area) and an adjusted touch point within the polygon.
<a href="#">EvaluateProximityToRect</a>	Returns the score of a rectangle as the probable touch target, compared to all other rectangles that intersect the touch contact area, and an adjusted touch point within the rectangle.
<a href="#">ExcludeUpdateRgn</a>	The ExcludeUpdateRgn function prevents drawing within invalid areas of a window by excluding an updated region in the window from a clipping region.
<a href="#">ExitWindows</a>	Calls the ExitWindowsEx function to log off the interactive user.
<a href="#">ExitWindowsEx</a>	Logs off the interactive user, shuts down the system, or shuts down and restarts the system.
<a href="#">FillRect</a>	The FillRect function fills a rectangle by using the specified brush. This function includes the left and top borders, but excludes the right and bottom borders of the rectangle.
<a href="#">FindWindowA</a>	Retrieves a handle to the top-level window whose class name and window name match the specified strings. This function does not search child windows. This function does not perform a case-sensitive search.
<a href="#">FindWindowExA</a>	Retrieves a handle to a window whose class name and window name match the specified strings. The function searches child windows, beginning with the one following the specified child window. This function does not perform a case-sensitive search.

TITLE	DESCRIPTION
<a href="#">FindWindowExW</a>	Retrieves a handle to a window whose class name and window name match the specified strings. The function searches child windows, beginning with the one following the specified child window. This function does not perform a case-sensitive search.
<a href="#">FindWindowW</a>	Retrieves a handle to the top-level window whose class name and window name match the specified strings. This function does not search child windows. This function does not perform a case-sensitive search.
<a href="#">FlashWindow</a>	Flashes the specified window one time. It does not change the active state of the window.
<a href="#">FlashWindowEx</a>	Flashes the specified window. It does not change the active state of the window.
<a href="#">FrameRect</a>	The FrameRect function draws a border around the specified rectangle by using the specified brush. The width and height of the border are always one logical unit.
<a href="#">GET_APPCOMMAND_LPARAM</a>	Retrieves the application command from the specified LPARAM value.
<a href="#">GET_DEVICE_LPARAM</a>	Retrieves the input device type from the specified LPARAM value.
<a href="#">GET_FLAGS_LPARAM</a>	Retrieves the state of certain virtual keys from the specified LPARAM value.
<a href="#">GET_KEYSTATE_LPARAM</a>	Retrieves the state of certain virtual keys from the specified LPARAM value.
<a href="#">GET_KEYSTATE_WPARAM</a>	Retrieves the state of certain virtual keys from the specified WPARAM value.
<a href="#">GET_NCHITTEST_WPARAM</a>	Retrieves the hit-test value from the specified WPARAM value.
<a href="#">GET_POINTERID_WPARAM</a>	Retrieves the pointer ID using the specified value.
<a href="#">GET_RAWINPUT_CODE_WPARAM</a>	Retrieves the input code from wParam in WM_INPUT.
<a href="#">GET_WHEEL_DELTA_WPARAM</a>	Retrieves the wheel-delta value from the specified WPARAM value.
<a href="#">GET_XBUTTON_WPARAM</a>	Retrieves the state of certain buttons from the specified WPARAM value.
<a href="#">GetActiveWindow</a>	Retrieves the window handle to the active window attached to the calling thread's message queue.
<a href="#">GetAltTabInfoA</a>	Retrieves status information for the specified window if it is the application-switching (ALT+TAB) window.

TITLE	DESCRIPTION
<a href="#">GetAltTabInfoW</a>	Retrieves status information for the specified window if it is the application-switching (ALT + TAB) window.
<a href="#">GetAncestor</a>	Retrieves the handle to the ancestor of the specified window.
<a href="#">GetAsyncKeyState</a>	Determines whether a key is up or down at the time the function is called, and whether the key was pressed after a previous call to <a href="#">GetAsyncKeyState</a> .
<a href="#">GetAutoRotationState</a>	Retrieves an AR_STATE value containing the state of screen auto-rotation for the system, for example whether auto-rotation is supported, and whether it is enabled by the user.
<a href="#">GetAwarenessFromDpiAwarenessContext</a>	Retrieves the DPI_AWARENESS value from a DPI_AWARENESS_CONTEXT.
<a href="#">GetCapture</a>	Retrieves a handle to the window (if any) that has captured the mouse. Only one window at a time can capture the mouse; this window receives mouse input whether or not the cursor is within its borders.
<a href="#">GetCaretBlinkTime</a>	Retrieves the time required to invert the caret's pixels. The user can set this value.
<a href="#">GetCaretPos</a>	Copies the caret's position to the specified POINT structure.
<a href="#">GetCIMSSM</a>	Retrieves the source of the input message ( <a href="#">GetCurrentInputMessageSourceInSendMessage</a> ).
<a href="#">GetClassInfoA</a>	Retrieves information about a window class.
<a href="#">GetClassInfoExA</a>	Retrieves information about a window class, including a handle to the small icon associated with the window class. The <a href="#">GetClassInfo</a> function does not retrieve a handle to the small icon.
<a href="#">GetClassInfoExW</a>	Retrieves information about a window class, including a handle to the small icon associated with the window class. The <a href="#">GetClassInfo</a> function does not retrieve a handle to the small icon.
<a href="#">GetClassInfoW</a>	Retrieves information about a window class.
<a href="#">GetClassLongA</a>	Retrieves the specified 32-bit (DWORD) value from the WNDCLASSEX structure associated with the specified window.
<a href="#">GetClassLongPtrA</a>	Retrieves the specified value from the WNDCLASSEX structure associated with the specified window.
<a href="#">GetClassLongPtrW</a>	Retrieves the specified value from the WNDCLASSEX structure associated with the specified window.

TITLE	DESCRIPTION
<a href="#">GetClassLongW</a>	Retrieves the specified 32-bit (DWORD) value from the WNDCLASSEX structure associated with the specified window.
<a href="#">GetClassName</a>	Retrieves the name of the class to which the specified window belongs.
<a href="#">GetClassNameA</a>	Retrieves the name of the class to which the specified window belongs.
<a href="#">GetClassNameW</a>	Retrieves the name of the class to which the specified window belongs.
<a href="#">GetClassWord</a>	Retrieves the 16-bit (WORD) value at the specified offset into the extra class memory for the window class to which the specified window belongs.
<a href="#">GetClientRect</a>	Retrieves the coordinates of a window's client area.
<a href="#">GetClipboardData</a>	Retrieves data from the clipboard in a specified format. The clipboard must have been opened previously.
<a href="#">GetClipboardFormatNameA</a>	Retrieves from the clipboard the name of the specified registered format. The function copies the name to the specified buffer.
<a href="#">GetClipboardFormatNameW</a>	Retrieves from the clipboard the name of the specified registered format. The function copies the name to the specified buffer.
<a href="#">GetClipboardOwner</a>	Retrieves the window handle of the current owner of the clipboard.
<a href="#">GetClipboardSequenceNumber</a>	Retrieves the clipboard sequence number for the current window station.
<a href="#">GetClipboardViewer</a>	Retrieves the handle to the first window in the clipboard viewer chain.
<a href="#">GetClipCursor</a>	Retrieves the screen coordinates of the rectangular area to which the cursor is confined.
<a href="#">GetComboBoxInfo</a>	Retrieves information about the specified combo box.
<a href="#">GetCurrentInputMessageSource</a>	Retrieves the source of the input message.
<a href="#">GetCursor</a>	Retrieves a handle to the current cursor.
<a href="#">GetCursorInfo</a>	Retrieves information about the global cursor.
<a href="#">GetCursorPos</a>	Retrieves the position of the mouse cursor, in screen coordinates.

TITLE	DESCRIPTION
<a href="#">GetDC</a>	The GetDC function retrieves a handle to a device context (DC) for the client area of a specified window or for the entire screen.
<a href="#">GetDCEx</a>	The GetDCEx function retrieves a handle to a device context (DC) for the client area of a specified window or for the entire screen.
<a href="#">GetDesktopWindow</a>	Retrieves a handle to the desktop window. The desktop window covers the entire screen. The desktop window is the area on top of which other windows are painted.
<a href="#">GetDialogBaseUnits</a>	Retrieves the system's dialog base units, which are the average width and height of characters in the system font.
<a href="#">GetDialogControlDpiChangeBehavior</a>	Retrieves and per-monitor DPI scaling behavior overrides of a child window in a dialog.
<a href="#">GetDialogDpiChangeBehavior</a>	Returns the flags that might have been set on a given dialog by an earlier call to SetDialogDpiChangeBehavior.
<a href="#">GetDisplayAutoRotationPreferences</a>	Retrieves the screen auto-rotation preferences for the current process.
<a href="#">GetDisplayAutoRotationPreferencesByProcessId</a>	Retrieves the screen auto-rotation preferences for the process indicated by the dwProcessId parameter.
<a href="#">GetDisplayConfigBufferSizes</a>	The GetDisplayConfigBufferSizes function retrieves the size of the buffers that are required to call the QueryDisplayConfig function.
<a href="#">GetDlgCtrlID</a>	Retrieves the identifier of the specified control.
<a href="#">GetDlgItem</a>	Retrieves a handle to a control in the specified dialog box.
<a href="#">GetDlgItemInt</a>	Translates the text of a specified control in a dialog box into an integer value.
<a href="#">GetDlgItemTextA</a>	Retrieves the title or text associated with a control in a dialog box.
<a href="#">GetDlgItemTextW</a>	Retrieves the title or text associated with a control in a dialog box.
<a href="#">GetDoubleClickTime</a>	Retrieves the current double-click time for the mouse.
<a href="#">GetDpiForSystem</a>	Returns the system DPI.
<a href="#">GetDpiForWindow</a>	Returns the dots per inch (dpi) value for the associated window.

TITLE	DESCRIPTION
<a href="#">GetDpiFromDpiAwarenessContext</a>	Retrieves the DPI from a given DPI_AWARENESS_CONTEXT handle. This enables you to determine the DPI of a thread without needed to examine a window created within that thread.
<a href="#">GetFocus</a>	Retrieves the handle to the window that has the keyboard focus, if the window is attached to the calling thread's message queue.
<a href="#">GetForegroundWindow</a>	Retrieves a handle to the foreground window (the window with which the user is currently working). The system assigns a slightly higher priority to the thread that creates the foreground window than it does to other threads.
<a href="#">GetGestureConfig</a>	Retrieves the configuration for which Windows Touch gesture messages are sent from a window.
<a href="#">GetGestureExtraArgs</a>	Retrieves additional information about a gesture from its GESTUREINFO handle.
<a href="#">GetGestureInfo</a>	Retrieves a GESTUREINFO structure given a handle to the gesture information.
<a href="#">GetGuiResources</a>	Retrieves the count of handles to graphical user interface (GUI) objects in use by the specified process.
<a href="#">GetGUIThreadInfo</a>	Retrieves information about the active window or a specified GUI thread.
<a href="#">GetIconInfo</a>	Retrieves information about the specified icon or cursor.
<a href="#">GetIconInfoExA</a>	Retrieves information about the specified icon or cursor. GetIconInfoEx extends GetIconInfo by using the newer ICONINFOEX structure.
<a href="#">GetIconInfoExW</a>	Retrieves information about the specified icon or cursor. GetIconInfoEx extends GetIconInfo by using the newer ICONINFOEX structure.
<a href="#">GetInputState</a>	Determines whether there are mouse-button or keyboard messages in the calling thread's message queue.
<a href="#">GetKBCodePage</a>	Retrieves the current code page.
<a href="#">GetKeyboardLayout</a>	Retrieves the active input locale identifier (formerly called the keyboard layout).
<a href="#">GetKeyboardLayoutList</a>	Retrieves the input locale identifiers (formerly called keyboard layout handles) corresponding to the current set of input locales in the system. The function copies the identifiers to the specified buffer.
<a href="#">GetKeyboardLayoutNameA</a>	Retrieves the name of the active input locale identifier (formerly called the keyboard layout) for the system.

TITLE	DESCRIPTION
<a href="#">GetKeyboardLayoutNameW</a>	Retrieves the name of the active input locale identifier (formerly called the keyboard layout) for the system.
<a href="#">GetKeyboardState</a>	Copies the status of the 256 virtual keys to the specified buffer.
<a href="#">GetKeyboardType</a>	Retrieves information about the current keyboard.
<a href="#">GetKeyNameTextA</a>	Retrieves a string that represents the name of a key.
<a href="#">GetKeyNameTextW</a>	Retrieves a string that represents the name of a key.
<a href="#">GetKeyState</a>	Retrieves the status of the specified virtual key. The status specifies whether the key is up, down, or toggled (on, off↕alternating each time the key is pressed).
<a href="#">GetLastActivePopup</a>	Determines which pop-up window owned by the specified window was most recently active.
<a href="#">GetLastInputInfo</a>	Retrieves the time of the last input event.
<a href="#">GetLayeredWindowAttributes</a>	Retrieves the opacity and transparency color key of a layered window.
<a href="#">GetListBoxInfo</a>	Retrieves the number of items per column in a specified list box.
<a href="#">GetMenu</a>	Retrieves a handle to the menu assigned to the specified window.
<a href="#">GetMenuBarInfo</a>	Retrieves information about the specified menu bar.
<a href="#">GetMenuCheckMarkDimensions</a>	Retrieves the dimensions of the default check-mark bitmap.
<a href="#">GetMenuContextHelpId</a>	Retrieves the Help context identifier associated with the specified menu.
<a href="#">GetMenuDefaultItem</a>	Determines the default menu item on the specified menu.
<a href="#">GetMenuInfo</a>	Retrieves information about a specified menu.
<a href="#">GetMenuItemCount</a>	Determines the number of items in the specified menu.
<a href="#">GetMenuItemID</a>	Retrieves the menu item identifier of a menu item located at the specified position in a menu.
<a href="#">GetMenuItemInfoA</a>	Retrieves information about a menu item.
<a href="#">GetMenuItemInfoW</a>	Retrieves information about a menu item.
<a href="#">GetMenuItemRect</a>	Retrieves the bounding rectangle for the specified menu item.

TITLE	DESCRIPTION
<a href="#">GetMenuState</a>	Retrieves the menu flags associated with the specified menu item.
<a href="#">GetMenuStringA</a>	Copies the text string of the specified menu item into the specified buffer.
<a href="#">GetMenuStringW</a>	Copies the text string of the specified menu item into the specified buffer.
<a href="#">GetMessage</a>	Retrieves a message from the calling thread's message queue. The function dispatches incoming sent messages until a posted message is available for retrieval.
<a href="#">GetMessageA</a>	Retrieves a message from the calling thread's message queue. The function dispatches incoming sent messages until a posted message is available for retrieval.
<a href="#">GetMessageExtraInfo</a>	Retrieves the extra message information for the current thread. Extra message information is an application- or driver-defined value associated with the current thread's message queue.
<a href="#">GetMessagePos</a>	Retrieves the cursor position for the last message retrieved by the GetMessage function.
<a href="#">GetMessageTime</a>	Retrieves the message time for the last message retrieved by the GetMessage function.
<a href="#">GetMessageW</a>	Retrieves a message from the calling thread's message queue. The function dispatches incoming sent messages until a posted message is available for retrieval.
<a href="#">GetMonitorInfoA</a>	The GetMonitorInfo function retrieves information about a display monitor.
<a href="#">GetMonitorInfoW</a>	The GetMonitorInfo function retrieves information about a display monitor.
<a href="#">GetMouseMovePointsEx</a>	Retrieves a history of up to 64 previous coordinates of the mouse or pen.
<a href="#">GetNextDlgGroupItem</a>	Retrieves a handle to the first control in a group of controls that precedes (or follows) the specified control in a dialog box.
<a href="#">GetNextDlgTabItem</a>	Retrieves a handle to the first control that has the WS_TABSTOP style that precedes (or follows) the specified control.
<a href="#">GetNextWindow</a>	Retrieves a handle to the next or previous window in the Z-Order. The next window is below the specified window; the previous window is above.
<a href="#">GetOpenClipboardWindow</a>	Retrieves the handle to the window that currently has the clipboard open.



TITLE	DESCRIPTION
GetParent	Retrieves a handle to the specified window's parent or owner.
GetPhysicalCursorPos	Retrieves the position of the cursor in physical coordinates.
GetPointerCursorId	Retrieves the cursor identifier associated with the specified pointer.
GetPointerDevice	Gets information about the pointer device.
GetPointerDeviceCursors	Gets the cursor IDs that are mapped to the cursors associated with a pointer device.
GetPointerDeviceProperties	Gets device properties that aren't included in the POINTER_DEVICE_INFO structure.
GetPointerDeviceRects	Gets the x and y range for the pointer device (in himetric) and the x and y range (current resolution) for the display that the pointer device is mapped to.
GetPointerDevices	Gets information about the pointer devices attached to the system.
GetPointerFrameInfo	Gets the entire frame of information for the specified pointers associated with the current message.
GetPointerFrameInfoHistory	Gets the entire frame of information (including coalesced input frames) for the specified pointers associated with the current message.
GetPointerFramePenInfo	Gets the entire frame of pen-based information for the specified pointers (of type PT_PEN) associated with the current message.
GetPointerFramePenInfoHistory	Gets the entire frame of pen-based information (including coalesced input frames) for the specified pointers (of type PT_PEN) associated with the current message.
GetPointerFrameTouchInfo	Gets the entire frame of touch-based information for the specified pointers (of type PT_TOUCH) associated with the current message.
GetPointerFrameTouchInfoHistory	Gets the entire frame of touch-based information (including coalesced input frames) for the specified pointers (of type PT_TOUCH) associated with the current message.
GetPointerInfo	Gets the information for the specified pointer associated with the current message.
GetPointerInfoHistory	Gets the information associated with the individual inputs, if any, that were coalesced into the current message for the specified pointer.

TITLE	DESCRIPTION
<a href="#">GetPointerInputTransform</a>	Gets one or more transforms for the pointer information coordinates associated with the current message.
<a href="#">GetPointerPenInfo</a>	Gets the pen-based information for the specified pointer (of type PT_PEN) associated with the current message.
<a href="#">GetPointerPenInfoHistory</a>	Gets the pen-based information associated with the individual inputs, if any, that were coalesced into the current message for the specified pointer (of type PT_PEN).
<a href="#">GetPointerTouchInfo</a>	Gets the touch-based information for the specified pointer (of type PT_TOUCH) associated with the current message.
<a href="#">GetPointerTouchInfoHistory</a>	Gets the touch-based information associated with the individual inputs, if any, that were coalesced into the current message for the specified pointer (of type PT_TOUCH).
<a href="#">GetPointerType</a>	Retrieves the pointer type for a specified pointer.
<a href="#">GetPriorityClipboardFormat</a>	Retrieves the first available clipboard format in the specified list.
<a href="#">GetProcessDefaultLayout</a>	Retrieves the default layout that is used when windows are created with no parent or owner.
<a href="#">GetProcessWindowStation</a>	Retrieves a handle to the current window station for the calling process.
<a href="#">GetPropA</a>	Retrieves a data handle from the property list of the specified window. The character string identifies the handle to be retrieved. The string and handle must have been added to the property list by a previous call to the SetProp function.
<a href="#">GetPropW</a>	Retrieves a data handle from the property list of the specified window. The character string identifies the handle to be retrieved. The string and handle must have been added to the property list by a previous call to the SetProp function.
<a href="#">GetQueueStatus</a>	Retrieves the type of messages found in the calling thread's message queue.
<a href="#">GetRawInputBuffer</a>	Performs a buffered read of the raw input data.
<a href="#">GetRawInputData</a>	Retrieves the raw input from the specified device.
<a href="#">GetRawInputDeviceInfoA</a>	Retrieves information about the raw input device.
<a href="#">GetRawInputDeviceInfoW</a>	Retrieves information about the raw input device.
<a href="#">GetRawInputDeviceList</a>	Enumerates the raw input devices attached to the system.
<a href="#">GetRawPointerDeviceData</a>	Gets the raw input data from the pointer device.

TITLE	DESCRIPTION
<a href="#">GetRegisteredRawInputDevices</a>	Retrieves the information about the raw input devices for the current application.
<a href="#">GetScrollBarInfo</a>	The GetScrollBarInfo function retrieves information about the specified scroll bar.
<a href="#">GetScrollInfo</a>	The GetScrollInfo function retrieves the parameters of a scroll bar, including the minimum and maximum scrolling positions, the page size, and the position of the scroll box (thumb).
<a href="#">GetScrollPos</a>	The GetScrollPos function retrieves the current position of the scroll box (thumb) in the specified scroll bar.
<a href="#">GetScrollRange</a>	The GetScrollRange function retrieves the current minimum and maximum scroll box (thumb) positions for the specified scroll bar.
<a href="#">GetShellWindow</a>	Retrieves a handle to the Shell's desktop window.
<a href="#">GetSubMenu</a>	Retrieves a handle to the drop-down menu or submenu activated by the specified menu item.
<a href="#">GetSysColor</a>	Retrieves the current color of the specified display element.
<a href="#">GetSysColorBrush</a>	The GetSysColorBrush function retrieves a handle identifying a logical brush that corresponds to the specified color index.
<a href="#">GetSystemDpiForProcess</a>	Retrieves the system DPI associated with a given process. This is useful for avoiding compatibility issues that arise from sharing DPI-sensitive information between multiple system-aware processes with different system DPI values.
<a href="#">GetSystemMenu</a>	Enables the application to access the window menu (also known as the system menu or the control menu) for copying and modifying.
<a href="#">GetSystemMetrics</a>	Retrieves the specified system metric or system configuration setting.
<a href="#">GetSystemMetricsForDpi</a>	Retrieves the specified system metric or system configuration setting taking into account a provided DPI.
<a href="#">GetTabbedTextExtentA</a>	The GetTabbedTextExtent function computes the width and height of a character string.
<a href="#">GetTabbedTextExtentW</a>	The GetTabbedTextExtent function computes the width and height of a character string.
<a href="#">GetThreadDesktop</a>	Retrieves a handle to the desktop assigned to the specified thread.
<a href="#">GetThreadDpiAwarenessContext</a>	Gets the DPI_AWARENESS_CONTEXT for the current thread.

TITLE	DESCRIPTION
<a href="#">GetThreadDpiHostingBehavior</a>	Retrieves the DPI_HOSTING_BEHAVIOR from the current thread.
<a href="#">GetTitleBarInfo</a>	Retrieves information about the specified title bar.
<a href="#">GetTopWindow</a>	Examines the Z order of the child windows associated with the specified parent window and retrieves a handle to the child window at the top of the Z order.
<a href="#">GetTouchInputInfo</a>	Retrieves detailed information about touch inputs associated with a particular touch input handle.
<a href="#">GetUnpredictedMessagePos</a>	Gets pointer data before it has gone through touch prediction processing.
<a href="#">GetUpdatedClipboardFormats</a>	Retrieves the currently supported clipboard formats.
<a href="#">GetUpdateRect</a>	The GetUpdateRect function retrieves the coordinates of the smallest rectangle that completely encloses the update region of the specified window.
<a href="#">GetUpdateRgn</a>	The GetUpdateRgn function retrieves the update region of a window by copying it into the specified region. The coordinates of the update region are relative to the upper-left corner of the window (that is, they are client coordinates).
<a href="#">GetUserObjectInformationA</a>	Retrieves information about the specified window station or desktop object.
<a href="#">GetUserObjectInformationW</a>	Retrieves information about the specified window station or desktop object.
<a href="#">GetUserObjectSecurity</a>	Retrieves security information for the specified user object.
<a href="#">GetWindow</a>	Retrieves a handle to a window that has the specified relationship (Z-Order or owner) to the specified window.
<a href="#">GetWindowContextHelpId</a>	Retrieves the Help context identifier, if any, associated with the specified window.
<a href="#">GetWindowDC</a>	The GetWindowDC function retrieves the device context (DC) for the entire window, including title bar, menus, and scroll bars.
<a href="#">GetWindowDisplayAffinity</a>	Retrieves the current display affinity setting, from any process, for a given window.
<a href="#">GetWindowDpiAwarenessContext</a>	Returns the DPI_AWARENESS_CONTEXT associated with a window.
<a href="#">GetWindowDpiHostingBehavior</a>	Returns the DPI_HOSTING_BEHAVIOR of the specified window.

TITLE	DESCRIPTION
<a href="#">GetWindowFeedbackSetting</a>	Retrieves the feedback configuration for a window.
<a href="#">GetWindowInfo</a>	Retrieves information about the specified window.
<a href="#">GetWindowLongA</a>	Retrieves information about the specified window.
<a href="#">GetWindowLongPtrA</a>	Retrieves information about the specified window. The function also retrieves the value at a specified offset into the extra window memory.
<a href="#">GetWindowLongPtrW</a>	Retrieves information about the specified window. The function also retrieves the value at a specified offset into the extra window memory.
<a href="#">GetWindowLongW</a>	Retrieves information about the specified window.
<a href="#">GetWindowModuleFileNameA</a>	Retrieves the full path and file name of the module associated with the specified window handle.
<a href="#">GetWindowModuleFileNameW</a>	Retrieves the full path and file name of the module associated with the specified window handle.
<a href="#">GetWindowPlacement</a>	Retrieves the show state and the restored, minimized, and maximized positions of the specified window.
<a href="#">GetWindowRect</a>	Retrieves the dimensions of the bounding rectangle of the specified window. The dimensions are given in screen coordinates that are relative to the upper-left corner of the screen.
<a href="#">GetWindowRgn</a>	The <a href="#">GetWindowRgn</a> function obtains a copy of the window region of a window.
<a href="#">GetWindowRgnBox</a>	The <a href="#">GetWindowRgnBox</a> function retrieves the dimensions of the tightest bounding rectangle for the window region of a window.
<a href="#">GetWindowTextA</a>	Copies the text of the specified window's title bar (if it has one) into a buffer. If the specified window is a control, the text of the control is copied. However, <a href="#">GetWindowText</a> cannot retrieve the text of a control in another application.
<a href="#">GetWindowTextLengthA</a>	Retrieves the length, in characters, of the specified window's title bar text (if the window has a title bar).
<a href="#">GetWindowTextLengthW</a>	Retrieves the length, in characters, of the specified window's title bar text (if the window has a title bar).
<a href="#">GetWindowTextW</a>	Copies the text of the specified window's title bar (if it has one) into a buffer. If the specified window is a control, the text of the control is copied. However, <a href="#">GetWindowText</a> cannot retrieve the text of a control in another application.

TITLE	DESCRIPTION
<a href="#">GetWindowThreadProcessId</a>	Retrieves the identifier of the thread that created the specified window and, optionally, the identifier of the process that created the window.
<a href="#">GID_ROTATE_ANGLE_FROM_ARGUMENT</a>	The GID_ROTATE_ANGLE_FROM_ARGUMENT macro is used to interpret the GID_ROTATE ullArgument value when receiving the value in the WM_GESTURE structure.
<a href="#">GID_ROTATE_ANGLE_TO_ARGUMENT</a>	Converts a radian value to an argument for rotation gesture messages.
<a href="#">GrayStringA</a>	The GrayString function draws gray text at the specified location.
<a href="#">GrayStringW</a>	The GrayString function draws gray text at the specified location.
<a href="#">HAS_POINTER_CONFIDENCE_WPARAM</a>	Checks whether the specified pointer message is considered intentional rather than accidental.
<a href="#">HideCaret</a>	Removes the caret from the screen. Hiding a caret does not destroy its current shape or invalidate the insertion point.
<a href="#">HiliteMenuItem</a>	Adds or removes highlighting from an item in a menu bar.
<a href="#">InflateRect</a>	The InflateRect function increases or decreases the width and height of the specified rectangle.
<a href="#">InitializeTouchInjection</a>	Configures the touch injection context for the calling application and initializes the maximum number of simultaneous contacts that the app can inject.
<a href="#">InjectSyntheticPointerInput</a>	Simulates pointer input (pen or touch).
<a href="#">InjectTouchInput</a>	Simulates touch input.
<a href="#">InSendMessage</a>	Determines whether the current window procedure is processing a message that was sent from another thread (in the same process or a different process) by a call to the SendMessage function.
<a href="#">InSendMessageEx</a>	Determines whether the current window procedure is processing a message that was sent from another thread (in the same process or a different process).
<a href="#">InsertMenuA</a>	Inserts a new menu item into a menu, moving other items down the menu.
<a href="#">InsertMenuItemA</a>	Inserts a new menu item at the specified position in a menu.
<a href="#">InsertMenuItemW</a>	Inserts a new menu item at the specified position in a menu.
<a href="#">InsertMenuW</a>	Inserts a new menu item into a menu, moving other items down the menu.

TITLE	DESCRIPTION
<a href="#">InternalGetWindowText</a>	Copies the text of the specified window's title bar (if it has one) into a buffer.
<a href="#">IntersectRect</a>	The IntersectRect function calculates the intersection of two source rectangles and places the coordinates of the intersection rectangle into the destination rectangle.
<a href="#">InvalidateRect</a>	The InvalidateRect function adds a rectangle to the specified window's update region. The update region represents the portion of the window's client area that must be redrawn.
<a href="#">InvalidateRgn</a>	The InvalidateRgn function invalidates the client area within the specified region by adding it to the current update region of a window.
<a href="#">InvertRect</a>	The InvertRect function inverts a rectangle in a window by performing a logical NOT operation on the color values for each pixel in the rectangle's interior.
<a href="#">IS_INTRESOURCE</a>	Determines whether a value is an integer identifier for a resource.
<a href="#">IS_POINTER_CANCELED_WPARAM</a>	Checks whether the specified pointer input ended abruptly, or was invalid, indicating the interaction was not completed.
<a href="#">IS_POINTER_FIFTHBUTTON_WPARAM</a>	Checks whether the specified pointer took fifth action.
<a href="#">IS_POINTER_FIRSTBUTTON_WPARAM</a>	Checks whether the specified pointer took first action.
<a href="#">IS_POINTER_FLAG_SET_WPARAM</a>	Checks whether a pointer macro sets the specified flag.
<a href="#">IS_POINTER_FOURTHBUTTON_WPARAM</a>	Checks whether the specified pointer took fourth action.
<a href="#">IS_POINTER_INCONTACT_WPARAM</a>	Checks whether the specified pointer is in contact.
<a href="#">IS_POINTER_INRANGE_WPARAM</a>	Checks whether the specified pointer is in range.
<a href="#">IS_POINTER_NEW_WPARAM</a>	Checks whether the specified pointer is a new pointer.
<a href="#">IS_POINTER_SECONDBUTTON_WPARAM</a>	Checks whether the specified pointer took second action.
<a href="#">IS_POINTER_THIRDBUTTON_WPARAM</a>	Checks whether the specified pointer took third action.
<a href="#">IsCharAlphaA</a>	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.
<a href="#">IsCharAlphaNumericA</a>	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.

TITLE	DESCRIPTION
<a href="#">IsCharAlphaNumericW</a>	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.
<a href="#">IsCharAlphaW</a>	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.
<a href="#">IsCharLowerA</a>	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.
<a href="#">IsCharLowerW</a>	
<a href="#">IsCharUpperA</a>	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.
<a href="#">IsCharUpperW</a>	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.
<a href="#">IsChild</a>	Determines whether a window is a child window or descendant window of a specified parent window.
<a href="#">IsClipboardFormatAvailable</a>	Determines whether the clipboard contains data in the specified format.
<a href="#">IsDialogMessageA</a>	Determines whether a message is intended for the specified dialog box and, if it is, processes the message.
<a href="#">IsDialogMessageW</a>	Determines whether a message is intended for the specified dialog box and, if it is, processes the message.
<a href="#">IsDlgButtonChecked</a>	The IsDlgButtonChecked function determines whether a button control is checked or whether a three-state button control is checked, unchecked, or indeterminate.
<a href="#">IsGUIThread</a>	Determines whether the calling thread is already a GUI thread. It can also optionally convert the thread to a GUI thread.
<a href="#">IsHungAppWindow</a>	Determines whether the system considers that a specified application is not responding.
<a href="#">IsIconic</a>	Determines whether the specified window is minimized (iconic).
<a href="#">IsImmersiveProcess</a>	Determines whether the process belongs to a Windows Store app.
<a href="#">IsMenu</a>	Determines whether a handle is a menu handle.



TITLE	DESCRIPTION
<a href="#">IsMouseInPointerEnabled</a>	Indicates whether EnableMouseInPointer is set for the mouse to act as a pointer input device and send WM_POINTER messages.
<a href="#">IsProcessDPIAware</a>	IsProcessDPIAware may be altered or unavailable. Instead, use GetProcessDPIAwareness.
<a href="#">IsRectEmpty</a>	The IsRectEmpty function determines whether the specified rectangle is empty.
<a href="#">IsTouchWindow</a>	Checks whether a specified window is touch-capable and, optionally, retrieves the modifier flags set for the window's touch capability.
<a href="#">IsValidDpiAwarenessContext</a>	Determines if a specified DPI_AWARENESS_CONTEXT is valid and supported by the current system.
<a href="#">IsWindow</a>	Determines whether the specified window handle identifies an existing window.
<a href="#">IsWindowEnabled</a>	Determines whether the specified window is enabled for mouse and keyboard input.
<a href="#">IsWindowUnicode</a>	Determines whether the specified window is a native Unicode window.
<a href="#">IsWindowVisible</a>	Determines the visibility state of the specified window.
<a href="#">IsWinEventHookInstalled</a>	Determines whether there is an installed WinEvent hook that might be notified of a specified event.
<a href="#">IsWow64Message</a>	Determines whether the last message read from the current thread's queue originated from a WOW64 process.
<a href="#">IsZoomed</a>	Determines whether a window is maximized.
<a href="#">keybd_event</a>	Synthesizes a keystroke.
<a href="#">KillTimer</a>	Destroys the specified timer.
<a href="#">LoadAcceleratorsA</a>	Loads the specified accelerator table.
<a href="#">LoadAcceleratorsW</a>	Loads the specified accelerator table.
<a href="#">LoadBitmapA</a>	The LoadBitmap function loads the specified bitmap resource from a module's executable file.
<a href="#">LoadBitmapW</a>	The LoadBitmap function loads the specified bitmap resource from a module's executable file.
<a href="#">LoadCursorA</a>	Loads the specified cursor resource from the executable (.EXE) file associated with an application instance.

TITLE	DESCRIPTION
<a href="#">LoadCursorFromFileA</a>	Creates a cursor based on data contained in a file.
<a href="#">LoadCursorFromFileW</a>	Creates a cursor based on data contained in a file.
<a href="#">LoadCursorW</a>	Loads the specified cursor resource from the executable (.EXE) file associated with an application instance.
<a href="#">LoadIconA</a>	Loads the specified icon resource from the executable (.exe) file associated with an application instance.
<a href="#">LoadIconW</a>	Loads the specified icon resource from the executable (.exe) file associated with an application instance.
<a href="#">LoadImageA</a>	Loads an icon, cursor, animated cursor, or bitmap.
<a href="#">LoadImageW</a>	Loads an icon, cursor, animated cursor, or bitmap.
<a href="#">LoadKeyboardLayoutA</a>	Loads a new input locale identifier (formerly called the keyboard layout) into the system.
<a href="#">LoadKeyboardLayoutW</a>	Loads a new input locale identifier (formerly called the keyboard layout) into the system.
<a href="#">LoadMenuA</a>	Loads the specified menu resource from the executable (.exe) file associated with an application instance.
<a href="#">LoadMenuIndirectA</a>	Loads the specified menu template in memory.
<a href="#">LoadMenuIndirectW</a>	Loads the specified menu template in memory.
<a href="#">LoadMenuW</a>	Loads the specified menu resource from the executable (.exe) file associated with an application instance.
<a href="#">LoadStringA</a>	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.
<a href="#">LoadStringW</a>	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.
<a href="#">LockSetForegroundWindow</a>	The foreground process can call the LockSetForegroundWindow function to disable calls to the SetForegroundWindow function.
<a href="#">LockWindowUpdate</a>	The LockWindowUpdate function disables or enables drawing in the specified window. Only one window can be locked at a time.
<a href="#">LockWorkStation</a>	Locks the workstation's display.
<a href="#">LogicalToPhysicalPoint</a>	Converts the logical coordinates of a point in a window to physical coordinates.

TITLE	DESCRIPTION
<a href="#">LogicalToPhysicalPointForPerMonitorDPI</a>	Converts a point in a window from logical coordinates into physical coordinates, regardless of the dots per inch (dpi) awareness of the caller.
<a href="#">LookupIconIdFromDirectory</a>	Searches through icon or cursor data for the icon or cursor that best fits the current display device.
<a href="#">LookupIconIdFromDirectoryEx</a>	Searches through icon or cursor data for the icon or cursor that best fits the current display device.
<a href="#">MAKEINTRESOURCEA</a>	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.
<a href="#">MAKEINTRESOURCEW</a>	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.
<a href="#">MAKELPARAM</a>	Creates a value for use as an lParam parameter in a message. The macro concatenates the specified values.
<a href="#">MAKELRESULT</a>	Creates a value for use as a return value from a window procedure. The macro concatenates the specified values.
<a href="#">MAKEWPARAM</a>	Creates a value for use as a wParam parameter in a message. The macro concatenates the specified values.
<a href="#">MapDialogRect</a>	Converts the specified dialog box units to screen units (pixels).
<a href="#">MapVirtualKeyA</a>	Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code.
<a href="#">MapVirtualKeyExA</a>	Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code. The function translates the codes using the input language and an input locale identifier.
<a href="#">MapVirtualKeyExW</a>	Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code. The function translates the codes using the input language and an input locale identifier.
<a href="#">MapVirtualKeyW</a>	Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code.
<a href="#">MapWindowPoints</a>	The MapWindowPoints function converts (maps) a set of points from a coordinate space relative to one window to a coordinate space relative to another window.
<a href="#">MenuItemFromPoint</a>	Determines which menu item, if any, is at the specified location.

TITLE	DESCRIPTION
<a href="#">MessageBeep</a>	Plays a waveform sound. The waveform sound for each sound type is identified by an entry in the registry.
<a href="#">MessageBox</a>	Displays a modal dialog box that contains a system icon, a set of buttons, and a brief application-specific message, such as status or error information. The message box returns an integer value that indicates which button the user clicked.
<a href="#">MessageBoxA</a>	Displays a modal dialog box that contains a system icon, a set of buttons, and a brief application-specific message, such as status or error information. The message box returns an integer value that indicates which button the user clicked.
<a href="#">MessageBoxExA</a>	Creates, displays, and operates a message box.
<a href="#">MessageBoxExW</a>	Creates, displays, and operates a message box.
<a href="#">MessageBoxIndirectA</a>	Creates, displays, and operates a message box. The message box contains application-defined message text and title, any icon, and any combination of predefined push buttons.
<a href="#">MessageBoxIndirectW</a>	Creates, displays, and operates a message box. The message box contains application-defined message text and title, any icon, and any combination of predefined push buttons.
<a href="#">MessageBoxW</a>	Displays a modal dialog box that contains a system icon, a set of buttons, and a brief application-specific message, such as status or error information. The message box returns an integer value that indicates which button the user clicked.
<a href="#">ModifyMenuA</a>	Changes an existing menu item.
<a href="#">ModifyMenuW</a>	Changes an existing menu item.
<a href="#">MonitorFromPoint</a>	The MonitorFromPoint function retrieves a handle to the display monitor that contains a specified point.
<a href="#">MonitorFromRect</a>	The MonitorFromRect function retrieves a handle to the display monitor that has the largest area of intersection with a specified rectangle.
<a href="#">MonitorFromWindow</a>	The MonitorFromWindow function retrieves a handle to the display monitor that has the largest area of intersection with the bounding rectangle of a specified window.
<a href="#">mouse_event</a>	The mouse_event function synthesizes mouse motion and button clicks.
<a href="#">MoveWindow</a>	Changes the position and dimensions of the specified window.
<a href="#">MsgWaitForMultipleObjects</a>	Waits until one or all of the specified objects are in the signaled state or the time-out interval elapses. The objects can include input event objects.

TITLE	DESCRIPTION
<a href="#">MsgWaitForMultipleObjectsEx</a>	Waits until one or all of the specified objects are in the signaled state, an I/O completion routine or asynchronous procedure call (APC) is queued to the thread, or the time-out interval elapses. The array of objects can include input event objects.
<a href="#">NEXTRAWINPUTBLOCK</a>	Retrieves the location of the next structure in an array of RAWINPUT structures.
<a href="#">NotifyWinEvent</a>	Signals the system that a predefined event occurred. If any client applications have registered a hook function for the event, the system calls the client's hook function.
<a href="#">OemKeyScan</a>	Maps OEMASCII codes 0 through 0xFF into the OEM scan codes and shift states. The function provides information that allows a program to send OEM text to another program by simulating keyboard input.
<a href="#">OemToCharA</a>	Translates a string from the OEM-defined character set into either an ANSI or a wide-character string. Warning Do not use.
<a href="#">OemToCharBuffA</a>	Translates a specified number of characters in a string from the OEM-defined character set into either an ANSI or a wide-character string.
<a href="#">OemToCharBuffW</a>	Translates a specified number of characters in a string from the OEM-defined character set into either an ANSI or a wide-character string.
<a href="#">OemToCharW</a>	Translates a string from the OEM-defined character set into either an ANSI or a wide-character string. Warning Do not use.
<a href="#">OffsetRect</a>	The OffsetRect function moves the specified rectangle by the specified offsets.
<a href="#">OpenClipboard</a>	Opens the clipboard for examination and prevents other applications from modifying the clipboard content.
<a href="#">OpenDesktopA</a>	Opens the specified desktop object.
<a href="#">OpenDesktopW</a>	Opens the specified desktop object.
<a href="#">OpenIcon</a>	Restores a minimized (iconic) window to its previous size and position; it then activates the window.
<a href="#">OpenInputDesktop</a>	Opens the desktop that receives user input.
<a href="#">OpenWindowStationA</a>	Opens the specified window station.
<a href="#">OpenWindowStationW</a>	Opens the specified window station.

TITLE	DESCRIPTION
<a href="#">PackTouchHitTestingProximityEvaluation</a>	Returns the proximity evaluation score and the adjusted touch-point coordinates as a packed value for the WM_TOUCHHITTESTING callback.
<a href="#">PaintDesktop</a>	The PaintDesktop function fills the clipping region in the specified device context with the desktop pattern or wallpaper. The function is provided primarily for shell desktops.
<a href="#">PeekMessageA</a>	Dispatches incoming sent messages, checks the thread message queue for a posted message, and retrieves the message (if any exist).
<a href="#">PeekMessageW</a>	Dispatches incoming sent messages, checks the thread message queue for a posted message, and retrieves the message (if any exist).
<a href="#">PhysicalToLogicalPoint</a>	Converts the physical coordinates of a point in a window to logical coordinates.
<a href="#">PhysicalToLogicalPointForPerMonitorDPI</a>	Converts a point in a window from physical coordinates into logical coordinates, regardless of the dots per inch (dpi) awareness of the caller.
<a href="#">POINTSTOPOINT</a>	The POINTSTOPOINT macro copies the contents of a POINTS structure into a POINT structure.
<a href="#">POINTTOPOINTS</a>	The POINTTOPOINTS macro converts a POINT structure to a POINTS structure.
<a href="#">PostMessageA</a>	Places (posts) a message in the message queue associated with the thread that created the specified window and returns without waiting for the thread to process the message.
<a href="#">PostMessageW</a>	Places (posts) a message in the message queue associated with the thread that created the specified window and returns without waiting for the thread to process the message.
<a href="#">PostQuitMessage</a>	Indicates to the system that a thread has made a request to terminate (quit). It is typically used in response to a WM_DESTROY message.
<a href="#">PostThreadMessageA</a>	Posts a message to the message queue of the specified thread. It returns without waiting for the thread to process the message.
<a href="#">PostThreadMessageW</a>	Posts a message to the message queue of the specified thread. It returns without waiting for the thread to process the message.
<a href="#">PrintWindow</a>	The PrintWindow function copies a visual window into the specified device context (DC), typically a printer DC.

TITLE	DESCRIPTION
<a href="#">PrivateExtractIconsA</a>	Creates an array of handles to icons that are extracted from a specified file.
<a href="#">PrivateExtractIconsW</a>	Creates an array of handles to icons that are extracted from a specified file.
<a href="#">PtInRect</a>	The PtInRect function determines whether the specified point lies within the specified rectangle.
<a href="#">QueryDisplayConfig</a>	The QueryDisplayConfig function retrieves information about all possible display paths for all display devices, or views, in the current setting.
<a href="#">RealChildWindowFromPoint</a>	Retrieves a handle to the child window at the specified point. The search is restricted to immediate child windows; grandchildren and deeper descendant windows are not searched.
<a href="#">RealGetWindowClassW</a>	Retrieves a string that specifies the window type.
<a href="#">RedrawWindow</a>	The RedrawWindow function updates the specified rectangle or region in a window's client area.
<a href="#">RegisterClassA</a>	Registers a window class for subsequent use in calls to the CreateWindow or CreateWindowEx function.
<a href="#">RegisterClassExA</a>	Registers a window class for subsequent use in calls to the CreateWindow or CreateWindowEx function.
<a href="#">RegisterClassExW</a>	Registers a window class for subsequent use in calls to the CreateWindow or CreateWindowEx function.
<a href="#">RegisterClassW</a>	Registers a window class for subsequent use in calls to the CreateWindow or CreateWindowEx function.
<a href="#">RegisterClipboardFormatA</a>	Registers a new clipboard format. This format can then be used as a valid clipboard format.
<a href="#">RegisterClipboardFormatW</a>	Registers a new clipboard format. This format can then be used as a valid clipboard format.
<a href="#">RegisterDeviceNotificationA</a>	Registers the device or type of device for which a window will receive notifications.
<a href="#">RegisterDeviceNotificationW</a>	Registers the device or type of device for which a window will receive notifications.
<a href="#">RegisterHotKey</a>	Defines a system-wide hot key.
<a href="#">RegisterPointerDeviceNotifications</a>	Registers a window to process the WM_POINTERDEVICECHANGE, WM_POINTERDEVICEINRANGE, and WM_POINTERDEVICEOUTOFRANGE pointer device notifications.

TITLE	DESCRIPTION
<a href="#">RegisterPointerInputTarget</a>	Allows the caller to register a target window to which all pointer input of the specified type is redirected.
<a href="#">RegisterPointerInputTargetEx</a>	RegisterPointerInputTargetEx may be altered or unavailable. Instead, use RegisterPointerInputTarget.
<a href="#">RegisterPowerSettingNotification</a>	Registers the application to receive power setting notifications for the specific power setting event.
<a href="#">RegisterRawInputDevices</a>	Registers the devices that supply the raw input data.
<a href="#">RegisterShellHookWindow</a>	Registers a specified Shell window to receive certain messages for events or notifications that are useful to Shell applications.
<a href="#">RegisterSuspendResumeNotification</a>	Registers to receive notification when the system is suspended or resumed. Similar to PowerRegisterSuspendResumeNotification, but operates in user mode and can take a window handle.
<a href="#">RegisterTouchHitTestingWindow</a>	Registers a window to process the WM_TOUCHHITTESTING notification.
<a href="#">RegisterTouchWindow</a>	Registers a window as being touch-capable.
<a href="#">RegisterWindowMessageA</a>	Defines a new window message that is guaranteed to be unique throughout the system. The message value can be used when sending or posting messages.
<a href="#">RegisterWindowMessageW</a>	Defines a new window message that is guaranteed to be unique throughout the system. The message value can be used when sending or posting messages.
<a href="#">ReleaseCapture</a>	Releases the mouse capture from a window in the current thread and restores normal mouse input processing.
<a href="#">ReleaseDC</a>	The ReleaseDC function releases a device context (DC), freeing it for use by other applications. The effect of the ReleaseDC function depends on the type of DC. It frees only common and window DCs. It has no effect on class or private DCs.
<a href="#">RemoveClipboardFormatListener</a>	Removes the given window from the system-maintained clipboard format listener list.
<a href="#">RemoveMenu</a>	Deletes a menu item or detaches a submenu from the specified menu.
<a href="#">RemovePropA</a>	Removes an entry from the property list of the specified window. The specified character string identifies the entry to be removed.
<a href="#">RemovePropW</a>	Removes an entry from the property list of the specified window. The specified character string identifies the entry to be removed.



TITLE	DESCRIPTION
<a href="#">ReplyMessage</a>	Replies to a message sent from another thread by the SendMessage function.
<a href="#">ScreenToClient</a>	The ScreenToClient function converts the screen coordinates of a specified point on the screen to client-area coordinates.
<a href="#">ScrollDC</a>	The ScrollDC function scrolls a rectangle of bits horizontally and vertically.
<a href="#">ScrollWindow</a>	The ScrollWindow function scrolls the contents of the specified window's client area.
<a href="#">ScrollWindowEx</a>	The ScrollWindowEx function scrolls the contents of the specified window's client area.
<a href="#">SendDlgItemMessageA</a>	Sends a message to the specified control in a dialog box.
<a href="#">SendDlgItemMessageW</a>	Sends a message to the specified control in a dialog box.
<a href="#">SendInput</a>	Synthesizes keystrokes, mouse motions, and button clicks.
<a href="#">SendMessage</a>	Sends the specified message to a window or windows. The SendMessage function calls the window procedure for the specified window and does not return until the window procedure has processed the message.
<a href="#">SendMessageA</a>	Sends the specified message to a window or windows. The SendMessage function calls the window procedure for the specified window and does not return until the window procedure has processed the message.
<a href="#">SendMessageCallbackA</a>	Sends the specified message to a window or windows.
<a href="#">SendMessageCallbackW</a>	Sends the specified message to a window or windows.
<a href="#">SendMessageTimeoutA</a>	Sends the specified message to one or more windows.
<a href="#">SendMessageTimeoutW</a>	Sends the specified message to one or more windows.
<a href="#">SendMessageW</a>	Sends the specified message to a window or windows. The SendMessage function calls the window procedure for the specified window and does not return until the window procedure has processed the message.
<a href="#">SendNotifyMessageA</a>	Sends the specified message to a window or windows.
<a href="#">SendNotifyMessageW</a>	Sends the specified message to a window or windows.

TITLE	DESCRIPTION
<a href="#">SetActiveWindow</a>	Activates a window. The window must be attached to the calling thread's message queue.
<a href="#">SetCapture</a>	Sets the mouse capture to the specified window belonging to the current thread.
<a href="#">SetCaretBlinkTime</a>	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.
<a href="#">SetCaretPos</a>	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.
<a href="#">SetClassLongA</a>	Replaces the specified 32-bit (long) value at the specified offset into the extra class memory or the WNDCLASSEX structure for the class to which the specified window belongs.
<a href="#">SetClassLongPtrA</a>	Replaces the specified value at the specified offset in the extra class memory or the WNDCLASSEX structure for the class to which the specified window belongs.
<a href="#">SetClassLongPtrW</a>	Replaces the specified value at the specified offset in the extra class memory or the WNDCLASSEX structure for the class to which the specified window belongs.
<a href="#">SetClassLongW</a>	Replaces the specified 32-bit (long) value at the specified offset into the extra class memory or the WNDCLASSEX structure for the class to which the specified window belongs.
<a href="#">SetClassWord</a>	Replaces the 16-bit (WORD) value at the specified offset into the extra class memory for the window class to which the specified window belongs.
<a href="#">SetClipboardData</a>	Places data on the clipboard in a specified clipboard format.
<a href="#">SetClipboardViewer</a>	Adds the specified window to the chain of clipboard viewers. Clipboard viewer windows receive a WM_DRAWCLIPBOARD message whenever the content of the clipboard changes. This function is used for backward compatibility with earlier versions of Windows.
<a href="#">SetCoalescableTimer</a>	Creates a timer with the specified time-out value and coalescing tolerance delay.
<a href="#">SetCursor</a>	Sets the cursor shape.
<a href="#">SetCursorPos</a>	Moves the cursor to the specified screen coordinates.
<a href="#">SetDialogControlDpiChangeBehavior</a>	Overrides the default per-monitor DPI scaling behavior of a child window in a dialog.

TITLE	DESCRIPTION
<a href="#">SetDialogDpiChangeBehavior</a>	Dialogs in Per-Monitor v2 contexts are automatically DPI scaled. This method lets you customize their DPI change behavior.
<a href="#">SetDisplayAutoRotationPreferences</a>	Sets the screen auto-rotation preferences for the current process.
<a href="#">SetDisplayConfig</a>	The SetDisplayConfig function modifies the display topology, source, and target modes by exclusively enabling the specified paths in the current session.
<a href="#">SetDlgItemInt</a>	Sets the text of a control in a dialog box to the string representation of a specified integer value.
<a href="#">SetDlgItemTextA</a>	Sets the title or text of a control in a dialog box.
<a href="#">SetDlgItemTextW</a>	Sets the title or text of a control in a dialog box.
<a href="#">SetDoubleClickTime</a>	Sets the double-click time for the mouse.
<a href="#">SetFocus</a>	Sets the keyboard focus to the specified window. The window must be attached to the calling thread's message queue.
<a href="#">SetForegroundWindow</a>	Brings the thread that created the specified window into the foreground and activates the window.
<a href="#">SetGestureConfig</a>	Configures the messages that are sent from a window for Windows Touch gestures.
<a href="#">SetKeyboardState</a>	Copies an array of keyboard key states into the calling thread's keyboard input-state table. This is the same table accessed by the GetKeyboardState and GetKeyState functions. Changes made to this table do not affect keyboard input to any other thread.
<a href="#">SetLastErrorEx</a>	Sets the last-error code.
<a href="#">SetLayeredWindowAttributes</a>	Sets the opacity and transparency color key of a layered window.
<a href="#">SetMenu</a>	Assigns a new menu to the specified window.
<a href="#">SetMenuContextHelpId</a>	Associates a Help context identifier with a menu.
<a href="#">SetMenuDefaultItem</a>	Sets the default menu item for the specified menu.
<a href="#">SetMenuInfo</a>	Sets information for a specified menu.
<a href="#">SetMenuItemBitmaps</a>	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.

TITLE	DESCRIPTION
<a href="#">SetMenuItemInfoA</a>	Changes information about a menu item.
<a href="#">SetMenuItemInfoW</a>	Changes information about a menu item.
<a href="#">SetMessageExtraInfo</a>	Sets the extra message information for the current thread.
<a href="#">SetParent</a>	Changes the parent window of the specified child window.
<a href="#">SetPhysicalCursorPos</a>	Sets the position of the cursor in physical coordinates.
<a href="#">SetProcessDefaultLayout</a>	Changes the default layout when windows are created with no parent or owner only for the currently running process.
<a href="#">SetProcessDPIAware</a>	SetProcessDPIAware may be altered or unavailable. Instead, use SetProcessDPIAwareness.
<a href="#">SetProcessDpiAwarenessContext</a>	Sets the current process to a specified dots per inch (dpi) awareness context. The DPI awareness contexts are from the DPI_AWARENESS_CONTEXT value.
<a href="#">SetProcessRestrictionExemption</a>	Exempts the calling process from restrictions preventing desktop processes from interacting with the Windows Store app environment. This function is used by development and debugging tools.
<a href="#">SetProcessWindowStation</a>	Assigns the specified window station to the calling process.
<a href="#">SetPropA</a>	Adds a new entry or changes an existing entry in the property list of the specified window.
<a href="#">SetPropW</a>	Adds a new entry or changes an existing entry in the property list of the specified window.
<a href="#">SetRect</a>	The SetRect function sets the coordinates of the specified rectangle. This is equivalent to assigning the left, top, right, and bottom arguments to the appropriate members of the RECT structure.
<a href="#">SetRectEmpty</a>	The SetRectEmpty function creates an empty rectangle in which all coordinates are set to zero.
<a href="#">SetScrollInfo</a>	The SetScrollInfo function sets the parameters of a scroll bar, including the minimum and maximum scrolling positions, the page size, and the position of the scroll box (thumb). The function also redraws the scroll bar, if requested.
<a href="#">SetScrollPos</a>	The SetScrollPos function sets the position of the scroll box (thumb) in the specified scroll bar and, if requested, redraws the scroll bar to reflect the new position of the scroll box.
<a href="#">SetScrollRange</a>	The SetScrollRange function sets the minimum and maximum scroll box positions for the specified scroll bar.
<a href="#">SetSysColors</a>	Sets the colors for the specified display elements.

TITLE	DESCRIPTION
<a href="#">SetSystemCursor</a>	Enables an application to customize the system cursors. It replaces the contents of the system cursor specified by the <code>id</code> parameter with the contents of the cursor specified by the <code>hcur</code> parameter and then destroys <code>hcur</code> .
<a href="#">SetThreadDesktop</a>	Assigns the specified desktop to the calling thread. All subsequent operations on the desktop use the access rights granted to the desktop.
<a href="#">SetThreadDpiAwarenessContext</a>	Set the DPI awareness for the current thread to the provided value.
<a href="#">SetThreadDpiHostingBehavior</a>	Sets the thread's <code>DPI_HOSTING_BEHAVIOR</code> . This behavior allows windows created in the thread to host child windows with a different <code>DPI_AWARENESS_CONTEXT</code> .
<a href="#">SetTimer</a>	Creates a timer with the specified time-out value.
<a href="#">SetUserObjectInformationA</a>	Sets information about the specified window station or desktop object.
<a href="#">SetUserObjectInformationW</a>	Sets information about the specified window station or desktop object.
<a href="#">SetUserObjectSecurity</a>	Sets the security of a user object. This can be, for example, a window or a DDE conversation.
<a href="#">SetWindowContextHelpId</a>	Associates a Help context identifier with the specified window.
<a href="#">SetWindowDisplayAffinity</a>	Stores the display affinity setting in kernel mode on the <code>hWnd</code> associated with the window.
<a href="#">SetWindowFeedbackSetting</a>	Sets the feedback configuration for a window.
<a href="#">SetWindowLongA</a>	Changes an attribute of the specified window. The function also sets the 32-bit (long) value at the specified offset into the extra window memory.
<a href="#">SetWindowLongPtrA</a>	Changes an attribute of the specified window.
<a href="#">SetWindowLongPtrW</a>	Changes an attribute of the specified window.
<a href="#">SetWindowLongW</a>	Changes an attribute of the specified window. The function also sets the 32-bit (long) value at the specified offset into the extra window memory.
<a href="#">SetWindowPlacement</a>	Sets the show state and the restored, minimized, and maximized positions of the specified window.

TITLE	DESCRIPTION
<a href="#">SetWindowPos</a>	Changes the size, position, and Z order of a child, pop-up, or top-level window. These windows are ordered according to their appearance on the screen. The topmost window receives the highest rank and is the first window in the Z order.
<a href="#">SetWindowRgn</a>	The SetWindowRgn function sets the window region of a window.
<a href="#">SetWindowsHookExA</a>	Installs an application-defined hook procedure into a hook chain.
<a href="#">SetWindowsHookExW</a>	Installs an application-defined hook procedure into a hook chain.
<a href="#">SetWindowTextA</a>	Changes the text of the specified window's title bar (if it has one). If the specified window is a control, the text of the control is changed. However, SetWindowText cannot change the text of a control in another application.
<a href="#">SetWindowTextW</a>	Changes the text of the specified window's title bar (if it has one). If the specified window is a control, the text of the control is changed. However, SetWindowText cannot change the text of a control in another application.
<a href="#">SetWinEventHook</a>	Sets an event hook function for a range of events.
<a href="#">ShowCaret</a>	Makes the caret visible on the screen at the caret's current position. When the caret becomes visible, it begins flashing automatically.
<a href="#">ShowCursor</a>	Displays or hides the cursor.
<a href="#">ShowOwnedPopups</a>	Shows or hides all pop-up windows owned by the specified window.
<a href="#">ShowScrollBar</a>	The ShowScrollBar function shows or hides the specified scroll bar.
<a href="#">ShowWindow</a>	Sets the specified window's show state.
<a href="#">ShowWindowAsync</a>	Sets the show state of a window without waiting for the operation to complete.
<a href="#">ShutdownBlockReasonCreate</a>	Indicates that the system cannot be shut down and sets a reason string to be displayed to the user if system shutdown is initiated.
<a href="#">ShutdownBlockReasonDestroy</a>	Indicates that the system can be shut down and frees the reason string.
<a href="#">ShutdownBlockReasonQuery</a>	Retrieves the reason string set by the ShutdownBlockReasonCreate function.

TITLE	DESCRIPTION
<a href="#">SkipPointerFrameMessages</a>	Determines which pointer input frame generated the most recently retrieved message for the specified pointer and discards any queued (unretrieved) pointer input messages generated from the same pointer input frame.
<a href="#">SoundSentry</a>	Triggers a visual signal to indicate that a sound is playing.
<a href="#">SubtractRect</a>	The SubtractRect function determines the coordinates of a rectangle formed by subtracting one rectangle from another.
<a href="#">SwapMouseButton</a>	Reverses or restores the meaning of the left and right mouse buttons.
<a href="#">SwitchDesktop</a>	Makes the specified desktop visible and activates it. This enables the desktop to receive input from the user.
<a href="#">SwitchToThisWindow</a>	Switches focus to the specified window and brings it to the foreground.
<a href="#">SystemParametersInfoA</a>	Retrieves or sets the value of one of the system-wide parameters.
<a href="#">SystemParametersInfoForDpi</a>	Retrieves the value of one of the system-wide parameters, taking into account the provided DPI value.
<a href="#">SystemParametersInfoW</a>	Retrieves or sets the value of one of the system-wide parameters.
<a href="#">TabbedTextOutA</a>	The TabbedTextOut function writes a character string at a specified location, expanding tabs to the values specified in an array of tab-stop positions. Text is written in the currently selected font, background color, and text color.
<a href="#">TabbedTextOutW</a>	The TabbedTextOut function writes a character string at a specified location, expanding tabs to the values specified in an array of tab-stop positions. Text is written in the currently selected font, background color, and text color.
<a href="#">TileWindows</a>	Tiles the specified child windows of the specified parent window.
<a href="#">ToAscii</a>	Translates the specified virtual-key code and keyboard state to the corresponding character or characters.
<a href="#">ToAsciiEx</a>	Translates the specified virtual-key code and keyboard state to the corresponding character or characters. The function translates the code using the input language and physical keyboard layout identified by the input locale identifier.
<a href="#">TOUCH_COORD_TO_PIXEL</a>	Converts touch coordinates to pixels.
<a href="#">ToUnicode</a>	Translates the specified virtual-key code and keyboard state to the corresponding Unicode character or characters.

TITLE	DESCRIPTION
<a href="#">ToUnicodeEx</a>	Translates the specified virtual-key code and keyboard state to the corresponding Unicode character or characters.
<a href="#">TrackMouseEvent</a>	Posts messages when the mouse pointer leaves a window or hovers over a window for a specified amount of time.
<a href="#">TrackPopupMenu</a>	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.
<a href="#">TrackPopupMenuEx</a>	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.
<a href="#">TranslateAcceleratorA</a>	Processes accelerator keys for menu commands.
<a href="#">TranslateAcceleratorW</a>	Processes accelerator keys for menu commands.
<a href="#">TranslateMDISysAccel</a>	Processes accelerator keystrokes for window menu commands of the multiple-document interface (MDI) child windows associated with the specified MDI client window.
<a href="#">TranslateMessage</a>	Translates virtual-key messages into character messages. The character messages are posted to the calling thread's message queue, to be read the next time the thread calls the GetMessage or PeekMessage function.
<a href="#">UnhookWindowsHookEx</a>	Removes a hook procedure installed in a hook chain by the SetWindowsHookEx function.
<a href="#">UnhookWinEvent</a>	Removes an event hook function created by a previous call to SetWinEventHook.
<a href="#">UnionRect</a>	The UnionRect function creates the union of two rectangles. The union is the smallest rectangle that contains both source rectangles.
<a href="#">UnloadKeyboardLayout</a>	Unloads an input locale identifier (formerly called a keyboard layout).
<a href="#">UnregisterClassA</a>	Unregisters a window class, freeing the memory required for the class.
<a href="#">UnregisterClassW</a>	Unregisters a window class, freeing the memory required for the class.
<a href="#">UnregisterDeviceNotification</a>	Closes the specified device notification handle.
<a href="#">UnregisterHotKey</a>	Frees a hot key previously registered by the calling thread.
<a href="#">UnregisterPointerInputTarget</a>	Allows the caller to unregister a target window to which all pointer input of the specified type is redirected.



TITLE	DESCRIPTION
<a href="#">UnregisterPointerInputTargetEx</a>	UnregisterPointerInputTargetEx may be altered or unavailable. Instead, use UnregisterPointerInputTarget.
<a href="#">UnregisterPowerSettingNotification</a>	Unregisters the power setting notification.
<a href="#">UnregisterSuspendResumeNotification</a>	Cancels a registration to receive notification when the system is suspended or resumed. Similar to PowerUnregisterSuspendResumeNotification but operates in user mode.
<a href="#">UnregisterTouchWindow</a>	Registers a window as no longer being touch-capable.
<a href="#">UpdateLayeredWindow</a>	Updates the position, size, shape, content, and translucency of a layered window.
<a href="#">UpdateWindow</a>	The UpdateWindow function updates the client area of the specified window by sending a WM_PAINT message to the window if the window's update region is not empty.
<a href="#">UserHandleGrantAccess</a>	Grants or denies access to a handle to a User object to a job that has a user-interface restriction.
<a href="#">ValidateRect</a>	The ValidateRect function validates the client area within a rectangle by removing the rectangle from the update region of the specified window.
<a href="#">ValidateRgn</a>	The ValidateRgn function validates the client area within a region by removing the region from the current update region of the specified window.
<a href="#">VkKeyScanA</a>	Translates a character to the corresponding virtual-key code and shift state for the current keyboard.
<a href="#">VkKeyScanExA</a>	Translates a character to the corresponding virtual-key code and shift state. The function translates the character using the input language and physical keyboard layout identified by the input locale identifier.
<a href="#">VkKeyScanExW</a>	Translates a character to the corresponding virtual-key code and shift state. The function translates the character using the input language and physical keyboard layout identified by the input locale identifier.
<a href="#">VkKeyScanW</a>	Translates a character to the corresponding virtual-key code and shift state for the current keyboard.
<a href="#">WaitForInputIdle</a>	Waits until the specified process has finished processing its initial input and is waiting for user input with no input pending, or until the time-out interval has elapsed.
<a href="#">WaitMessage</a>	Yields control to other threads when a thread has no other messages in its message queue. The WaitMessage function suspends the thread and does not return until a new message is placed in the thread's message queue.

TITLE	DESCRIPTION
<a href="#">WindowFromDC</a>	The WindowFromDC function returns a handle to the window associated with the specified display device context (DC). Output functions that use the specified device context draw into this window.
<a href="#">WindowFromPhysicalPoint</a>	Retrieves a handle to the window that contains the specified physical point.
<a href="#">WindowFromPoint</a>	Retrieves a handle to the window that contains the specified point.
<a href="#">WinHelpA</a>	Launches Windows Help (Winhelp.exe) and passes additional data that indicates the nature of the help requested by the application.
<a href="#">WinHelpW</a>	Launches Windows Help (Winhelp.exe) and passes additional data that indicates the nature of the help requested by the application.
<a href="#">wsprintfA</a>	Writes formatted data to the specified buffer.
<a href="#">wsprintfW</a>	Writes formatted data to the specified buffer.
<a href="#">wvsprintfA</a>	Writes formatted data to the specified buffer using a pointer to a list of arguments.
<a href="#">wvsprintfW</a>	Writes formatted data to the specified buffer using a pointer to a list of arguments.

## Callback functions

TITLE	DESCRIPTION
<a href="#">DLGPROC</a>	Application-defined callback function used with the CreateDialog and DialogBox families of functions.
<a href="#">DRAWSTATEPROC</a>	The DrawStateProc function is an application-defined callback function that renders a complex image for the DrawState function.
<a href="#">EDITWORDBREAKPROCA</a>	An application-defined callback function used with the EM_SETWORDBREAKPROC message.
<a href="#">EDITWORDBREAKPROCW</a>	An application-defined callback function used with the EM_SETWORDBREAKPROC message.
<a href="#">GRAYSTRINGPROC</a>	The OutputProc function is an application-defined callback function used with the GrayString function.

TITLE	DESCRIPTION
<a href="#">HOOKPROC</a>	An application-defined or library-defined callback function used with the SetWindowsHookEx function. The system calls this function after the SendMessage function is called. The hook procedure can examine the message; it cannot modify it.
<a href="#">MONITORENUMPROC</a>	A MonitorEnumProc function is an application-defined callback function that is called by the EnumDisplayMonitors function.
<a href="#">PROPENUMPROCA</a>	An application-defined callback function used with the EnumProps function.
<a href="#">PROPENUMPROCEXA</a>	Application-defined callback function used with the EnumPropsEx function.
<a href="#">PROPENUMPROCEXW</a>	Application-defined callback function used with the EnumPropsEx function.
<a href="#">PROPENUMPROCW</a>	An application-defined callback function used with the EnumProps function.
<a href="#">SENDASYNCPROC</a>	An application-defined callback function used with the SendMessageCallback function.
<a href="#">TIMERPROC</a>	An application-defined callback function that processes WM_TIMER messages. The TIMERPROC type defines a pointer to this callback function. TimerProc is a placeholder for the application-defined function name.
<a href="#">WINEVENTPROC</a>	An application-defined callback (or hook) function that the system calls in response to events generated by an accessible object.

## Structures

TITLE	DESCRIPTION
<a href="#">ACCEL</a>	Defines an accelerator key used in an accelerator table.
<a href="#">ACCESSTIMEOUT</a>	Contains information about the time-out period associated with the accessibility features.
<a href="#">ALTTABINFO</a>	Contains status information for the application-switching (ALT+TAB) window.
<a href="#">ANIMATIONINFO</a>	Describes the animation effects associated with user actions.
<a href="#">AUDIODESCRIPTION</a>	Contains information associated with audio descriptions. This structure is used with the SystemParametersInfo function when the SPI_GETAUDIODESCRIPTION or SPI_SETAUDIODESCRIPTION action value is specified.

TITLE	DESCRIPTION
BSMINFO	Contains information about a window that denied a request from BroadcastSystemMessageEx.
CBT_CREATEWENDA	Contains information passed to a WH_CBT hook procedure, CBTProc, before a window is created.
CBT_CREATEWNDW	Contains information passed to a WH_CBT hook procedure, CBTProc, before a window is created.
CBTACTIVATESTRUCT	Contains information passed to a WH_CBT hook procedure, CBTProc, before a window is activated.
CHANGEFILTERSTRUCT	Contains extended result information obtained by calling the ChangeWindowMessageFilterEx function.
CLIENTCREATESTRUCT	Contains information about the menu and first multiple-document interface (MDI) child window of an MDI client window.
COMBOBOXINFO	Contains combo box status information.
COMPAREITEMSTRUCT	Supplies the identifiers and application-supplied data for two items in a sorted, owner-drawn list box or combo box.
COPYDATASTRUCT	Contains data to be passed to another application by the WM_COPYDATA message.
CREATESTRUCTA	Defines the initialization parameters passed to the window procedure of an application. These members are identical to the parameters of the CreateWindowEx function.
CREATESTRUCTW	Defines the initialization parameters passed to the window procedure of an application. These members are identical to the parameters of the CreateWindowEx function.
CURSORINFO	Contains global cursor information.
CURSORSHAPE	Contains information about a cursor.
CWPRETSTRUCT	Defines the message parameters passed to a WH_CALLWNDPROCRET hook procedure, CallWndRetProc.
CWPSTRUCT	Defines the message parameters passed to a WH_CALLWNDPROC hook procedure, CallWndProc.
DEBUGHOOKINFO	Contains debugging information passed to a WH_DEBUG hook procedure, DebugProc.
DELETEITEMSTRUCT	Describes a deleted list box or combo box item.
DLGITEMTEMPLATE	Defines the dimensions and style of a control in a dialog box. One or more of these structures are combined with a DLGTEMPLATE structure to form a standard template for a dialog box.

TITLE	DESCRIPTION
DLGTEMPLATE	Defines the dimensions and style of a dialog box.
DRAWITEMSTRUCT	Provides information that the owner window uses to determine how to paint an owner-drawn control or menu item.
DRAWTEXTPARAMS	The DRAWTEXTPARAMS structure contains extended formatting options for the DrawTextEx function.
EVENTMSG	Contains information about a hardware message sent to the system message queue. This structure is used to store message information for the JournalPlaybackProc callback function.
FILTERKEYS	Contains information about the FilterKeys accessibility feature, which enables a user with disabilities to set the keyboard repeat rate (RepeatKeys), acceptance delay (SlowKeys), and bounce rate (BounceKeys).
FLASHWINFO	Contains the flash status for a window and the number of times the system should flash the window.
GESTURECONFIG	Gets and sets the configuration for enabling gesture messages and the type of this configuration.
GESTUREINFO	Stores information about a gesture.
GESTURENOTIFYSTRUCT	When transmitted with WM_GESTURENOTIFY messages, passes information about a gesture.
GUITHREADINFO	Contains information about a GUI thread.
HARDWAREINPUT	Contains information about a simulated message generated by an input device other than a keyboard or mouse.
HELPINFO	Contains information about an item for which context-sensitive help has been requested.
HELPWININFOA	Contains the size and position of either a primary or secondary Help window. An application can set this information by calling the WinHelp function with the HELP_SETWINPOS value.
HELPWININFOW	Contains the size and position of either a primary or secondary Help window. An application can set this information by calling the WinHelp function with the HELP_SETWINPOS value.
HIGHCONTRASTA	Contains information about the high contrast accessibility feature.
HIGHCONTRASTW	Contains information about the high contrast accessibility feature.

TITLE	DESCRIPTION
ICONINFO	Contains information about an icon or a cursor.
ICONINFOEXA	Contains information about an icon or a cursor. Extends ICONINFO. Used by GetIconInfoEx.
ICONINFOEXW	Contains information about an icon or a cursor. Extends ICONINFO. Used by GetIconInfoEx.
ICONMETRICSA	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.
ICONMETRICSW	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.
INPUT	Used by SendInput to store information for synthesizing input events such as keystrokes, mouse movement, and mouse clicks.
INPUT_INJECTION_VALUE	Contains the input injection details.
INPUT_MESSAGE_SOURCE	Contains information about the source of the input message.
INPUT_TRANSFORM	Defines the matrix that represents a transform on a message consumer.
KBDLLHOOKSTRUCT	Contains information about a low-level keyboard input event.
KEYBDINPUT	Contains information about a simulated keyboard event.
LASTINPUTINFO	Contains the time of the last input.
MDICREATESTRUCTA	Contains information about the class, title, owner, location, and size of a multiple-document interface (MDI) child window.
MDICREATESTRUCTW	Contains information about the class, title, owner, location, and size of a multiple-document interface (MDI) child window.
MDINEXTMENU	Contains information about the menu to be activated.
MEASUREITEMSTRUCT	Informs the system of the dimensions of an owner-drawn control or menu item. This allows the system to process user interaction with the control correctly.
MENUBARINFO	Contains menu bar information.

TITLE	DESCRIPTION
<a href="#">MENUGETOBJECTINFO</a>	Contains information about the menu that the mouse cursor is on.
<a href="#">MENUINFO</a>	Contains information about a menu.
<a href="#">MENUITEMINFOA</a>	Contains information about a menu item.
<a href="#">MENUITEMINFOW</a>	Contains information about a menu item.
<a href="#">MENUITEMTEMPLATE</a>	Defines a menu item in a menu template.
<a href="#">MENUITEMTEMPLATEHEADER</a>	Defines the header for a menu template. A complete menu template consists of a header and one or more menu item lists.
<a href="#">MINIMIZEDMETRICS</a>	Contains the scalable metrics associated with minimized windows.
<a href="#">MINMAXINFO</a>	Contains information about a window's maximized size and position and its minimum and maximum tracking size.
<a href="#">MONITORINFO</a>	The MONITORINFO structure contains information about a display monitor.The GetMonitorInfo function stores information in a MONITORINFO structure or a MONITORINFOEX structure.The MONITORINFO structure is a subset of the MONITORINFOEX structure.
<a href="#">MONITORINFOEXA</a>	The MONITORINFOEX structure contains information about a display monitor.The GetMonitorInfo function stores information into a MONITORINFOEX structure or a MONITORINFO structure.The MONITORINFOEX structure is a superset of the MONITORINFO structure.
<a href="#">MONITORINFOEXW</a>	The MONITORINFOEX structure contains information about a display monitor.The GetMonitorInfo function stores information into a MONITORINFOEX structure or a MONITORINFO structure.The MONITORINFOEX structure is a superset of the MONITORINFO structure.
<a href="#">MOUSEHOOKSTRUCT</a>	Contains information about a mouse event passed to a WH_MOUSE hook procedure, MouseProc.
<a href="#">MOUSEHOOKSTRUCTEX</a>	Contains information about a mouse event passed to a WH_MOUSE hook procedure, MouseProc. This is an extension of the MOUSEHOOKSTRUCT structure that includes information about wheel movement or the use of the X button.
<a href="#">MOUSEINPUT</a>	Contains information about a simulated mouse event.
<a href="#">MOUSEKEYS</a>	Contains information about the MouseKeys accessibility feature.
<a href="#">MOUSEMOVEPOINT</a>	Contains information about the mouse's location in screen coordinates.

TITLE	DESCRIPTION
MSG	Contains message information from a thread's message queue.
MSGBOXPARAMSA	Contains information used to display a message box. The MessageBoxIndirect function uses this structure.
MSGBOXPARAMSW	Contains information used to display a message box. The MessageBoxIndirect function uses this structure.
MSLLHOOKSTRUCT	Contains information about a low-level mouse input event.
MULTIKEYHELPA	Specifies a keyword to search for and the keyword table to be searched by Windows Help.
MULTIKEYHELPW	Specifies a keyword to search for and the keyword table to be searched by Windows Help.
NCCALCSIZE_PARAMS	Contains information that an application can use while processing the WM_NCCALCSIZE message to calculate the size, position, and valid contents of the client area of a window.
NMHDR	Contains information about a notification message.
NONCLIENTMETRICS	Contains the scalable metrics associated with the nonclient area of a nonminimized window.
NONCLIENTMETRICSW	Contains the scalable metrics associated with the nonclient area of a nonminimized window.
PAINTSTRUCT	The PAINTSTRUCT structure contains information for an application. This information can be used to paint the client area of a window owned by that application.
POINTER_DEVICE_CURSOR_INFO	Contains cursor ID mappings for pointer devices.
POINTER_DEVICE_INFO	Contains information about a pointer device. An array of these structures is returned from the GetPointerDevices function. A single structure is returned from a call to the GetPointerDevice function.
POINTER_DEVICE_PROPERTY	Contains pointer-based device properties (Human Interface Device (HID) global items that correspond to HID usages).
POINTER_INFO	Contains basic pointer information common to all pointer types. Applications can retrieve this information using the GetPointerInfo, GetPointerFrameInfo, GetPointerInfoHistory and GetPointerFrameInfoHistory functions.
POINTER_PEN_INFO	Defines basic pen information common to all pointer types.



TITLE	DESCRIPTION
POINTER_TOUCH_INFO	Defines basic touch information common to all pointer types.
POINTER_TYPE_INFO	Contains information about the pointer input type.
POWERBROADCAST_SETTING	Sent with a power setting event and contains data about the specific change.
RAWHID	Describes the format of the raw input from a Human Interface Device (HID).
RAWINPUT	Contains the raw input from a device.
RAWINPUTDEVICE	Defines information for the raw input devices.
RAWINPUTDEVICELIST	Contains information about a raw input device.
RAWINPUTHEADER	Contains the header information that is part of the raw input data.
RAWKEYBOARD	Contains information about the state of the keyboard.
RAWMOUSE	Contains information about the state of the mouse.
RID_DEVICE_INFO	Defines the raw input data coming from any device.
RID_DEVICE_INFO_HID	Defines the raw input data coming from the specified Human Interface Device (HID).
RID_DEVICE_INFO_KEYBOARD	Defines the raw input data coming from the specified keyboard.
RID_DEVICE_INFO_MOUSE	Defines the raw input data coming from the specified mouse.
SCROLLBARINFO	The SCROLLBARINFO structure contains scroll bar information.
SCROLLINFO	The SCROLLINFO structure contains scroll bar parameters to be set by the SetScrollInfo function (or SBM_SETSCROLLINFO message), or retrieved by the GetScrollInfo function (or SBM_GETSCROLLINFO message).
SERIALKEYSA	Contains information about the SerialKeys accessibility feature, which interprets data from a communication aid attached to a serial port as commands causing the system to simulate keyboard and mouse input.
SERIALKEYSW	Contains information about the SerialKeys accessibility feature, which interprets data from a communication aid attached to a serial port as commands causing the system to simulate keyboard and mouse input.

TITLE	DESCRIPTION
SOUNDSENTRYA	Contains information about the SoundSentry accessibility feature. When the SoundSentry feature is on, the computer displays a visual indication only when a sound is generated.
SOUNDSENTRYW	Contains information about the SoundSentry accessibility feature. When the SoundSentry feature is on, the computer displays a visual indication only when a sound is generated.
STICKYKEYS	Contains information about the StickyKeys accessibility feature.
STYLESTRUCT	Contains the styles for a window.
TITLEBARINFO	Contains title bar information.
TITLEBARINFOEX	Expands on the information described in the TITLEBARINFO structure by including the coordinates of each element of the title bar.
TOGGLEKEYS	Contains information about the ToggleKeys accessibility feature.
TOUCH_HIT_TESTING_INPUT	Contains information about the touch contact area reported by the touch digitizer.
TOUCH_HIT_TESTING_PROXIMITY_EVALUATION	Contains the hit test score that indicates whether the object is the likely target of the touch contact area, relative to other objects that intersect the touch contact area.
TOUCHINPUT	Encapsulates data for touch input.
TOUCHPREDICTIONPARAMETERS	Contains hardware input details that can be used to predict touch targets and help compensate for hardware latency when processing touch and gesture input that contains distance and velocity data.
TPMPARAMS	Contains extended parameters for the TrackPopupMenuEx function.
TRACKMOUSEEVENT	Used by the TrackMouseEvent function to track when the mouse pointer leaves a window or hovers over a window for a specified amount of time.
UPDATELAYEREDWINDOWINFO	Used by UpdateLayeredWindowIndirect to provide position, size, shape, content, and translucency information for a layered window.
USAGE_PROPERTIES	Contains device properties (Human Interface Device (HID) global items that correspond to HID usages) for any type of HID input device.
USEROBJECTFLAGS	Contains information about a window station or desktop handle.

TITLE	DESCRIPTION
WINDOWINFO	Contains window information.
WINDOWPLACEMENT	Contains information about the placement of a window on the screen.
WINDOWPOS	Contains information about the size and position of a window.
WNDCLASSA	Contains the window class attributes that are registered by the RegisterClass function.
WNDCLASSEXA	Contains window class information.
WNDCLASSEXW	Contains window class information.
WNDCLASSW	Contains the window class attributes that are registered by the RegisterClass function.
WTSSESSION_NOTIFICATION	Provides information about the session change notification. A service receives this structure in its HandlerEx function in response to a session change event.

## Enumerations

TITLE	DESCRIPTION
AR_STATE	Indicates the state of screen auto-rotation for the system. For example, whether auto-rotation is supported, and whether it is enabled by the user.
DIALOG_CONTROL_DPI_CHANGE_BEHAVIORS	Describes per-monitor DPI scaling behavior overrides for child windows within dialogs. The values in this enumeration are bitfields and can be combined.
DIALOG_DPI_CHANGE_BEHAVIORS	In Per Monitor v2 contexts, dialogs will automatically respond to DPI changes by resizing themselves and re-computing the positions of their child windows (here referred to as re-layouting).
FEEDBACK_TYPE	Specifies the visual feedback associated with an event.
INPUT_MESSAGE_DEVICE_TYPE	The type of device that sent the input message.
INPUT_MESSAGE_ORIGIN_ID	The ID of the input message source.
ORIENTATION_PREFERENCE	Indicates the screen orientation preference for a desktop app process.
POINTER_BUTTON_CHANGE_TYPE	Identifies a change in the state of a button associated with a pointer.
POINTER_DEVICE_CURSOR_TYPE	Identifies the pointer device cursor types.

TITLE	DESCRIPTION
POINTER_DEVICE_TYPE	Identifies the pointer device types.
POINTER_FEEDBACK_MODE	Identifies the visual feedback behaviors available to CreateSyntheticPointerDevice.
tagPOINTER_INPUT_TYPE	Identifies the pointer input types.

# AddClipboardFormatListener function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Places the given window in the system-maintained clipboard format listener list.

## Syntax

```
BOOL AddClipboardFormatListener(  
    HWND hwnd  
);
```

## Parameters

hwnd

Type: **HWND**

A handle to the window to be placed in the clipboard format listener list.

## Return value

Type: **BOOL**

Returns **TRUE** if successful, **FALSE** otherwise. Call [GetLastError](#) for additional details.

## Remarks

When a window has been added to the clipboard format listener list, it is posted a [WM\\_CLIPBOARDUPDATE](#) message whenever the contents of the clipboard have changed.

## Requirements

<b>Minimum supported client</b>	Windows Vista [desktop apps only]
<b>Minimum supported server</b>	Windows Server 2008 [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	winuser.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll
<b>API set</b>	ext-ms-win-ntuser-misc-l1-5-1 (introduced in Windows 10, version 10.0.14393)

## See also

[GetClipboardSequenceNumber](#)

[RemoveClipboardFormatListener](#)

[WM\\_CLIPBOARDUPDATE](#)

# ChangeClipboardChain function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Removes a specified window from the chain of clipboard viewers.

## Syntax

```
BOOL ChangeClipboardChain(  
    HWND hWndRemove,  
    HWND hWndNewNext  
);
```

## Parameters

`hWndRemove`

Type: **HWND**

A handle to the window to be removed from the chain. The handle must have been passed to the [SetClipboardViewer](#) function.

`hWndNewNext`

Type: **HWND**

A handle to the window that follows the *hWndRemove* window in the clipboard viewer chain. (This is the handle returned by [SetClipboardViewer](#), unless the sequence was changed in response to a [WM\\_CHANGECHAIN](#) message.)

## Return value

Type: **BOOL**

The return value indicates the result of passing the [WM\\_CHANGECHAIN](#) message to the windows in the clipboard viewer chain. Because a window in the chain typically returns **FALSE** when it processes **WM\_CHANGECHAIN**, the return value from **ChangeClipboardChain** is typically **FALSE**. If there is only one window in the chain, the return value is typically **TRUE**.

## Remarks

The window identified by *hWndNewNext* replaces the *hWndRemove* window in the chain. The [SetClipboardViewer](#) function sends a [WM\\_CHANGECHAIN](#) message to the first window in the clipboard viewer chain.

For an example, see [Removing a Window from the Clipboard Viewer Chain](#).

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]

<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	winuser.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll
<b>API set</b>	ext-ms-win-ntuser-misc-l1-5-1 (introduced in Windows 10, version 10.0.14393)

## See also

[ChangeClipboardChain](#)

[Clipboard](#)

**Conceptual**

**Reference**

[SetClipboardViewer](#)

[WM\\_CHANGECHAIN](#)



# CloseClipboard function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Closes the clipboard.

## Syntax

```
BOOL CloseClipboard();
```

## Parameters

This function has no parameters.

## 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

When the window has finished examining or changing the clipboard, close the clipboard by calling **CloseClipboard**. This enables other windows to access the clipboard.

Do not place an object on the clipboard after calling **CloseClipboard**.

### Examples

For an example, see [Example of a Clipboard Viewer](#).

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	winuser.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll
<b>API set</b>	ext-ms-win-ntuser-misc-l1-2-0 (introduced in Windows 8.1)

## See also

[Clipboard](#)

### Conceptual

[GetOpenClipboardWindow](#)

[OpenClipboard](#)

### Reference

# COPYDATASTRUCT structure (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Contains data to be passed to another application by the [WM\\_COPYDATA](#) message.

## Syntax

```
typedef struct tagCOPYDATASTRUCT {  
    ULONG_PTR dwData;  
    DWORD      cbData;  
    PVOID      lpData;  
} COPYDATASTRUCT, *PCOPYDATASTRUCT;
```

## Members

**dwData**

Type: **ULONG\_PTR**

The type of the data to be passed to the receiving application. The receiving application defines the valid types.

**cbData**

Type: **DWORD**

The size, in bytes, of the data pointed to by the **lpData** member.

**lpData**

Type: **PVOID**

The data to be passed to the receiving application. This member can be **NULL**.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Header</b>	winuser.h (include Windows.h)

## See also

[WM\\_COPYDATA](#)

# CountClipboardFormats function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Retrieves the number of different data formats currently on the clipboard.

## Syntax

```
int CountClipboardFormats();
```

## Parameters

This function has no parameters.

## Return value

Type: int

If the function succeeds, the return value is the number of different data formats currently on the clipboard.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	winuser.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

## See also

[Clipboard](#)

**Conceptual**

[EnumClipboardFormats](#)

**Reference**

[RegisterClipboardFormat](#)

# EmptyClipboard function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Empties the clipboard and frees handles to data in the clipboard. The function then assigns ownership of the clipboard to the window that currently has the clipboard open.

## Syntax

```
BOOL EmptyClipboard();
```

## Parameters

This function has no parameters.

## 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 calling **EmptyClipboard**, an application must open the clipboard by using the [OpenClipboard](#) function. If the application specifies a **NULL** window handle when opening the clipboard, **EmptyClipboard** succeeds but sets the clipboard owner to **NULL**. Note that this causes [SetClipboardData](#) to fail.

### Examples

For an example, see [Copying Information to the Clipboard](#).

## Requirements

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-l1-2-0 (introduced in Windows 8.1)

## See also

[Clipboard](#)

**Conceptual**

[OpenClipboard](#)

**Reference**

[SetClipboardData](#)

[WM\\_DESTROYCLIPBOARD](#)

# EnumClipboardFormats function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Enumerates the data formats currently available on the clipboard.

Clipboard data formats are stored in an ordered list. To perform an enumeration of clipboard data formats, you make a series of calls to the **EnumClipboardFormats** function. For each call, the *format* parameter specifies an available clipboard format, and the function returns the next available clipboard format.

## Syntax

```
UINT EnumClipboardFormats(  
    UINT format  
);
```

## Parameters

**format**

Type: **UINT**

A clipboard format that is known to be available.

To start an enumeration of clipboard formats, set *format* to zero. When *format* is zero, the function retrieves the first available clipboard format. For subsequent calls during an enumeration, set *format* to the result of the previous **EnumClipboardFormats** call.

## Return value

Type: **UINT**

If the function succeeds, the return value is the clipboard format that follows the specified format, namely the next available clipboard format.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#). If the clipboard is not open, the function fails.

If there are no more clipboard formats to enumerate, the return value is zero. In this case, the [GetLastError](#) function returns the value **ERROR\_SUCCESS**. This lets you distinguish between function failure and the end of enumeration.

## Remarks

You must open the clipboard before enumerating its formats. Use the [OpenClipboard](#) function to open the clipboard. The **EnumClipboardFormats** function fails if the clipboard is not open.

The **EnumClipboardFormats** function enumerates formats in the order that they were placed on the clipboard. If you are copying information to the clipboard, add clipboard objects in order from the most descriptive clipboard format to the least descriptive clipboard format. If you are pasting information from the clipboard, retrieve the first clipboard format that you can handle. That will be the most descriptive clipboard format that you can handle.

The system provides automatic type conversions for certain clipboard formats. In the case of such a format, this function enumerates the specified format, then enumerates the formats to which it can be converted. For more information, see [Standard Clipboard Formats](#) and [Synthesized Clipboard Formats](#).

#### Examples

For an example, see [Example of a Clipboard Viewer](#).

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	winuser.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll
<b>API set</b>	ext-ms-win-ntuser-misc-l1-5-1 (introduced in Windows 10, version 10.0.14393)

## See also

[Clipboard](#)

#### Conceptual

[CountClipboardFormats](#)

[OpenClipboard](#)

#### Reference

[RegisterClipboardFormat](#)



# GetClipboardData function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Retrieves data from the clipboard in a specified format. The clipboard must have been opened previously.

## Syntax

```
HANDLE GetClipboardData(  
    UINT uFormat  
);
```

## Parameters

**uFormat**

Type: **UINT**

A clipboard format. For a description of the standard clipboard formats, see [Standard Clipboard Formats](#).

## Return value

Type: **HANDLE**

If the function succeeds, the return value is the handle to a clipboard object in the specified format.

If the function fails, the return value is **NULL**. To get extended error information, call [GetLastError](#).

## Remarks

**Caution** Clipboard data is not trusted. Parse the data carefully before using it in your application.

An application can enumerate the available formats in advance by using the [EnumClipboardFormats](#) function.

The clipboard controls the handle that the **GetClipboardData** function returns, not the application. The application should copy the data immediately. The application must not free the handle nor leave it locked. The application must not use the handle after the [EmptyClipboard](#) or [CloseClipboard](#) function is called, or after the [SetClipboardData](#) function is called with the same clipboard format.

The system performs implicit data format conversions between certain clipboard formats when an application calls the **GetClipboardData** function. For example, if the [CF\\_OEMTEXT](#) format is on the clipboard, a window can retrieve data in the [CF\\_TEXT](#) format. The format on the clipboard is converted to the requested format on demand. For more information, see [Synthesized Clipboard Formats](#).

### Examples

For an example, see [Copying Information to the Clipboard](#).

## Requirements

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-l1-2-0 (introduced in Windows 8.1)

# See also

[Clipboard](#)

[CloseClipboard](#)

## Conceptual

[EmptyClipboard](#)

[EnumClipboardFormats](#)

## Reference

[SetClipboardData](#)

# GetClipboardFormatNameA function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Retrieves from the clipboard the name of the specified registered format. The function copies the name to the specified buffer.

## Syntax

```
int GetClipboardFormatNameA(  
    UINT format,  
    LPSTR lpszFormatName,  
    int cchMaxCount  
);
```

## Parameters

`format`

Type: **UINT**

The type of format to be retrieved. This parameter must not specify any of the predefined clipboard formats.

`lpszFormatName`

Type: **LPTSTR**

The buffer that is to receive the format name.

`cchMaxCount`

Type: **int**

The maximum length, in characters, of the string to be copied to the buffer. If the name exceeds this limit, it is truncated.

## Return value

Type: **int**

If the function succeeds, the return value is the length, in characters, of the string copied to the buffer.

If the function fails, the return value is zero, indicating that the requested format does not exist or is predefined. To get extended error information, call [GetLastError](#).

## Remarks

### Security Considerations

Using this function incorrectly might compromise the security of your program. For example, miscalculating the proper size of the *lpszFormatName* buffer, especially when the application is used in both ANSI and Unicode versions, can cause a buffer overflow. Also, note that the string is truncated if it is longer than the *cchMaxCount* parameter, which can lead to loss of information.

### Examples

For an example, see [Example of a Clipboard Viewer](#).

#### NOTE

The `winuser.h` header defines `GetClipboardFormatName` 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

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

## See also

[Clipboard](#)

Conceptual

[EnumClipboardFormats](#)

Reference

[RegisterClipboardFormat](#)

# GetClipboardFormatNameW function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Retrieves from the clipboard the name of the specified registered format. The function copies the name to the specified buffer.

## Syntax

```
int GetClipboardFormatNameW(  
    UINT    format,  
    LPWSTR  lpszFormatName,  
    int     cchMaxCount  
);
```

## Parameters

`format`

Type: **UINT**

The type of format to be retrieved. This parameter must not specify any of the predefined clipboard formats.

`lpszFormatName`

Type: **LPTSTR**

The buffer that is to receive the format name.

`cchMaxCount`

Type: **int**

The maximum length, in characters, of the string to be copied to the buffer. If the name exceeds this limit, it is truncated.

## Return value

Type: **int**

If the function succeeds, the return value is the length, in characters, of the string copied to the buffer.

If the function fails, the return value is zero, indicating that the requested format does not exist or is predefined. To get extended error information, call [GetLastError](#).

## Remarks

### Security Considerations

Using this function incorrectly might compromise the security of your program. For example, miscalculating the proper size of the *lpszFormatName* buffer, especially when the application is used in both ANSI and Unicode versions, can cause a buffer overflow. Also, note that the string is truncated if it is longer than the *cchMaxCount* parameter, which can lead to loss of information.

### Examples

For an example, see [Example of a Clipboard Viewer](#).

#### NOTE

The `winuser.h` header defines `GetClipboardFormatName` 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

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

## See also

[Clipboard](#)

Conceptual

[EnumClipboardFormats](#)

Reference

[RegisterClipboardFormat](#)

# GetClipboardOwner function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Retrieves the window handle of the current owner of the clipboard.

## Syntax

```
HWND GetClipboardOwner();
```

## Parameters

This function has no parameters.

## Return value

Type: **HWND**

If the function succeeds, the return value is the handle to the window that owns the clipboard.

If the clipboard is not owned, the return value is **NULL**. To get extended error information, call [GetLastError](#).

## Remarks

The clipboard can still contain data even if the clipboard is not currently owned.

In general, the clipboard owner is the window that last placed data in clipboard. The [EmptyClipboard](#) function assigns clipboard ownership.

### Examples

For an example, see [Example of a Clipboard Viewer](#).

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	winuser.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll
<b>API set</b>	ext-ms-win-ntuser-misc-l1-2-0 (introduced in Windows 8.1)

## See also

[Clipboard](#)

**Conceptual**

[EmptyClipboard](#)

[GetClipboardViewer](#)

**Reference**



# GetClipboardSequenceNumber function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Retrieves the clipboard sequence number for the current window station.

## Syntax

```
DWORD GetClipboardSequenceNumber();
```

## Parameters

This function has no parameters.

## Return value

Type: **DWORD**

The return value is the clipboard sequence number. If you do not have **WINSTA\_ACCESSCLIPBOARD** access to the window station, the function returns zero.

## Remarks

The system keeps a serial number for the clipboard for each window station. This number is incremented whenever the contents of the clipboard change or the clipboard is emptied. You can track this value to determine whether the clipboard contents have changed and optimize creating DataObjects. If clipboard rendering is delayed, the sequence number is not incremented until the changes are rendered.

## Requirements

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-l1-5-1 (introduced in Windows 10, version 10.0.14393)

## See also



# GetClipboardViewer function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Retrieves the handle to the first window in the clipboard viewer chain.

## Syntax

```
HWND GetClipboardViewer();
```

## Parameters

This function has no parameters.

## Return value

Type: **HWND**

If the function succeeds, the return value is the handle to the first window in the clipboard viewer chain.

If there is no clipboard viewer, the return value is **NULL**. To get extended error information, call [GetLastError](#).

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	winuser.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll
<b>API set</b>	ext-ms-win-ntuser-misc-l1-5-1 (introduced in Windows 10, version 10.0.14393)

## See also

[Clipboard](#)

**Conceptual**

[GetClipboardOwner](#)

**Reference**



# GetOpenClipboardWindow function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Retrieves the handle to the window that currently has the clipboard open.

## Syntax

```
HWND GetOpenClipboardWindow();
```

## Parameters

This function has no parameters.

## Return value

Type: **HWND**

If the function succeeds, the return value is the handle to the window that has the clipboard open. If no window has the clipboard open, the return value is **NULL**. To get extended error information, call [GetLastError](#).

## Remarks

If an application or DLL specifies a **NULL** window handle when calling the [OpenClipboard](#) function, the clipboard is opened but is not associated with a window. In such a case, **GetOpenClipboardWindow** returns **NULL**.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	winuser.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

## See also

[Clipboard](#)

**Conceptual**

[GetClipboardOwner](#)

[GetClipboardViewer](#)

[OpenClipboard](#)

**Reference**

# GetPriorityClipboardFormat function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Retrieves the first available clipboard format in the specified list.

## Syntax

```
int GetPriorityClipboardFormat(  
    UINT *paFormatPriorityList,  
    int  cFormats  
);
```

## Parameters

`paFormatPriorityList`

Type: **UINT\***

The clipboard formats, in priority order. For a description of the standard clipboard formats, see [Standard Clipboard Formats](#).

`cFormats`

Type: **int**

The number of entries in the *paFormatPriorityList* array. This value must not be greater than the number of entries in the list.

## Return value

Type: **int**

If the function succeeds, the return value is the first clipboard format in the list for which data is available. If the clipboard is empty, the return value is NULL. If the clipboard contains data, but not in any of the specified formats, the return value is -1. To get extended error information, call [GetLastError](#).

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	winuser.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll

API set	ext-ms-win-ntuser-misc-l1-5-1 (introduced in Windows 10, version 10.0.14393)

# See also

[Clipboard](#)

## Conceptual

[CountClipboardFormats](#)

[EnumClipboardFormats](#)

[GetClipboardFormatName](#)

[IsClipboardFormatAvailable](#)

## Reference

[RegisterClipboardFormat](#)



# GetUpdatedClipboardFormats function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Retrieves the currently supported clipboard formats.

## Syntax

```
BOOL GetUpdatedClipboardFormats(  
    PUINT lpuiFormats,  
    UINT cFormats,  
    PUINT pcFormatsOut  
);
```

## Parameters

`lpuiFormats`

Type: **PUINT**

An array of clipboard formats. For a description of the standard clipboard formats, see [Standard Clipboard Formats](#).

`cFormats`

Type: **UINT**

The number of entries in the array pointed to by *lpuiFormats*.

`pcFormatsOut`

Type: **PUINT**

The actual number of clipboard formats in the array pointed to by *lpuiFormats*.

## Return value

Type: **BOOL**

The function returns **TRUE** if successful; otherwise, **FALSE**. Call [GetLastError](#) for additional details.

## Requirements

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

# IsClipboardFormatAvailable function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Determines whether the clipboard contains data in the specified format.

## Syntax

```
BOOL IsClipboardFormatAvailable(  
    UINT format  
);
```

## Parameters

format

Type: **UINT**

A standard or registered clipboard format. For a description of the standard clipboard formats, see [Standard Clipboard Formats](#).

## Return value

Type: **BOOL**

If the clipboard format is available, the return value is nonzero.

If the clipboard format is not available, the return value is zero. To get extended error information, call [GetLastError](#).

## Remarks

Typically, an application that recognizes only one clipboard format would call this function when processing the [WM\\_INITMENU](#) or [WM\\_INITMENUPOPUP](#) message. The application would then enable or disable the Paste menu item, depending on the return value. Applications that recognize more than one clipboard format should use the [GetPriorityClipboardFormat](#) function for this purpose.

### Examples

For an example, see [Pasting Information from the Clipboard](#).

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	winuser.h (include Windows.h)

<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll
<b>API set</b>	ext-ms-win-ntuser-misc-l1-2-0 (introduced in Windows 8.1)

# See also

[Clipboard](#)

## Conceptual

[CountClipboardFormats](#)

[EnumClipboardFormats](#)

[GetPriorityClipboardFormat](#)

## Reference

[RegisterClipboardFormat](#)

[WM\\_INITMENU](#)

[WM\\_INITMENUPOPUP](#)

# OpenClipboard function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Opens the clipboard for examination and prevents other applications from modifying the clipboard content.

## Syntax

```
BOOL OpenClipboard(  
    HWND hWndNewOwner  
);
```

## Parameters

`hWndNewOwner`

Type: **HWND**

A handle to the window to be associated with the open clipboard. If this parameter is **NULL**, the open clipboard is associated with the current task.

## 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

**OpenClipboard** fails if another window has the clipboard open.

An application should call the [CloseClipboard](#) function after every successful call to **OpenClipboard**.

The window identified by the *hWndNewOwner* parameter does not become the clipboard owner unless the [EmptyClipboard](#) function is called.

If an application calls **OpenClipboard** with *hWnd* set to **NULL**, [EmptyClipboard](#) sets the clipboard owner to **NULL**; this causes [SetClipboardData](#) to fail.

### Examples

For an example, see [Copying Information to the Clipboard](#).

## Requirements

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-l1-2-0 (introduced in Windows 8.1)

# See also

[Clipboard](#)

[CloseClipboard](#)

## Conceptual

[EmptyClipboard](#)

## Reference

# RegisterClipboardFormatA function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Registers a new clipboard format. This format can then be used as a valid clipboard format.

## Syntax

```
UINT RegisterClipboardFormatA(  
    LPCSTR lpszFormat  
);
```

## Parameters

`lpszFormat`

Type: LPCTSTR

The name of the new format.

## Return value

Type: UINT

If the function succeeds, the return value identifies the registered clipboard format.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

## Remarks

If a registered format with the specified name already exists, a new format is not registered and the return value identifies the existing format. This enables more than one application to copy and paste data using the same registered clipboard format. Note that the format name comparison is case-insensitive.

Registered clipboard formats are identified by values in the range 0xC000 through 0xFFFF.

When registered clipboard formats are placed on or retrieved from the clipboard, they must be in the form of an **HGLOBAL** value.

### Examples

For an example, see [Registering a Clipboard Format](#).

#### NOTE

The winuser.h header defines RegisterClipboardFormat 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

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-l1-2-0 (introduced in Windows 8.1)

## See also

[Clipboard](#)

### Conceptual

[CountClipboardFormats](#)

[EnumClipboardFormats](#)

[GetClipboardFormatName](#)

### Reference



# RegisterClipboardFormatW function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Registers a new clipboard format. This format can then be used as a valid clipboard format.

## Syntax

```
UINT RegisterClipboardFormatW(  
    LPCWSTR lpszFormat  
);
```

## Parameters

`lpszFormat`

Type: LPCTSTR

The name of the new format.

## Return value

Type: UINT

If the function succeeds, the return value identifies the registered clipboard format.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

## Remarks

If a registered format with the specified name already exists, a new format is not registered and the return value identifies the existing format. This enables more than one application to copy and paste data using the same registered clipboard format. Note that the format name comparison is case-insensitive.

Registered clipboard formats are identified by values in the range 0xC000 through 0xFFFF.

When registered clipboard formats are placed on or retrieved from the clipboard, they must be in the form of an **HGLOBAL** value.

### Examples

For an example, see [Registering a Clipboard Format](#).

#### NOTE

The winuser.h header defines RegisterClipboardFormat 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

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-l1-2-0 (introduced in Windows 8.1)

# See also

[Clipboard](#)

## Conceptual

[CountClipboardFormats](#)

[EnumClipboardFormats](#)

[GetClipboardFormatName](#)

## Reference

# RemoveClipboardFormatListener function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Removes the given window from the system-maintained clipboard format listener list.

## Syntax

```
BOOL RemoveClipboardFormatListener(  
    HWND hwnd  
);
```

## Parameters

hwnd

Type: **HWND**

A handle to the window to remove from the clipboard format listener list.

## Return value

Type: **BOOL**

Returns **TRUE** if successful, **FALSE** otherwise. Call [GetLastError](#) for additional details.

## Remarks

When a window has been removed from the clipboard format listener list, it will no longer receive [WM\\_CLIPBOARDUPDATE](#) messages.

## Requirements

<b>Minimum supported client</b>	Windows Vista [desktop apps only]
<b>Minimum supported server</b>	Windows Server 2008 [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	winuser.h (include Windows.h)
<b>Library</b>	User32.lib
<b>DLL</b>	User32.dll
<b>API set</b>	ext-ms-win-ntuser-misc-l1-5-1 (introduced in Windows 10, version 10.0.14393)

## See also

[AddClipboardFormatListener](#)

[GetClipboardSequenceNumber](#)

[WM\\_CLIPBOARDUPDATE](#)

# SetClipboardData function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Places data on the clipboard in a specified clipboard format. The window must be the current clipboard owner, and the application must have called the [OpenClipboard](#) function. (When responding to the [WM\\_RENDERFORMAT](#) message, the clipboard owner must not call [OpenClipboard](#) before calling [SetClipboardData](#).)

## Syntax

```
HANDLE SetClipboardData(  
    UINT    uFormat,  
    HANDLE  hMem  
);
```

## Parameters

uFormat

Type: **UINT**

The clipboard format. This parameter can be a registered format or any of the standard clipboard formats. For more information, see [Standard Clipboard Formats](#) and [Registered Clipboard Formats](#).

hMem

Type: **HANDLE**

A handle to the data in the specified format. This parameter can be **NULL**, indicating that the window provides data in the specified clipboard format (renders the format) upon request; this is known as [delayed rendering](#). If a window delays rendering, it must process the [WM\\_RENDERFORMAT](#) and [WM\\_RENDERALLFORMATS](#) messages.

If [SetClipboardData](#) succeeds, the system owns the object identified by the *hMem* parameter. The application may not write to or free the data once ownership has been transferred to the system, but it can lock and read from the data until the [CloseClipboard](#) function is called. (The memory must be unlocked before the Clipboard is closed.) If the *hMem* parameter identifies a memory object, the object must have been allocated using the function with the **GMEM\_MOVEABLE** flag.

## Return value

Type: **HANDLE**

If the function succeeds, the return value is the handle to the data.

If the function fails, the return value is **NULL**. To get extended error information, call [GetLastError](#).

## Remarks

**Windows 8:** Bitmaps to be shared with Windows Store app apps must be in the **CF\_BITMAP** format (device-dependent bitmap).

If an application calls [SetClipboardData](#) in response to [WM\\_RENDERFORMAT](#) or [WM\\_RENDERALLFORMATS](#),

the application should not use the handle after **SetClipboardData** has been called.

If an application calls [OpenClipboard](#) with `hwnd` set to **NULL**, [EmptyClipboard](#) sets the clipboard owner to **NULL**; this causes **SetClipboardData** to fail.

The system performs implicit data format conversions between certain clipboard formats when an application calls the [GetClipboardData](#) function. For example, if the **CF\_OEMTEXT** format is on the clipboard, a window can retrieve data in the **CF\_TEXT** format. The format on the clipboard is converted to the requested format on demand. For more information, see [Synthesized Clipboard Formats](#).

#### Examples

For an example, see [Copying Information to the Clipboard](#).

## Requirements

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-l1-2-0 (introduced in Windows 8.1)

## See also

[Clipboard](#)

[CloseClipboard](#)

#### Conceptual

[GetClipboardData](#)

[OpenClipboard](#)

#### Reference

[RegisterClipboardFormat](#)

[WM\\_RENDERALLFORMATS](#)

[WM\\_RENDERFORMAT](#)

# SetClipboardViewer function (winuser.h)

1/15/2021 • 2 minutes to read • [Edit Online](#)

Adds the specified window to the chain of clipboard viewers. Clipboard viewer windows receive a [WM\\_DRAWCLIPBOARD](#) message whenever the content of the clipboard changes. This function is used for backward compatibility with earlier versions of Windows.

## Syntax

```
HWND SetClipboardViewer(  
    HWND hWndNewViewer  
);
```

## Parameters

`hWndNewViewer`

Type: **HWND**

A handle to the window to be added to the clipboard chain.

## Return value

Type: **HWND**

If the function succeeds, the return value identifies the next window in the clipboard viewer chain. If an error occurs or there are no other windows in the clipboard viewer chain, the return value is NULL. To get extended error information, call [GetLastError](#).

## Remarks

The windows that are part of the clipboard viewer chain, called clipboard viewer windows, must process the clipboard messages [WM\\_CHANGECHAIN](#) and [WM\\_DRAWCLIPBOARD](#). Each clipboard viewer window calls the [SendMessage](#) function to pass these messages to the next window in the clipboard viewer chain.

A clipboard viewer window must eventually remove itself from the clipboard viewer chain by calling the [ChangeClipboardChain](#) function — for example, in response to the [WM\\_DESTROY](#) message.

The **SetClipboardViewer** function exists to provide backward compatibility with earlier versions of Windows. The clipboard viewer chain can be broken by an application that fails to handle the clipboard chain messages properly. New applications should use more robust techniques such as the clipboard sequence number or the registration of a clipboard format listener. For further details on these alternative techniques, see [Monitoring Clipboard Contents](#).

### Examples

For an example, see [Adding a Window to the Clipboard Viewer Chain](#).

## Requirements

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-l1-5-1 (introduced in Windows 10, version 10.0.14393)

## See also

[ChangeClipboardChain](#)

[Clipboard](#)

**Conceptual**

[GetClipboardViewer](#)

**Reference**

[SendMessage](#)