256-Color Support in OLE 2.0 Containers

PSS ID Number: Q98872
Authored 16-May-1993 Last modified 24-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.0
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SUMMARY

If a client application draws a 256-color object in OLE version 1.0, the palette is always realized and used during the OleDraw command. As a result, the palette from the last object displayed is always used, causing previously drawn objects to update poorly.

In OLE version 2.0, OleDraw does not realize the palette automatically. Instead, a container must call IViewObject::GetColorSet() to retrieve the logical palette for each of its displayed objects. With this information, the container can construct a palette that best suits all of its displayed objects.

MORE INFORMATION

To get the equivalent of OLE 1.0 functionality in an OLE 2.0 container, IViewObject::GetColorSet must still be used to get a logical palette for an object, and in turn a palette must be created and realized from this information.

The following C++ function demonstrates the implementation of a drawing routine for an OLE 2.0 container that behaves similar to an OLE 1.0 client application.

Sample Code

```
void DrawObject(LPOLEOBJECT lpOleObject)
{
LPVIEWOBJECT lpViewObject;

    // Get a pointer to IViewObject.
    lpOleObject->QueryInterface(IID_IViewObject,
                                (LPVOID FAR *) &lpViewObject);

    // If the QI succeeds, get the LOGPALETTE for the object.
    if (lpView)
        lpView->GetColorSet(DVASPECT_CONTENT, -1, NULL, NULL, NULL,
                            &pColorSet);
```

```
    HPALETTE hPal=NULL;
    HPALETTE hOldPal=NULL;

    // If a LOGPALETTE was returned (not guaranteed), create the
    // palette and realize it. Note: A smarter application
    // would want to get the LOGPALETTE for each of its visible
    // objects, and try to create a palette that satisfies all of the
    // visible objects. Also, OleStdFree() is use to free the
    // returned LOGPALETTE.
    if ((pColorSet))
        {
        hPal = CreatePalette((const LPLOGPALETTE) pColorSet);
        hOldPal = SelectPalette(hDC, hPal, FALSE);
        RealizePalette(hDC);
        OleStdFree(pColorSet);
        }

    // Draw the object.
    OleDraw(m_lpOleObject, DVASPECT_CONTENT, hDC, &rect);

    // If we created a palette, restore the old one and destroy
    // the object.
    if (hPal)
        {
        SelectPalette(hDC,hOldPal,FALSE);
        DeleteObject(hPal);
        }

    // If a view pointer was successfully returned, it needs to be
    // released.
    if (lpView)
        lpView->Release();
}
```

Additional reference words: 1.00 2.00 3.50 4.00 95
KBCategory: kbole kbprg
KBSubcategory: LeTwoPrs

Adding Type Libraries as Resources to .DLL and .EXE Files

PSS ID Number: Q122285
Authored 01-Nov-1994 Last modified 21-Apr-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.02

A type library can be added as a resource to an .EXE or .DLL file by using the following statement in the .RC file:

```
1 typelib TEST.TLB
```

TEST.TLB is the type library to be added as a resource.

If a type library is added as a resource, a separate .TLB file need not be shipped with the application. LoadTypeLib() and LoadRegTypeLib() can be used to load the type library from the .EXE or .DLL file in which it exists as a resource.

NOTE: OLE version 2.02 or higher is needed to read a type library from an .EXE file.

Additional reference words: 2.00 Automation
KBCategory: kbole kbprg
KBSubcategory: LeTwoAto

BUG: (I)CntrOutl Does Not Set Target Device Information

PSS ID Number: Q108310
Authored 08-Dec-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

During the print process from (I)CntrOutl, a server application is asked to render its display with the TARGETDEVICE structure set to NULL instead of containing a valid structure. This implies that (I)CntrOutl wants a screen representation rather than a printed representation of the object.

CAUSE

(I)CntrOutl is not filling out the TARGETDEVICE structure; it prefers to use a screen representation on the printer, rather than request the printed presentation.

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here as it becomes available.

Additional reference words: 2.01 3.50 4.00
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoTls

BUG: Accelerator Causes Crash in ISvrOutl Embedded in C12Test

PSS ID Number: Q108311
Authored 08-Dec-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

After embedding an ISvrOutl object in C12Test or C132Test, pressing the ALT+BACKSPACE accelerator (Undo) results in a general protection (GP) fault within ISvrOutl.

STATUS

Microsoft has confirmed this to be a problem in the C12Test (C132Test) test application. We are researching this problem and will post new information here as it becomes available.

Additional reference words: 2.01 3.50 4.00 Outline Server Inplace GPF gp-fault crash
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoTls

BUG: Borland WINSIGHT Causes GP Faults w/ Some OLE Sample

PSS ID Number: Q112410
Authored 08-Mar-1994 Last modified 21-Apr-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

SYMPTOMS

If Borland's WINSIGHT tool is running when you start some of the sample applications that have been included in the OLE 2.01 SDK, a general protection (GP) Fault will occur before the main application window appears.

The following OLE 2.01 SDK sample applications are affected:

```
cl2test.exe
sr2test.exe
outline.exe
cntroutl.exe
svroutl.exe
icntrotl.exe
isvrotl.exe
```

STATUS

Microsoft has confirmed this to be a problem in the OLE SDK version 2.01. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: 2.01 toolkit gpf gp-fault buglist2.01
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoTls

BUG: Cannot Paste Link SR2TEST Object in OLE 1.0 Client

PSS ID Number: Q108932
Authored 20-Dec-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

The Paste Link option is not available in an OLE 1.0 client application after copying an unsaved SR2TEST object to the clipboard.

CAUSE

SR2TEST is not setting the OLEMISC_CANLINKBYOLE1 bit in the Link Source Descriptor that it places onto the clipboard.

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here as it becomes available.

Additional reference words: 2.01 3.50 4.00 test
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoTls

BUG: CL2TEST Does Not Display Prompt String from GetCurFile()

PSS ID Number: Q111014
Authored 03-Feb-1994 Last modified 24-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

IPersistFile::GetCurFile() returns S_FALSE to indicate that the document has no currently associated file. In this case, GetCurFile() returns the default prompt string for the filename (as would be displayed in the Save As dialog box under the File menu) in the lplpszFileName parameter.

When CL2TEST calls IPersistFile::GetCurFile(), and GetCurFile() returns S_FALSE, CL2TEST fails to display the returned prompt string.

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

MORE INFORMATION

To call IPersistFile::GetCurFile() on an object embedded in CL2TEST, click the object using the right mouse button, choose PersistFile Method from the pop-up menu, then choose GetCurFile.

Additional reference words: 2.01 3.50 4.00 95 Toolkit
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoTls

BUG: CL2TEST Fails to Parse Filenames with Extended

PSS ID Number: Q109548
Authored 04-Jan-1994 Last modified 16-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

SYMPTOMS

When attempting to insert a new embedded object from a file, CL2TEST fails with the error STG_E_FILENOTFOUND if the filename or directory name provided contains an extended character.

STATUS

Microsoft has confirmed this to be a problem with the CL2TEST sample application in the Object Linking and Embedding Software Development Kit (SDK) version 2.01. We are researching this problem and will post new information here as it becomes available.

This problem does not occur with 32-bit OLE.

MORE INFORMATION

To insert a new embedded object from a file into a CL2TEST document, from use CL2TEST's Insert menu, choose Embed From File.

Additional reference words: 2.01 toolkit buglist2.01
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoTls

BUG: CL2TEST Handles Icon Aspect Incorrectly

PSS ID Number: Q111339
Authored 09-Feb-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

Selecting the iconic aspect when inserting an object in CL2TEST results in the object being created and displayed with the content aspect.

CAUSE

CL2TEST is passing the DVASPECT_CONTENT flag to the embedded object when calling IViewObject::Draw. The correct flag to pass is DVASPECT_ICON.

STATUS

Microsoft has confirmed this to be a problem in the CL2TEST test application included with the Object Linking and Embedding SDK version 2.01. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

MORE INFORMATION

To specify a custom FORMATETC when inserting an object into CL2TEST, choose the "Select FormatETC" button found on CL2TEST's Create Invisible dialog box (from the Insert menu, choose Object).

Additional reference words: 2.01 3.50 4.00
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoTls

BUG: CL2TEST Not Properly Activating Links to Embedded

PSS ID Number: Q111340
Authored 09-Feb-1994 Last modified 16-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

SYMPTOMS

Double-clicking a link to an embedded object in CL2TEST results in the activation remaining on CL2TEST. The correct action would be for the container of the embedded object to obtain the activation and edit the object visually (if possible).

STATUS

Microsoft has confirmed this to be a problem in the Object Linking and Embedding SDK version 2.01.

This is not a problem with the Visual C++ 2.x CL32TEST under Windows NT 3.5.

Additional reference words: 2.01 toolkit buglist2.01
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoTls

BUG: CreateFromTemplate of OLE 2 Object into OLE 1 Container

PSS ID Number: Q111595
Authored 14-Feb-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

An OLE 2.01 or later object is inserted into an OLE 1.0 container using OleCreateFromTemplate(). The OLE libraries start the server, but then the object creation fails, or a blank object is created.

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: 2.01 3.50 4.00
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoArc

BUG: Deleting an Open Packager Object Causes GP Fault

PSS ID Number: Q108931
Authored 20-Dec-1993 Last modified 15-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

SYMPTOMS

Deleting an object from within the Object Packager results in a general protection (GP) fault.

CAUSE

The object is being edited in its server as a result of double-clicking the Content window. Deleting the object while in this state causes the Object Packager to GP fault.

STATUS

Microsoft has confirmed this to be a problem in Object Packager version 2.01. We are researching this problem and will post new information here as it becomes available.

This problem does not occur on Windows NT 3.5, with Object Packager 3.5.

Additional reference words: 2.01 GPF gp-fault crash
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoApp

BUG: DIB Can Be Returned Only on TYMED_HGLOBAL

PSS ID Number: Q98679
Authored 11-May-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.0
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

IDataObject::GetData returns DV_E_FORMATETC when requesting a device-independent bitmap (DIB) from an OLE object.

CAUSE

The DIB must be returned in a STGMEDIUM of type TYMED_HGLOBAL.

RESOLUTION

Add TYMED_HGLOBAL to the tymed member of the FORMATETC structure being passed to IDataObject::GetData.

STATUS

Microsoft has confirmed this to be a problem the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledgebase as it becomes available.

Additional reference words: 2.00 3.50 4.00
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoDdc

BUG: Embedded Object's Size Changes When it Is Run

PSS ID Number: Q110872
Authored 01-Feb-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

A non-running embedded object is resized in an OLE 2.01 container application. The next time the object is run, it snaps back to its previous size.

CAUSE

The container application failed to call IOleObject::SetExtent() after calling OleRun().

RESOLUTION

IOleObject::SetExtent() can be called only when an object is running. If an object is resized while it is not running, the container application should take note of this. When the object is subsequently run, the container should then inform the object of its new size by calling IOleObject::SetExtent().

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base when it becomes available.

Additional reference words: 2.01 3.50 4.00
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoCdt

BUG: First Entry in Paste Special Dialog Is Blank

PSS ID Number: Q110798
Authored 28-Jan-1994 Last modified 24-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

SYMPTOMS

The first format entry in the Paste Special dialog box is blank after copying an OLE2Link object from an OLE 1.0 client to the clipboard.

NOTE: An OLE2Link object is created by the OLE 1.0 compatibility layer when an OLE 2.0 link is copied from an OLE 2.0 application to the clipboard, and then pasted into an OLE 1.0 client application.

CAUSE

The CLSID and lpszFullUserTypeName of the OBJECTDESCRIPTOR structure are left blank by the OLE 1.0 compatibility layer.

STATUS

Microsoft has confirmed this to be a problem in the OLE libraries version 2.01. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

This is not a problem in 32-bit OLE.

Additional reference words: 2.01 buglist2.01
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoArc

BUG: Iconic OLE Object Prints as Black Rectangle on

PSS ID Number: Q110796
Authored 28-Jan-1994 Last modified 16-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

SYMPTOMS

Calling OleDraw() or IViewObject::Draw() to print an iconic OLE object to a PostScript printer results in a black rectangle being drawn.

CAUSE

Internally, the OLE DLLs (dynamic-link libraries) are using the DrawIcon() function to draw the icon. DrawIcon() requires the target device to support the SRCINVERT ROP (raster operation) code. PostScript does not support this particular ROP code.

RESOLUTION

An application can call OleDraw() or IViewObject::Draw() using a memory DC (device context). A bitmap of the memory DC can be retrieved, which can be used with the BitBlt() function to get the correct output to the printer.

STATUS

Microsoft has confirmed this to be a problem in the OLE Libraries version 2.01.

This is not a problem in 32-bit OLE.

Additional reference words: 2.01
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoPrs

BUG: IEnumUnknown Is Not Remoted

PSS ID Number: Q110799
Authored 28-Jan-1994 Last modified 21-Apr-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

SYMPTOMS

All calls to IOleContainer::EnumObjects() fail.

CAUSE

IOleContainer::EnumObjects() returns a pointer to IEnumUnknown. The current implementation of OLE does not include remoting code for the IEnumUnknown interface. There is no proxy and stub code, therefore the methods in this interface cannot be marshaled to another process.

STATUS

Microsoft has confirmed this to be a problem in the OLE Libraries version 2.01. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: 2.01 buglist2.01
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoCdt

BUG: Insert Object from Zero Length File Causes GP Fault

PSS ID Number: Q108371
Authored 09-Dec-1993 Last modified 15-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

SYMPTOMS

Creating an object from a zero length file that does not have a valid file class extension results in a general protection (GP) fault after displaying the message, "Not enough memory to perform this operation. Close one or more applications and try again."

CAUSE

When creating an object from a file that does not have a valid class extension, the Object Packager is invoked to generate the embedded object. In the case of a zero length source file, the Packager document-level DDE window is rendered invalid before a terminate message can be sent to it.

STATUS

Microsoft has confirmed this to be a problem in the Object Linking and Embedding (OLE) libraries version 2.01.

While the operation still fails with Object Packager 3.5 under Windows NT 3.5, the Object Packager provides a better error message.

Additional reference words: 2.01 gpf gp fault
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoArc

BUG: Invisible MSDRAW Object Retains Keyboard Focus

PSS ID Number: Q110715
Authored 27-Jan-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft OLE version 1.0
- Microsoft OLE Libraries for Windows and Win32s, versions 2.0 and 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, version 3.5

SYMPTOMS

When an invisible MSDRAW object is inserted as an OLE embedded object, the object retains the keyboard focus. This behaviour is incorrect. The MSDRAW object should not take the focus until it has been made visible.

This problem occurs whether the MSDRAW object is inserted into an OLE 1.0 client application or into an OLE 2.0 container application.

STATUS

Microsoft has confirmed this to be a problem with MSDRAW version 1.0. We are researching this problem and will post new information here as it becomes available.

Additional reference words: 1.00 2.00 3.50
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoApp

BUG: IOleCache::Cache Returns Incorrect Error Value

PSS ID Number: Q110488
Authored 24-Jan-1994 Last modified 24-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

An OLE 2.0 container application calls IOleCache::Cache for a particular FORMATETC and is returned the value OLE_S_FORMATETC_NOTSUPPORTED. A subsequent call to IOleCache::Cache with the same FORMATETC returns CACHE_S_SAMECACHE.

CAUSE

The OLE 2.0 server application has a message filter installed and is rejecting all incoming calls. The default object handler cannot get the information from the server to fill the cache, but the cache is still created. A more appropriate return value would be OLE_S_SOMECACHES_NOTUPDATED.

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: 2.01 3.50 4.00 95
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoPrs

BUG: IOleCache::Cache(), ADVF_DATAONSTOP, and OLE 1.0 Objects

PSS ID Number: Q111614
Authored 14-Feb-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

An OLE 2.0 container application document contains an embedded OLE 1.0 object. The container calls IOleCache::Cache() to control the cached presentation data for the object, and specifies ADVF_DATAONSTOP as the advise flag to Cache(). The user makes some changes to the object in the OLE 1.0 server, then attempts to update the object's presentation in the server by choosing the Update command from the File menu. Finally, the user closes the object server.

In this specific scenario, the object's presentation in the server is not updated. The object's native data, however, is correct.

CAUSE

When the user selects Update, the presentation for the object does not go across to the cache or the container, because the container specified ADVF_DATAONSTOP as the cache option. As part of the update operation, the server internally marks the object as "not dirty". Because the object is not dirty, when the server is subsequently closed, it does not send any data to the container or the cache.

RESOLUTION

Container applications should not specify the ADVF_DATAONSTOP flag when calling IOleCache::Cache(). Instead, they should specify ADVFCACHE_ONSAVE. When ADVFCACHE_ONSAVE is used, the OLE 1.0 object's cached presentation data will be updated correctly.

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here as it becomes available.

Additional reference words: 2.01 3.50 4.00
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoPrs

BUG: IOleObject::Close(OLECLOSE_NOSAVE) and DoVerb()

PSS ID Number: Q111577
Authored 14-Feb-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

An OLE 2.0 container application inserts a new embedded object into a document. The container calls IOleObject::Update() on the object to update the cache, then calls IOleObject::Close(OLECLOSE_NOSAVE) to transition the object back to the running state. Finally, the container tries to rerun the object by calling IOleObject::DoVerb().

In this particular scenario, DoVerb() will return STG_E_FILENOTFOUND, and the object will not be rerun.

CAUSE

Because the container specifies OLECLOSE_NOSAVE when calling Close(), the object is never saved to persistent storage. However, the object has internally been marked as not being a brand new object. Consequently, when the container calls DoVerb(), the object's handler calls IPersistStorage::Load() on the object, attempting to put the object into the running state. This call will fail, because the object has never been saved.

RESOLUTION

The solution to this problem is for the container to specify OLECLOSE_SAVEIFDIRTY when calling Close().

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: 2.10 3.50 4.00
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoCdt

BUG: IROT::Register() and IOL::SetDisplayName() Inconsistency

PSS ID Number: Q111607
Authored 14-Feb-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

IRunningObjectTable::Register() allows monikers that are not valid filenames to be registered as file monikers. However, if the display name string passed to IOleLink::SetSourceDisplayName() is not a valid filename, SetSourceDisplayName() returns STG_E_FILENOTFOUND.

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

MORE INFORMATION

This inconsistency may become apparent when an OLE application is calling IOleLink::SetSourceDisplayName() to change the source of a link object. It is possible that a new file moniker that is not a valid source name will be registered on the running object table--for example, a moniker of "New 1" for a new, as yet unsaved document. It will not be possible to use SetSourceDisplayName() to link to this object, however, because "New 1" is not a valid filename, and therefore it will be rejected by SetSourceDisplayName().

Additional reference words: 2.01 3.10 3.50 4.00
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoLnk

BUG: Object Packager GPFs w/ Paths Greater Than 64 Characters

PSS ID Number: Q109116
Authored 21-Dec-1993 Last modified 16-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

SYMPTOMS

Packaging an object that has a fully qualified path of more than 64 letters results in a general protection (GP) fault in the Object Packager.

CAUSE

MS-DOS paths have a limit of 64 characters. The Object Packager does not correctly handle scenarios that involve File Manager paths greater than this limit.

STATUS

Microsoft has confirmed this to be a problem in the Windows 3.1 Object Packager.

This problem does not occur with Object Packager 3.5 on Windows NT 3.5.

MORE INFORMATION

Steps to Reproduce
---

1. Create a directory structure in File Manager that results in a path greater than 64 characters. You must use File Manager because MS-DOS will not allow a directory structure this large to be created.

2. Use File Manager to copy a file into this new subdirectory. File Manager must be used because MS-DOS will not recognize this directory.

3. Select the file from within File Manager.

4. Choose Copy from the File menu.

5. Choose the Copy To Clipboard button to place the filename and path on the clipboard.

6. Launch CntOutl and Paste Link.

Additional reference words: 2.10 3.10 GPF GP-Fault buglist2.01
KBCategory: kbole kbbuglist

KBSubcategory: LeTwoApp

BUG: OLE 1.0 Server Launched for Paste Link

PSS ID Number: Q111578
Authored 14-Feb-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

An OLE 1.0 server application copies an object to the clipboard, then enters the blocked state. An OLE 2.0 container application then attempts to paste a link to the object from the clipboard. The OLE 2.0 application is frozen.

Next, the OLE 1.0 application is shut down. This unfreezes the OLE 2.0 container application, which proceeds to paste the link to the OLE 1.0 object. However, an instance of the OLE 1.0 server is launched. Normally, the server is not run in a paste-link scenario.

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: 3.10 3.50 4.00
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoArc

BUG: OLE 2.0 Compatibility Layer Uses Document IDataObject

PSS ID Number: Q111585
Authored 14-Feb-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

Whenever the OLE 1.0 client compatibility layer uses IDataObject, it uses the OLE 2.0 server's document-level data object rather than the server's pseudo-object data object.

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

MORE INFORMATION

One of the side-effects of this problem is that the following key is required in the OLE 2.0 server application's CLSID listing in the registration database

```
GetSet
   |- X = 3, 1, 32, 1
```

where X is application-dependent. If this key is missing from the registration database, then any attempt to embed the OLE 2.0 object into an OLE 1.0 client will fail, even if the server application fully supports IDataObject::EnumFormatEtc.

The GetSet key is used by server applications to describe the formats that an object is capable of rendering, or having set. OLE looks at this key when it needs to know which formats an object supports, without having to run the object. This key is required in this situation because the OLE 1.0 compatibility layer is always going to refer to the registration database to enumerate the formats supported by the server application. This particular key represents the METAFILEPICT format, which must be supported for OLE 1.0 compatibility.

Additional reference words: 2.01 3.50 4.00 toolkit
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoPrs

BUG: OLE 2.0 Containers & 1.0 Objects that Close w/out Saving

PSS ID Number: Q109547
Authored 04-Jan-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE version 1.0
- Microsoft OLE Libraries for Windows and Win32s, versions 2.0 and 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

When an OLE 1.0 object is inserted into an OLE 2.0 container document and then closed without an update being invoked, the correct streams for that object are not written to storage. Any subsequent attempt by the container to load the object will fail.

CAUSE

When the OLE 1.0 object is closed without an update, the OLE 2.0 container receives an IAdviseSink::OnClose() message without having first received an IClientSite::SaveObject() message. Unless the container takes steps to work around this problem, incorrect streams will be saved.

RESOLUTION

To protect against this scenario, OLE 2.0 containers that want data to be available after an OLE 1.0 object closes without updating can implement code similar to the following pseudocode:

```
   OleCreate();
   OleRun();
   IOleObject::Update();       \\ To get a snapshot of the data.
   OleSave();                  \\ To save the data to storage.
   IOleObject::DoVerb();
```

Alternatively, the container could just delete any objects that call IAdviseSink::OnClose() without first calling IClientSite::SaveObject().

STATUS

Microsoft has confirmed this to be a problem in the products listed at the beginning of this article. We are researching this problem and will post new information here as it becomes available.

MORE INFORMATION

For more information about known idiosyncrasies of embedding or linking OLE 1.0 objects into OLE 2.0 containers, please see the "Compatibility with OLE 1.0" section of the OLE SDK Help file.

Additional reference words: 1.00 2.00 2.01 3.50 4.00 toolkit
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoArc

BUG: OLE 2.0 Does Not Support CF_OWNERDISPLAY

PSS ID Number: Q109552
Authored 04-Jan-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

If an application places a data transfer object onto the clipboard and that data object enumerates CF_OWNERDISPLAY as one of the clipboard formats, the application will not receive the WM_PAINTCLIPBOARD message.

CAUSE

The WM_PAINTCLIPBOARD message is sent to the clipboard owner in order to paint the clipboard viewer window. After performing an OleSetClipboard, clipboard ownership belongs to the OLE libraries. Because no window handle information is passed with OleSetClipboard, the OLE libraries cannot forward this message to the calling application.

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here as it becomes available.

Additional reference words: 2.01 3.50 4.00
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoDdc

BUG: OLE Type Emulation for Previously Loaded Objects

PSS ID Number: Q111608
Authored 14-Feb-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

OLE type emulation is the process that allows the application user to specify that all objects of some particular type are henceforth to be activated as objects of some alternate, emulating type. When objects of the original type are subsequently run, the server for the emulating type is launched to serve them.

However, this emulation does not occur for objects of the original type that were already in the loaded state when the type emulation occurred. When such objects are subsequently run, they are run as the original type, not the emulating type. The original server is launched, not the server for the emulating type.

CAUSE

When an object is in the loaded state, its object handler is running. The handler is, in general, particular to the original type for that object, and therefore cannot emulate another object type.

STATUS

Microsoft has confirmed this to be a problem with the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

MORE INFORMATION

If a container application has previously loaded instances of the original type, it should reload each such object as the alternative, emulating type the next time that object is run.

For more information on OLE type emulation, see "Emulating Different Object Types" in the Object Linking and Embedding Software Development Kit (SDK) version 2.01 "Programmer's Reference" help file.

Additional reference words: 2.01 3.50 4.00 95

KBCategory: kbole kbbuglist
KBSubcategory: LeTwoUim

BUG: OleConvertStorageToOLESTREAM() Fails When CLSID Is NULL

PSS ID Number: Q111611
Authored 14-Feb-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

The OleConvertStorageToOLESTREAM() API (application programming interface) call is used to convert the storage of an embedded object from the OLE 2.0 storage model to the OLE 1.0 storage model. However, if the OLE 2.0 object has a CLSID of NULL, then OleConvertStorageToOLESTREAM() fails with a return code of OLE_E_CLASS.

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

MORE INFORMATION

Note that it is unusual for an embedded object to have a CLSID of NULL. However, after IOleLink::SetSourceDisplayName() has been called to change the source of an embedded link object, it is possible for that link object to have a CLSID of NULL.

Additional reference words: 2.01 3.50 4.00
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoPst

BUG: OleCreate and IOleCache::Cache Fail with Multiple TYMEDs

PSS ID Number: Q109543
Authored 04-Jan-1994 Last modified 16-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

SYMPTOMS

The OLE 2.01 creation functions [OleCreate(), OleCreateFromData(), and so forth] and IOleCache::Cache() fail if multiple values are specified for the TYMED field of the FORMATETC parameter. The functions fail even if the object server supports at least one of the TYMED values specified.

STATUS

Microsoft has confirmed this to be a problem in version 2.01 of the OLE libraries. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

This is not a problem in 32-bit OLE.

Additional reference words: 2.01
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoCom

BUG: OLERENDER_ASIS Results in Blank Embedded Object

PSS ID Number: Q110714
Authored 27-Jan-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

If an OLE 1.0 client application contains an embedded object and copies the object to the clipboard and an OLE 2.0 container application then performs a paste operation by creating a new embedded object based on the clipboard data by calling OleGetClipboard() and then calling OleCreateFromData(), the resulting embedded object appears blank in the container's document if the OLE 2.0 container specifies OLERENDER_ASIS as the renderOpt parameter to OleCreateFromData().

CAUSE

The OLE 1.0 compatibility layer has placed the CF_EMBEDSOURCE format onto the clipboard on behalf of the OLE 1.0 client application. This is incorrect. When copying an embedded object to the clipboard, the CF_EMBEDDEDOBJECT format should be supplied.

NOTE: The DOBJVIEW test application supplied with the OLE Software Development Kit (SDK) version 2.01 can be used to determine which clipboard formats are supplied by the clipboard data object.

RESOLUTION

An OLE 2.0 container application can avoid specifying OLERENDER_ASIS as the renderOpt parameter to OleCreateFromData() when the clipboard data object does not supply the CF_EMBEDDEDOBJECT format.

STATUS

Microsoft has confirmed this to be a problem in the products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: 2.01 3.50 4.00
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoCdt

BUG: Paste Link Disabled Across the Network

PSS ID Number: Q108939
Authored 20-Dec-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

The OleQueryLinkFromData application programming interface (API) function returns failure when clipbook data from another machine is copied to the local clipboard, even if the clipbook contains all of the data needed to create a link.

CAUSE

cfLinkSource is not being rendered by the OLE libraries when data is being transferred via NetDDE.

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here as it becomes available.

Reference words: 2.01 3.50 4.00 Paste Link
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoDdc

BUG: Paste-Linking a 256-Color Paintbrush Object

PSS ID Number: Q111612
Authored 14-Feb-1994 Last modified 16-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries for Windows NT, version 2.1

SYMPTOMS

A 256-color bitmap is loaded into Microsoft Paintbrush version 3.11 (included with Windows 3.11) and 3.50 (included with Windows NT 3.5). The image is selected and then copied to the clipboard. An OLE 2.0 application then pastes a link to the object from the clipboard. To perform the paste-link operation, the container calls OleCreateLinkFromData(), specifying OLERENDER_FORMAT as the rendering option and CF_DIB as the desired format.

The resulting link object in the OLE 2.0 application has an incorrect appearance, such as being colored solid black.

STATUS

Microsoft has confirmed this to be a problem with the Object Linking and Embedding libraries version 2.01. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

MORE INFORMATION

Normally an OLE 2.0 application will specify OLERENDER_DRAW as the rendering option. This allows OLE to cache a metafile presentation for the object. When a metafile is cached, the 256-color Paintbrush object is displayed correctly.

Additional reference words: 2.00 2.01 2.10 3.50
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoApp

BUG: Printing Does Not Work from CL2TEST.EXE

PSS ID Number: Q112413
Authored 08-Mar-1994 Last modified 16-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft Visual C++ 32-bit Edition, versions 2.0 and 2.1
- Microsoft Win32 Software Development Kit (SDK), version 3.5

SYMPTOMS

Printing does not work correctly from CL2TEST.EXE or CL32TEST.EXE. Choosing Print from the File menu results in a blank printout.

CAUSE

Printing has not been correctly implemented in the test container application.

STATUS

Microsoft has confirmed this to be a problem in the CL2TEST test application included with the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: 2.00 2.01 2.10 3.50
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoTls

BUG: Relative Monikers and OLE 1.0 Link Objects

PSS ID Number: Q111609
Authored 14-Feb-1994 Last modified 21-Apr-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

SYMPTOMS

When a link object from an OLE 2.0 application is copied to the clipboard and then pasted into an OLE 1.0 application, the OLE 2.0 emulation layer preserves the object's relative moniker. If the OLE 1.0 application document is closed and then reopened by an OLE 2.0 version of the application, OLE 2.0 will create a full moniker for the linked object by appending the preserved relative moniker to the file moniker for the document. This full moniker will be incorrect, because it will point to the new document rather than to the first OLE 2.0 application's document that contains the original linked object.

However, OLE 2.0 will still be able to bind to the linked object by following the absolute moniker that is also stored with the object (providing that the absolute location of the original linked object has not changed). At the time of this binding, OLE will then correct the full moniker for the object.

STATUS

Microsoft has confirmed this to be a problem with the Object Linking and Embedding libraries version 2.01. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: 2.10 buglist2.01
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoArc

BUG: Retaining Clipboard IDataObject Causes Unexpected Result

PSS ID Number: Q109545
Authored 04-Jan-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

An application holding on to a clipboard IDataObject pointer, and making repeated calls to IDataObject::EnumFormatEtc() through that pointer, may find that the set of formats returned by EnumFormatEtc() changes between calls.

RESOLUTION

As noted in the Object Linking and Embedding SDK version 2.01 documentation, an application that calls OleGetClipboard() to retrieve an IDataObject interface pointer should hold on to that IDataObject pointer only for a very short time.

STATUS

Microsoft has confirmed this to be a problem in the products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

MORE INFORMATION

Steps to Reproduce
---

1. Copy an object from an OLE 1.0 server to the clipboard.

2. Call OleGetClipboard() to obtain an IDataObject pointer to the clipboard object.

3. Call IDataObject::EnumFormatEtc() to enumerate the available formats.

4. Without calling IDataObject::Release(), copy an object from an OLE 2.0 server to the clipboard.

5. Call IDataObject::EnumFormatEtc() again. The formats enumerated have changed to the formats provided by the new OLE 2.0 object.

The OLE2UI library's implementation of the Paste Special dialog box (provided in the Object Linking and Embedding SDK version 2.01) works around this problem. The Paste Special dialog box is launched by calling OleUIPasteSpecial(). If the user changes the contents of the clipboard while this dialog is up, the OLE2UI code detects the change and responds by ending the dialog box. When this happens, OleUIPasteSpecial() returns OLEUI_PSERR_CLIPBOARDCHANGED.

The dialog box detects changes to the clipboard contents by calling SetClipboardViewer() to splice itself into the clipboard viewer chain.

Additional reference words: 2.01 3.50 4.00 toolkit
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoDdc

BUG: Set Line Height on ISvrOutl Object Causes GPF in Cl2Test

PSS ID Number: Q108930
Authored 20-Dec-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

In certain circumstances, editing the Line Height of an ISvrOutl object embedded within Cl2Test results in a general protection (GP) fault after displaying the following message:

> GDI - An error has occurred in your application. If you choose Ignore, you should save your work in a new file. If you choose Close, your application will terminate.

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here as it becomes available.

MORE INFORMATION

Steps to Reproduce
---

1. Launch ISvrOutl, create a line item.
2. Edit/Copy the line.
3. Launch CL2TEST.EXE.
4. Edit/Paste.
5. Edit/Outline/Open or double-click the object.
6. Within ISvrOutl, Line/Set Line Height.
7. Enter 1 in the height field, choose OK.

Additional reference words: 2.01 3.50 4.00 In Place Inplace sample gpf gp-fault
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoTls

BUG: SR2TEST Menu Items Enabled Incorrectly

PSS ID Number: Q109546
Authored 04-Jan-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

During an in-place activation session, SR2TEST sometimes incorrectly enables menu items on its Edit menu.

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here as it becomes available.

MORE INFORMATION

Steps to Reproduce Problem
---

1. Start SR2TEST.

2. Start CL2TEST.

3. Create a new CL2TEST document.

4. Insert an SR2TEST object in-place into the CL2TEST document.

5. While the SR2TEST object is still in-place active within CL2TEST, choose Delete from the Edit menu to delete the shape.

At this point, the Cut, Copy, and Delete menu items on the Edit menu are still enabled. This is incorrect because there is no shape remaining in the SR2TEST object to cut, copy, or delete.

Additional reference words: 2.01 3.50 4.00 toolkit
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoTls

BUG: SR2TEST Won't Close After Editing Link Object

PSS ID Number: Q108935
Authored 20-Dec-1993 Last modified 24-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

SYMPTOMS

After editing a linked SR2TEST object from CL2TEST, SR2TEST fails to shut down after first displaying the following message box:

Error Document not unlocked. Attempt Self-unlocking?

CAUSE

The BindToSource button from the Properties dialog box was chosen in CL2TEST before running the object. CL2TEST is not properly releasing the bind context, locking SR2TEST open.

STATUS

Microsoft has confirmed this to be a problem in the CL2TEST test application shipped with the OLE Toolkit version 2.01. We are researching this problem and will post new information here as it becomes available.

This is not a problem with CL32TEST included with the 32-bit OLE libraries.

Additional reference words: 2.01
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoTls

BUG: Status Bar Not Redrawn With SR2TEST When In-Place

PSS ID Number: Q109541
Authored 04-Jan-1994 Last modified 24-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

SYMPTOMS

After inserting an SR2TEST object into the in-place version of the OUTLINE sample application, the status bar will not be redrawn until the container application's window is resized.

STATUS

Microsoft has confirmed this to be a problem in the SR2TEST sample application included with the OLE SDK version 2.01. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

This is not a problem with SR32TEST included with the 32-bit OLE libraries.

MORE INFORMATION

Steps to Reproduce
---

1. Run ICNTOUTL.EXE.
2. Choose Insert Object from Edit menu.
3. Choose OleTest SrTest 2.0 Shape from the list box.
4. Choose OK.
5. Resize the container window.

Additional reference words: 2.01 In Place Inplace
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoTls

BUG: SVROUTL Link Not Displayed Correctly in CNTROUTL

PSS ID Number: Q110871
Authored 01-Feb-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

Selecting the Paste Link menu item in the CNTROUTL sample application does not correctly display the contents of the link if the source application is the SVROUTL sample application.

This problem occurs only if the information created in SVROUTL is edited further after the Edit Copy operation in SVROUTL, but before the Paste Link operation in CNTROUTL. In this situation, the resulting link object is displayed as originally copied to the clipboard, and does not reflect the changes made after the copy operation.

STATUS

Microsoft has confirmed this to be a problem with the CNTROUTL and SVROUTL samples included with the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here as it becomes available.

MORE INFORMATION

This problem occurs only when the SVROUTL link object is first pasted into CNTROUTL. After the initial paste operation, any further modifications to the information in SVROUTL are correctly reflected in the display of the linked object in CNTROUTL.

Additional reference words: 2.01 3.50 4.00
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoTls

BUG: Windows OLE DLLs Do Not Convert Mac OLESTREAM

PSS ID Number: Q112412
Authored 08-Mar-1994 Last modified 21-Apr-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

SYMPTOMS

The OLE 2.01 libraries will not convert a Mac OLESTREAM to a Windows OLESTREAM. Similarly, the Windows OLE dynamic-link libraries (DLLs) will not convert a Mac IStorage to a Windows OLESTREAM.

STATUS

Microsoft has confirmed this to be a problem in the OLE SDK version 2.01. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

MORE INFORMATION

The Macintosh OLE 2.0 libraries will be able to convert non-Macintosh OLESTREAMs by means of OleConvertOLESTREAMToIStorage. At this time, there are no known Macintosh OLE 1.0 applications that generate Windows-compatible files that contain Mac OLESTREAMs.

Additional reference words: 2.01 buglist2.01
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoArc

BUG:IOleObject::IsUpToDate Returns Wrong Value for Manual

PSS ID Number: Q111655
Authored 15-Feb-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

To determine whether an OLE link needs to be updated, an OLE 2.0 container application needs to call IOleObject::IsUpToDate. If this method returns S_FALSE, the link needs to be updated; otherwise, the link can be considered up to date. However, calling IOleObject::IsUpToDate on a manual link while the link server is not running results in an a return value of S_FALSE, even if the link is current. The result of this is that OLE 2.0 container applications may update links unnecessarily.

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: 2.01 3.50 4.00
KBCategory: kbole kbbuglist
KBSubcategory: LeTwoLnk

CLEANDB.EXE Not Included in OLE 2.01 SDK

PSS ID Number: Q109431
Authored 04-Jan-1994 Last modified 21-Apr-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

The Registration Database Cleaner (CLEANDB.EXE), which shipped with the OLE 2.0 SDK, is no longer shipped in version 2.01 of the OLE SDK.

CLEANDB.EXE was used to remove keys from the registration database that were added by pre-release versions of OLE 2.0 but not used in the final version. Because these keys are obsolete, CLEANDB.EXE was removed from the OLE SDK in version 2.01.

Additional reference words: 2.01
KBCategory: kbole kbprg
KBSubcategory: LeTwoTls

Containers Should Not Query for IOleInPlaceActiveObject

PSS ID Number: Q98678
Authored 11-May-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.0
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

OLE (object linking and embedding) version 2.0 container applications should never call QueryInterface to get a pointer to the IOleInPlaceActiveObject interface. Instead, the container should AddRef() the pointer passed to IOleInPlaceUIWindow::SetActiveObject and save it until needed.

Consider the scenario in which a nested object is currently inplace active; that is, the container contains an object and this contained object in turn holds another object that is inplace active. The container has a pointer only to the "outer" object, and is not aware of the "inner" object. Calling QueryInterface to get a pointer to IOleInPlaceActiveObject is meaningless on the outer object because the outer object is not currently inplace active. The only reliable way to get a pointer to the correct IOleInPlaceActiveObject interface is through IOleInPlaceUIWindow::SetActiveObject.

Objects capable of inplace activation should guard against this case, and never return IOleInPlaceActiveObject from QueryInterface.

Additional reference words: 2.00 3.50 4.00
KBCategory: kbole kbprg
KBSubcategory: LeTwoInp

Corrections for Inside OLE 2 Sample Code

PSS ID Number: Q113255
Authored 29-Mar-1994 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, version 2.1, included with:
  - Microsoft Windows NT, version 3.5

SUMMARY

The sample code for the book "Inside OLE 2," by Kraig Brockschmidt (Microsoft Press) was originally written to be compatible with OLE 2.0. This code has been updated to be compatible with OLE 2.01, and also to fix some problems in the original code. This article describes the changes that were made in updating the code. The changes are described in detail in the "More Information" section of this article below; below a quick list of the problems that required changes:

- Compiler Errors on IViewObject::Draw()
- Compiler Errors on OleUIAddVerbMenu()
- Compiler Errors on OleStdGetObjectDescriptorFromOleObject()
- Compiler Errors on IDS_CLOSE
- Linker Fails with CLASSLIB Sample
- Compiler Errors Referencing GETICON.H
- Incorrect Prototypes for LibMain() and WEP()
- Patron GP Faults During Activate As
- Patron's Convert To Fails
- CLASSLIB String Allocations Fail with Some Compilers
- CImpIPolyline::WriteToFile() Returns an Incorrect Value
- CImpIOleObject::GetClientSite() Returns an Incorrect Value
- Polyline Registers a Format with an Uninitialized String
- DataTran Reference Counting Problem
- Component Cosmo Fails to Release Polyline Object
- Cosmo Fails to Load When Used with HCosmo Handler
- Patron Does Not Work with Some Printer Drivers
- ClassLib Creates Table of Invalid String Pointers
- Errors in Enumerator Samples
- Samples GP Fault When Closing Iconized Documents

Updated Sample Code Available

The INOLE2 Software Library sample contains a complete source tree for updated "Inside OLE 2" sample code. All of the changes described in this article have already been made to the INOLE2 code.

The easiest way to update all of the "Inside OLE 2" samples is to get a completely new source tree from INOLE2, and then rebuild the samples from scratch under OLE 2.01. This article can then be used as a reference, explaining what changes were made and why there were necessary.

INOLE2 can be downloaded as a self-extracting file from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL and download INOLE2.EXE

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download INOLE2.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \softlib\mslfiles directory
  Get INOLE2.EXE

NOTE: This archive contains subdirectories, be sure to use the -d command line switch during extraction.

MORE INFORMATION

BOOKUI.DLL

If the original source code to the Inside OLE 2 samples is updated according to this article, and the samples are then rebuilt to be compatible with OLE 2.01, it will also be necessary to rebuild an OLE 2.01 version of the OLE2UI library for the samples to use. To do so, build a library called BOOKUI.DLL according to the instructions in the OLE 2.01 SDK.

Use this new DLL to replace the INOLE2\BUILD\BOOKUI.DLL file included with the book.

If the samples are rebuilt from the INOLE2 Software Library sample, it is not necessary to manually rebuild BOOKUI.DLL, because INOLE2 already includes an updated version of BOOKUI.DLL.

1. Compiler Errors on IViewObject::Draw()

Problem:

The prototype for IViewObject::Draw() changed between OLE 2.0 and OLE 2.01. In 2.0 the lprcBounds and lprcWBounds parameters were prototyped as "const LPRECTL"; in 2.01 they are prototyped as "LPCRECTL". This change causes the HCosmo (Chapter 11) and Polyline (Chapters 11 and 16) sample applications to fail during compilation under 2.01.

Solution:

To correct this error, update all references to IViewObject::Draw() in the Inside OLE 2 sample code by changing const LPRECTL to LPCRECTL.

This change will be necessary in the following files:

   interfac\iviewobj.cpp
   interfac\iviewobj.h

   chap11\hcosmo\hcosmo.h
   chap11\hcosmo\iviewobj.cpp

   chap11\polyline\iviewobj.cpp
   chap11\polyline\polyline.h

   chap16\polyline\iviewobj.cpp
   chap16\polyline\polyline.h

2. Compiler Errors on OleUIAddVerbMenu()

Problem:

The prototype for OleUIAddVerbMenu() changed between OLE 2.0 and OLE 2.01. The OLE2UI library version OLE 2.01 added an additional parameter to OleUIAddVerbMenu(). This new parameter is an integer value called "idVerbMax", and it indicates the largest number the container allows for a verb menu item identifier. The idVerbMax parameter immediately follows the idVerbMin parameter.

The Patron sample applications were written for the OLE 2.0 version of the user interface library, which did not include this new parameter. Consequently, these samples do not compile with the OLE 2.01 header files. This difference affects the versions of Patron found in Chapters 9, 12, 13, 14, and 15.

Solution:

To correct this problem, perform the following two steps:

1. Build an OLE 2.01 version of BOOKUI.DLL, as described in problem 1 above.

2. Update all calls to OleUIAddVerbMenu() in Patron by passing "ID_VERBMAX" (already defined in RESOURCE.H) as the idVerbMax parameter. For example, in the function CPage::FQueryObjectSelected(), change the following

```
OleUIAddVerbMenu(NULL, NULL, hMenu, MENUPOS_OBJECT
    , IDM_VERBMIN, FALSE, 0, &hMenuTemp);
```

   to read as follows:

```
OleUIAddVerbMenu(NULL, NULL, hMenu, MENUPOS_OBJECT
    , IDM_VERBMIN, IDM_VERBMAX, FALSE, 0, &hMenuTemp);
```

   This change will be necessary in the functions CPage::FQueryObjectSelected() and CTenant::AddVerbMenu() in each of the following files:

```
chap09\patron\page.cpp
chap09\patron\tenant.cpp

chap12\patron\page.cpp
chap12\patron\tenant.cpp

chap13\patron\page.cpp
chap13\patron\tenant.cpp

chap14\patron\page.cpp
chap14\patron\tenant.cpp

chap15\patron\page.cpp
chap15\patron\tenant.cpp
```

3. Compiler Errors on OleStdGetObjectDescriptorFromOleObject()

Problem:

The prototype for OleStdGetObjectDescriptorFromOleObject() changed between OLE 2.0 and OLE 2.01. Version 2.01 of the OLE2UI library added an additional parameter to OleStdGetObjectDescriptorFromOleObject(). This new parameter, "lpSizelHim", is an LPSIZEL value and points to a structure that indicates the dimensions of the object. The lpSizelHim parameter was added to the end of the existing parameter list.

The Patron sample applications were written for the OLE 2.0 version of the user interface library, which did not include this new parameter. Consequently, these samples do not compile with the OLE 2.01 header files. This difference affects the versions of Patron in Chapters 9, 12, 13, 14, and 15.

Solution:

To correct this problem, perform the following two steps:

1. Build an OLE 2.01 version of BOOKUI.DLL, as described in problem 1 above.

2a. To quickly solve the problem, update all calls to OleStdGetObjectDescriptorFromOleObject() in Patron by passing NULL as the lpSizelHim parameter.

2b. For a more complete solution to the problem, first declare a local variable:

```
SIZEL   szl;
```

Then replace the code:

```
stm.hGlobal=OleStdGetObjectDescriptorDataFromOleObject
    (m_pIOleObject, NULL, m_fe.dwAspect, ptl);
```

with the code:

```
        SETSIZEL(szl, (10*(m_rcl.right-m_rcl.left))
            , (10 * (m_rcl.bottom-m_rcl.top)));

        stm.hGlobal=OleStdGetObjectDescriptorDataFromOleObject
            (m_pIOleObject, NULL, m_fe.dwAspect, ptl, &szl);
```

This code correctly computes the size of the object and then stores those extents into the object descriptor.

These changes will be necessary in the function CTenant::CopyEmbeddedObject() in each of the following files:

```
chap09\patron\tenant.cpp
chap12\patron\tenant.cpp
chap13\patron\tenant.cpp
chap14\patron\tenant.cpp
chap15\patron\tenant.cpp
```

These changes will also be necessary in the function CTenant::CopyLinkedObject(), found in TENANT.CPP in Chapters 12, 13, 14, and 15.

4. Compiler Errors on IDS_CLOSE

Problem:

The OLE2UI.H header file shipped with OLE 2.01 defines a symbol IDS_CLOSE. This symbol causes a conflict with a symbol of the same name defined in the Cosmo samples.

This conflict affects the versions of Cosmo in Chapters 10, 13, 14, and 16. The files affected are COSMO.RC and RESOURCE.H.

Solution:

To correct the problem, rename IDS_CLOSE in RESOURCE.H to IDS_CLOSE2, and change the single occurrence of IDS_CLOSE in COSMO.RC to IDS_CLOSE2 in each chapter. These changes are necessary in the following files:

```
chap10\cosmo\cosmo.rc
chap10\cosmo\resource.h

chap13\cosmo\cosmo.rc
chap13\cosmo\resource.h

chap14\cosmo\cosmo.rc
chap14\cosmo\resource.h

chap16\cosmo\cosmo.rc
chap16\cosmo\resource.h
```

5. Linker Fails with CLASSLIB Sample

Problem:

There is an error in the FILES.LST file in the CLASSLIB directory of the sample code. The first entry in this file, "cstrtable.obj", is a valid long filename under Windows NT, but the file created during compilation is actually "cstrtabl.obj" (no "e" on table). This causes no problems with a 16-bit compiler, because the extra "e" is ignored. However, it will cause an error with a 32-bit compiler.

Solution:

To solve the problem, remove the "e", thus changing "cstrtable.obj" to "cstrtabl.obj".

6. Compiler Errors Referencing GETICON.H

Problem:

In OLE 2.0, container applications must include the file GETICON.H, which is supplied with the OLE Software Development Kit (SDK) version 2.0. In OLE 2.01, applications no longer need to include GETICON.H, because the information in that file has been moved to other header files. Because it is no longer needed, GETICON.H is not included with the OLE 2.01 SDK.

The Patron sample applications of Chapters 14 and 15 were written for OLE 2.0, so they include GETICON.H. Because this file is no longer available, these versions of Patron fail to compile with the OLE 2.01 SDK.

Solution:

To correct the problem, comment out or delete the line "#include <geticon.h>" in TENANT.CPP. This change is necessary in the following files:

   chap14\patron\tenant.cpp
   chap15\patron\tenant.cpp

7. Incorrect Prototypes for LibMain() and WEP()

Problem:

The LibMain() and WEP() functions in all of the DLL samples are prototyped incorrectly. These prototypes cause errors when using some compilers (for example, Borland C++ 4.0); they do not cause problems with other compilers (for example, Microsoft Visual C++ versions 1.0 and 1.5).

Solution:

The DLL samples in Inside OLE 2 implement the LibMain() and WEP() functions as follows:

```
HANDLE FAR PASCAL LibMain(HANDLE hInstance, WORD wDataSeg
    , WORD cbHeapSize, LPSTR lpCmdLine)
    {
    ...

    return hInstance;
    }

void FAR PASCAL WEP(int bSystemExit)
    {
    return;
    }
```

To match the Windows SDK specifications, both of these functions should return an int:

```
int FAR PASCAL LibMain(HANDLE hInstance, WORD wDataSeg
    , WORD cbHeapSize, LPSTR lpCmdLine)
    {
    ...

    return (int)hInstance;
    }

int FAR PASCAL WEP(int bSystemExit)
    {
    return 0;
    }
```

To solve this problem, make the changes above to all occurrences of LibMain() and WEP() in the Inside OLE 2 sample code.

8. Patron GP Faults During Activate As

Summary:

The Patron sample applications from Chapters 14 and 15 support the OLE 2 Convert dialog box. Choosing Activate As from this dialog box sometimes causes a GP fault.

Problem:

The problem lies in the way that the CTenant::Close() function uses the fReopen flag. When Close() determines that there are no references to the tenant IStorage, it normally resets the internal state of the tenant to default values. As part of this process, it sets the member m_pIStorage to NULL. However, if fReopen is TRUE, this assignment to NULL is skipped.

This can lead to situations where m_pIStorage is non-NULL but the storage itself has been destroyed. Thus m_pIStorage is an invalid pointer, and trying to call through it causes a GP fault.

To correct this problem, ensure that the m_pIStorage variable is set to NULL any time its reference count is 0 (zero). This correction involves

removing a condition in the CTenant::Close() function, which is found in the Patron source file TENANT.CPP.

Solution:

To correct this problem, edit the code to CTenant::Close() as follows:

Where you see the lines

```
   if (!fReopen)
       m_pIStorage=NULL;
```

remove the "if" statement, leaving:

```
   m_pIStorage=NULL;
```

The fReopen flag was used in an attempt to provide some optimization when performing Activate As, but this flag is not necessary. The updated Inside OLE 2 sample code referenced above has removed the flag entirely. This affects the TENANT.CPP, TENANT.H, and PAGE.CPP source files of the PATRON samples in Chapters 14 and 15.

9. Patron's Convert To Fails

Summary:

The Patron sample applications from Chapters 14 and 15 support the OLE 2 Convert dialog box. When Convert To is chosen from this dialog box, PATRON fails to perform the conversion properly.

Problem:

The problem is that CPage::FConvertObject() releases the page's IStorage pointer without committing it. Because Patron uses transacted storage, this discards all changes, including the conversion that just happened.

The code in error (listed on Page 817 of Inside OLE 2) is as follows:

```
   if ((CF_SELECTCONVERTTO & ct.dwFlags)
       && !IsEqualCLSID(ct.clsid, ct.clsidNew))
       {
       LPSTORAGE   pIStorage;

       //This should be the only close necessary.
       m_pTenantCur->RectGet(&rcl, FALSE);
       m_pTenantCur->StorageGet(&pIStorage);
       m_pTenantCur->Close(FALSE, FALSE);

       hr=OleStdDoConvert(pIStorage, ct.clsidNew);
       pIStorage->Release();
```

This code does not commit the changes to the storage affected by OleStdDoConvert before releasing pIStorage.

Solution:

To correct this problem, include a call to pIStorage->Commit() before releasing the storage:

```
   if ((CF_SELECTCONVERTTO & ct.dwFlags)
       && !IsEqualCLSID(ct.clsid, ct.clsidNew))
       {
       LPSTORAGE   pIStorage;

       //This should be the only close necessary.
       m_pTenantCur->RectGet(&rcl, FALSE);
       m_pTenantCur->StorageGet(&pIStorage);
       m_pTenantCur->Close(FALSE, FALSE);

       hr=OleStdDoConvert(pIStorage, ct.clsidNew);

       pIStorage->Commit(STGC_ONLYIFCURRENT);
       pIStorage->Release();

   ...
```

10. CLASSLIB String Allocations Fail with Some Compilers

Summary:

The Inside OLE 2 sample code includes a CLASSLIB library. Compiling the CStringTable class from this library may cause string space allocation failures with some compilers.

Problem:

The problem is an uninitialized local variable "cchUsed" in the function CStringTable::FInit(). This routine is found in the CLASSLIB source file SCSTRTABL.CPP.

Solution:

Initialize cchUsed to zero, for example, change the line

```
   UINT        cchUsed;
```

to read as follows:

```
   UINT        cchUsed=0;
```

11. CImpIPolyline::WriteToFile() Returns an Incorrect Value

Summary:

The Polyline sample DLL in Chapter 4 contains a function, CImpIPolyline::WriteToFile(), which returns an incorrect value.

Problem:

If WriteToFile()'s write operation succeeds, it should return NOERROR. Instead, it returns POLYLINE_E_WRITEFAILURE.

Solution:

In the code for the CImpIPolyline::WriteToFile() function, change the lines that read

```
   return (m_pObj->m_fDirty) ?
       NOERROR : ResultFromScode(POLYLINE_E_WRITEFAILURE);
```

to read as follows:

```
   return (!m_pObj->m_fDirty) ?
       NOERROR : ResultFromScode(POLYLINE_E_WRITEFAILURE);
```

Make this change to the Chapter 4 version of Polyline. CImpIPolyline::WriteToFile() is in the Polyline source file IPOLYLIN.CPP.

12. CImpIOleObject::GetClientSite() Returns an Incorrect Value
---

Summary:

The Cosmo sample applications in Chapters 10, 13, 14 and 16 contain a function, CImpIOleObject::GetClientSite(), that returns an incorrect value.

Problem:

When GetClientSite() encounters no errors, it should return NOERROR. Instead, it returns E_NOTIMPL.

Solution:

In the code for CImpIOleObject::GetClientSite(), change the line that reads

```
   return ResultFromScode(E_NOTIMPL);
```

to read as follows:

```
   return NOERROR;
```

Make this change to the versions of Cosmo in Chapters 10, 13, 14 and 16. CImpIOleObject::GetClientSite() is in the Cosmo source file IOLEOBJ.CPP.

13. Polyline Registers a Format with an Uninitialized String

Summary:

The Polyline sample DLLs in Chapters 5, 6, 11 and 16 attempt to register a clipboard format using an uninitialized string. As a result, data formats do not get properly transferred during clipboard, drag & drop, and compound document operations.

Problem:

The offending line is:

```
m_cf=RegisterClipboardFormat(PSZ(IDS_STORAGEFORMAT));
```

At constructor time, the stringtable referenced by the PSZ macro (this macro tries to access the member CPolyine::m_pST) has not been initialized, so m_cf is not set properly.

Solution:

To correct the error, the line of code listed above that initializes m_cf must be moved into the function CPolyline::FInit(). Specifically, move the line of code to occur after the code that initializes the stringtable:

```
if (!m_pST->FInit(IDS_POLYLINEMIN, IDS_POLYLINEMAX))
    return FALSE;

m_cf=RegisterClipboardFormat(PSZ(IDS_STORAGEFORMAT));
```

Make this one-line change to the versions of Polyline in Chapters 5, 11, and 16. CPolyline::FInit() is in the Polyline source file POLYLINE.CPP.

Correcting this error in the Chapter 6 version of Polyline requires an extra step. All lines initializing the FORMATETC arrays m_rgfeGet[0] and m_rgfeSet[0] in which m_cf is stored must also be moved into CPolyline::FInit(). The most convenient way to make this change is to simply move all the FORMATETC initialization to the end of FInit(). Note that Polyline does not use the FORMATETC arrays in Chapters 11 and 16, so this extra step is not necessary in those chapters.

14. DataTran Reference Counting Problem

Summary:

The DataTran data transfer object in Chapter 7 has a reference-counting problem in the function CImpIDataObj::GetData(). As a result, storage elements in the STGMEDIUM returned by GetData() may become invalid while they are still in use.

Problem:

GetData() incorrectly fails to call AddRef() on any IStorage or IStream pointer contained in the STGMEDIUM it returns.

Solution:

To correct this error, add the following lines of code to GetData(), immediately after the line "*pSTM = pRen->stm;":

```
/*
 * Must remember to AddRef any other objects
```

```
     * in the STGMEDIUM:  storages and streams.
     */
    if (TYMED_ISTORAGE==pSTM->tymed)
        pSTM->pstg->AddRef();

    if (TYMED_ISTREAM==pSTM->tymed)
        pSTM->pstm->AddRef();
```

GetData() is in the DataTran source file IDATAOBJ.CPP.

15. Component Cosmo Fails to Release Polyline Object

Summary:

The Component Cosmo samples in Chapter 6, 7, 8, 11 and 16 fail to fully clean up their allocations when closing a document. As a result, files may not be saved completely and unused code is left in memory.

Problem:

In the function CCosmoDoc::~CCosmoDoc(), CoCosmo obtains an IDataObject pointer on the Polyline object it maintains in the document. It then fails to release this pointer, resulting in the problems listed above.

Solution:

In the code for CCosmoDoc::~CCosmoDoc(), add a call to IDataObject::Release().

Change the lines

```
    if (SUCCEEDED(hr))
        pIDataObject->DUnadvise(m_dwConn);
```

to read as follows:

```
    if (SUCCEEDED(hr))
        {
        pIDataObject->DUnadvise(m_dwConn);
        pIDataObject->Release();
        }
```

CCosmoDoc::~CCosmoDoc() is in the CoCosmo source file DOCUMENT.CPP.

16. Cosmo Fails to Load When Used with HCosmo Handler

Summary:

The Cosmo sample applications in Chapter 10, 13, 14, or 16, when they are being used in conjunction with the custom object handler HCosmo from Chapter 11, fail to load and activate a Cosmo object. This problem occurs when a container application is reloading a Cosmo object from a saved file, and also when a container application is reactivating a Cosmo object after

the object has been first created and then deactivated.

Problem:

The problem actually lies in HCosmo. HCosmo incorrectly keeps the object stream open in the object's IStorage. It does this so that it can perform low-memory saves without having to reopen the stream.

However, when a process has a stream open and that stream has a particular IStorage as its parent, the OLE 2 implementation of STORAGE.DLL does not allow the process to open the stream a second time from the same parent IStorage. Therefore, when the Cosmo application receives the IStorage pointer from HCosmo, and attempts to open the same object stream that HCosmo already has open, the attempt to open the stream fails.

This problem shows up in HCosmo's implementation of IOleObject::DoVerb(). HCosmo delegates the DoVerb() call to the default handler, and the default handler attempts to launch Cosmo. As described above, Cosmo cannot open the object stream, so it returns an error of STG_E_READFAULT to the default handler, which in turn returns STG_E_READFAULT to HCosmo.

As noted above, HCosmo keeps the object stream open so that it can perform low-memory saves without having to reopen the stream. However, this is not necessary. Because HCosmo never makes changes to the object, it never has to save anything in a low-memory situation. Specifically, it never has to save any changes to the storage when its IPersistStorage::Save() is called with the fSameAsLoad flag set to TRUE. Because this is the only case where an IPersistStorage implementation should not attempt to allocate memory, it is totally unnecessary for HCosmo to cache any pointers to the stream.

Solution:

To correct the problem, make the following five changes to HCosmo's IPersistStorage implementation (HCosmo's implementation of IPersistStorage is in the source file IPERSTOR.CPP):

1. In CImpIPersistStorage::InitNew(), add the lines below immediately before the call to WriteClassStg():

```
m_pObj->m_pIStream->Release();
m_pObj->m_pIStream=NULL;
```

2. In CImpIPersistStorage::Load(), replace the line

```
m_pObj->m_pIStream=pIStream;
```

   with the following:

```
pIStream->Release();
m_pObj->m_pIStream=NULL;
```

3. In CImpIPersistStorage::Save(), remove all the code inside the "if (fSameAsLoad)" condition (it is unnecessary).

4. In CImpIPersistStorage::SaveCompleted(), remove the lines of code below

that occur inside the "if (NULL!=pIStorage)" condition:

```
hr=pIStorage->OpenStream("CONTENTS", 0, STGM_DIRECT
    | STGM_READWRITE | STGM_SHARE_EXCLUSIVE, 0
    , &pIStream);

if (FAILED(hr))
    return hr;

if (NULL!=m_pObj->m_pIStream)
    m_pObj->m_pIStream->Release();

m_pObj->m_pIStream=pIStream;
```

5. In CImpIPersistStorage::HandsOffStorage(), remove the condition and body of code below:

```
if (NULL!=m_pObj->m_pIStream)
    {
    m_pObj->m_pIStream->Release();
    m_pObj->m_pIStream=NULL;
    }
```

The m_pIStream member of CFigure can then be eliminated entirely, because it is no longer used in this IPersistStorage implementation.

17. Patron Does Not Work with Some Printer Drivers

Summary:

The Patron sample applications do not work correctly with certain printer drivers, including the standard PostScript driver.

One symptom of failure is getting a blank document window when creating a new Patron document; that is, no page image appears. Another symptom is getting a GP fault in the printer driver when attempting to execute the Printer Setup command from Patron's File menu.

Problem:

The problem is caused by the fact that Patron is not sensitive to the variable length of the DEVMODE structure returned by the printer driver. Patron normally obtains the DEVMODE data by calling the Windows PrintDlg() API; it then copies that data to a private stream in its Compound File. However, Patron only copies sizeof(DEVMODE) bytes, and therefore loses any additional driver-specific information that might exist at the end of the DEVMODE structure.

Some printer drivers, such as the Hewlett-Packard (HP) LaserJet PCL drivers, use no extra information. Patron works correctly with these drivers. Other drivers, such as the standard PostScript driver, use extra information; with these drivers, Patron fails.

Solution:

To correct this problem, change the implementation of Patron's CPages class so that it works with variable length DEVMODE structures, and therefore with all printer drivers.

The changes are too extensive to list in this article; as noted at the top of this article, an updated version of the Inside OLE 2 sample code source tree is available on the Software Library. The following is a brief overview of the code that needs to be changed to correct this problem:

The CPages class is defined in the header file PAGES.H, and implemented in PAGES.CPP. The functions that need to be updated are:

- CPages::DevmodeSet() and CPages::DevModeGet() in all chapters
- CPages::ConfigureForDevice() in Chapters 2 and 5
- CPages::DevReadConfig() in all chapters other than 2 and 5

After these changes have been made, the new Patron will be unable to read files generated by previous versions of Patron. There is no automatic conversion facility provided in these samples. To convert a file from the old Patron format to the new Patron format, follow these steps:

1. Run an old version of Patron from Chapter 14.
2. Run a new version of Patron from Chapter 15 (or, run the old Chapter 15 Patron and the new Chapter 14 Patron).
3. Load the old file into the old Patron.
4. Transfer all objects from the old Patron to the new Patron, using the clipboard or drag-and-drop method.
5. Save the new document in the new Patron.

18. ClassLib Creates Table of Invalid String Pointers

Summary:

The ClassLib sample framework implements a string table class called CStringTable. In certain situations the implementation of this class can create a table of invalid string pointers.

Problem:

The function CStringTable::FInit() allocates a block of memory and loads strings from the application's resources into that memory. It then initializes a table of far pointers into that memory, one pointer for each string.

FInit() then reallocates the memory block, in order to free up any unused space. This reallocation could move the memory block. Because the far pointers in the pointer array point into the block that has moved, they all become invalid.

Solution:

To avoid this problem, delete the following lines in the CStringTable::FInit() function:

```
   //Now reallocate the string memory to however much we used, plus 1
   psz=(LPSTR)_frealloc(m_pszStrings, cchUsed+1);

   if (NULL!=psz)
       m_pszStrings=psz;
```

This solution wastes a little memory, but is simpler than trying to recompute all the string pointers; ClassLib is intended to be a simple example.

CStringTable::FInit() is in ClassLib's CSTRABL.CPP file.

19. Errors in Enumerator Samples

Summary:

The enumerator samples in "Inside OLE 2" contain two errors. This section describes the errors and the changes necessary to fix them. The changes are detailed for the IEnumRect enumerator found in Chapter 3; similar changes are required in all enumerators in the book's sample code.

Problem 1:

If the first parameter to an enumerator's Next() function (called "celt" in the OLE 2 documentation, "cRects" in the sample) is 1, it is permissible for the last parameter (called "pceltFetched" in the documentation, "pdwRects" in the sample) to be NULL. IEnumRect::Next() simply returns FALSE in this case, which is incorrect.

Solution 1:

To correct this error, change the lines of code in IEnumRect::New() that read

```
   if (NULL==pdwRects)
       return FALSE;

   *pdwRects=0L;
```

to read as follows:

```
   if (NULL==pdwRects)
       {
       if (1L!=cRect)
           return FALSE;
       }
   else
       *pdwRects=0L;
```

Problem 2:

IEnumRect::Next() stores an incorrect value into the out parameter pdwRects.

Solution 2:

To correct this error, change the line of code in IEnumRect::New() that reads

```
   *pdwRects=(cRectReturn-cRect);
```

to read as follows:

```
   if (NULL!=pdwRects)
       *pdwRects=cRectReturn;
```

The same changes should be made to the IENUM.CPP files in the following "Inside OLE 2" sample code directories:

```
   interface
   chap03\enumc
   chap03\enumcpp
   chap06\polyline
   chap06\ddataobj
   chap06\edataobj
   chap07\datatran
```

20. Samples GP Fault When Closing Iconized Documents

Summary:

The sample applications may cause a GP fault when closing one or more iconized documents.

Problem:

The CClient::QueryCloseAllDocuments() function (found in the ClassLib source file CCLIENT.CPP) can cause this GP fault through incorrect use of its local variable hPrevClose.

Solution:

To correct this problem, change the code to CClient::QueryCloseAllDocuments() by adding the two lines below that read "hPrevClose=NULL":

```
BOOL CClient::QueryCloseAllDocuments(BOOL fClose)
    {
    ...

    for ( ; hWndT; hWndT=GetWindow(hWndT, GW_HWNDNEXT))
        {
        if (NULL!=hPrevClose)
            {
            pDoc=(LPCDocument)SendMessage(hPrevClose
                , DOCM_PDOCUMENT, 0, 0L);
            CloseDocument(pDoc);
```

```
            hPrevClose=NULL;            // ADD THIS LINE
            }

    ...

    //Close the last window as necessary.
    if (fClose && NULL!=hPrevClose)
        {
        pDoc=(LPCDocument)SendMessage(hPrevClose, DOCM_PDOCUMENT, 0, 0L);
        CloseDocument(pDoc);
        hPrevClose=NULL;                // ADD THIS LINE
        }

    ...
```

Additional reference words: 2.01 gpf gp-fault
KBCategory: kbole kbfile
KBSubcategory: LeTwoOth

Determining If an Object Is Capable of Visual Editing

PSS ID Number: Q99045
Authored 20-May-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, versions 2.0 and 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SUMMARY

It is impossible for a container to consistently determine whether an object will attempt negotiation for visual editing. However, by determining whether the object supports the IOleInPlaceObject interface, the container may want to assume that the negotiation will occur.

The default object handler will always fail a call to QueryInterface for IOleInPlaceObject if the object is in the loaded state. The container must first run the object and then query for the IOleInPlaceObject interface. If this QueryInterface call succeeds, then the container can assume that the object supports visual editing, and may start negotiation on the execution of a verb.

MORE INFORMATION

The following C++ code returns TRUE if an object may start visual negotiation when IOleObject::DoVerb is called:

```
BOOL fCanInPlaceActivate(LPOLEOBJECT lpObject)
{
LPOLEINPLACEOBJECT lpInPlaceObject;
BOOL retval;

    // Run the object.
    OleRun(lpObject);

    // Query for IOleInPlaceObject.
    HRESULT herr = lpObject->QueryInterface(IID_IOleInPlaceObject,
                                            (LPVOID FAR *)
                                            lpInPlaceObject);

    // Check the return value.
    if (herr == NOERROR)
        {
        retval = TRUE;
        lpInPlaceObject->Release();
        }
    else
```

```
        retval = FALSE;

    lpObject::Close(OLECLOSE_NOSAVE);

    return retval;
}
```

Additional reference words: 2.00 3.50 4.00
KBCategory: kbole kbprg kbcode
KBSubcategory: LeTwoInp

DOCERR: DISPCALC and OP_MULTIPLY

PSS ID Number: Q114972
Authored 18-May-1994 Last modified 16-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

SUMMARY

In the "OLE 2.0 Programmer's Reference," volume 2, page 12, please change the following line:

```
Calculator.Op  OP_MULTIPLY
```

to read as follows:

```
Calculator.Op = 3 ' OP_MULTIPLY
```

MORE INFORMATION

In order to set the Op method to multiply, you have to use the equal sign("=") to set the Op member equal to 3.

For easier reading, you should add the following line after the "DIM Calculator As Object" line:

```
Const OP_MULTIPLY = 3
```

This allows you to use the OP_MULTIPLY value instead of the hard-coded value of 3.

This code snippet has been corrected in the Online documentation provided with Visual C++ 2.x and the Win32 SDK.

Additional reference words: 2.01
KBCategory: kbole kbdocerr
KBSubCategory: LeTwoOth

DOCERR: F1 Help for Menu Items in an OLE 2.0 Container

PSS ID Number: Q118897
Authored 01-Aug-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SUMMARY

The "OLE 2.0 Programmer's Reference," volume 1, describes on pages 727 through 731 how an application that you would like to be a container for inplace edited objects can support F1 Help for menu items. This information is also contained in the OLE Technical Note CSHELP.DOC.

The application may have its own keyboard message filter added so that it can intercept the F1 key, or it may ask the OLE default object handler to add a message filter. A sample keyboard message filter is provided for applications that you would like to implement their own message filters. However, the sample message filter does not handle the case properly when an object is inplace activated.

This article contains a sample message filter that properly handles the inplace activated case.

MORE INFORMATION

If the sample message filter is used, when an object is inplace activated and the application attempts to provide F1 Help for one of the server's menu items, the server processes the menu command instead of providing Help. F1 Help works properly for menu items handled by the container.

In the sample filter, if an object is inplace activated and the F1 key is pressed, the message filter sends the registered message OM_POST_WM_COMMAND to the frame window. The frame window handles OM_POST_WM_COMMAND by posting a WM_COMMAND message to the container or the server as appropriate. After the OM_POST_WM_COMMAND message is sent (and thus after the WM_COMMAND message is posted), the filter calls IOleWindow::ContextSensitiveHelp(TRUE) for both the frame window and the inplace activated object, to enter context-sensitive Help mode. OLE's LRPC call to ContextSensitiveHelp() for the inplace activated object gives the server application the opportunity to check its message queue, and the WM_COMMAND message gets processed before the ContextSensitiveHelp() method is executed in the server. Because the inplace activated object does not know that it should be in context-sensitive Help mode, it processes the WM_COMMAND message as normal.

To correctly provide F1 Help for all menu items, the container may use the standard OLE 2.0 implementation of F1 menu processing. To use the standard

OLE 2.0 implementation of F1 menu processing, pass non-NULL pointers to the "lpFrame" and "lpActiveObj" parameters of OleSetMenuDescriptor().

Alternatively, the container may use the revised filter (listed below). In the revised filter, the calls to IOleWindow::ContextSensitiveHelp() are made before OM_POST_WM_COMMAND is sent to the frame window. This ensures that the server enters context-sensitive Help mode before the WM_COMMAND message is posted. One side effect is that if the F1 key is pressed while the current selection is a popup menu, the message filter will need to exit context-sensitive Help mode by calling IOleWindow::ContextSensitiveHelp(FALSE) for both the frame window and the inplace activated object.

Sample Code

```
/* Compile options needed:    (or  ; Assemble options needed:  )
*/

MessageFilterProc(int nCode, WPARAM wParam, LPARAM lParam)
{
   LPMSG lpMsg = (LPMSG) lParam;

   // If it is not the F1 key then let the message
   // (without modification) go to the next proc
   // in the message filter chain.
   if (lpMsg  &&  lpMsg->message == WM_KEYDOWN  &&
       lpMsg->wParam == VK_F1) {
      HMENU   hmenuPopUp;

      // Container app can not know what the current menu
      // selection is if an object is getting inplace
      // edited in it, because the menu bar is shared and
      // the menu messages are  intercepted and dispatched
      // to the appropriate process by OLE's frame
      // window sub-classing (see Box 3). So, when the
      // container sends the OM_POST_WM_COMMAND,
      //
      //    if (an object is getting inplace edited)
      //       it will go to Box 3, and OLE posts WM_COMMAND
      //          to the right process.
      //    else
      //       it will go to Box 4, which is app's own
      //          frame wnd proc, and its OM_POST_WM_COMMAND
      //          code can post the WM_COMMAND for the
      //          current selection (that the app maintains).
      //
      // With this scheme the same message filter proc can
      // be used whether the object is getting inplace
      // edited or not.
      // NOTE 1: If the current selection is a popup menu,
      // then WM_COMMAND can not be generated, so the code
      // for OM_POST_WM_COMMAND returns handle of the popup
      // menu, otherwise it returns NULL
      // NOTE 2: Do uOmPostWmCommand =
```

```
      //           RegisterWindowMessage("OM_POST_WM_COMMND")
      // at startup.

      // call the frame and active inplace object's
      // ContextSensitiveHelp() methods.
      lpFrame->ContextSensitiveHelp(TRUE);
      lpInPlaceActiveObj->ContextSensitiveHelp(TRUE);

      // when either of these 2 objects receive the WM_COMMAND
      // they will call the other one's ContextSensitiveHelp()
      // method. Note that the tree walk will not happen if the
      // CSH mode has been entered because of  F1 on selected menu.

      if (hmenuPopup = (HMENU)SendMessage(hwndFrame,
                                          uOmPostWmCommand,
                                          0, 0L)) {
         // Now the apps have 4 options:
         // 1. Give Help for the popup menu and cancel the
         //    menu mode as well as the CSH mode (WORD does
         //    this).
         // 2. Change the cursor to question mark cursor
         //    (== SHIFT+F1), and do not disturb the menu
         //    state, and enter the CSH mode. (EXCEL does
         //    this)
         // 3. Remove the CSH mode and cancel the menu mode.
         // 4. Leave the menu state as it is and ignore the
         //    F1 key (ie. don't pass the F1 key down the
         //    msg filter chain).
         //
         // We (OLE) recommend option 4, and if the container
         // app wants us to install the message filter, then
         // this is what we will do. In this sample code also
         // we are going for option 4.

         lpFrame->ContextSensitiveHelp(FALSE);
         lpInPlaceActiveObj->ContextSensitiveHelp(FALSE);
         return TRUE; // let the system know that we have
                      // handled the message
      }

      // Change message value to be WM_CANCELMODE and then call
      // the next proc in the message filter chain. When windows
      // USER's menu processing code sees this message it will
      // bring down menu state and come out of its menu processing
      // loop.

      lpMsg->message = WM_CANCELMODE;
      lpMsg->wParam  = NULL;
      lpMsg->lParam  = NULL;
   }
   return CallNextHookEx (hMsgHook, nCode, wParam, lParam);
}
```

Additional reference words: 2.01 3.50 4.00 Containers docerr
KBCategory: kbole kbdocerr

KBSubCategory: LeTwoCli

DOCERR: OleGetClipboard Not Needed Before OleSetClipboard

PSS ID Number: Q104961
Authored 04-Oct-1993 Last modified 16-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.0

The function descriptions for OleSetClipboard and OleGetClipboard in the OLE version 2.0 "Programmer's Reference: Creating Programmable Applications" manual incorrectly state that an application must obtain access to the clipboard by successfully calling OleGetClipboard before making a call to OleSetClipboard. In fact, a call to OleGetClipboard is not required before making a call to OleSetClipboard.

This documentation error also appears in the OLE version 2.0 online reference file, OLE2API.HLP, in the function descriptions for OleSetClipboard and OleGetClipboard.

The correct information is in the OLE 2.0 documentation included with Visual C++ 2.0.

Additional reference words: 2.00
KBCategory: kbole kbdocerr
KBSubcategory: LeTwoDdc

DOCERR: Text Corrections for Inside OLE 2

PSS ID Number: Q113188
Authored 29-Mar-1994 Last modified 21-Apr-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

SUMMARY

This article contains a list of errors in the text of the book "Inside OLE 2," by Kraig Brockschmidt (Microsoft Press).

Many of the errors listed below also affect the book's sample code; the descriptions for those errors include references to another Knowledge Base article, which describes the problems (and their solutions) in more detail.

For additional information on these problems and their solutions, please see the following article(s) in the Microsoft Knowledge Base:

ARTICLE-ID: Q113255
TITLE : SAMPLE: Corrections for Inside OLE 2 Sample Code

The sample code for "Inside OLE 2" was originally written to be compatible with OLE 2.0. This code has been updated to be compatible with OLE 2.01, and also to fix some problems in the original code. The entire source tree for the updated sample code is included in the INOLE2 Software Library sample. INOLE2 can be found in the Microsoft Software Library by searching on the word INOLE2.

MORE INFORMATION

List of Text Errors

Page 71: The second line of the last paragraph should read "with all the functions in the preceding table...". The book has "tale" instead of "table".

Pages 82-83, 92-93: The exact implementations of CImpIEnumRECT::Next on these pages are incorrect--for more information, see the section "Errors in Enumerator Samples" in the Knowledge Base article "SAMPLE: Corrections for Inside OLE 2 Sample Code."
Also, on page 83 there should be no space between "book" and "1632" in the text "#include <book 1632.h>".

Page 105: The second use of QueryInterface in the Transitive property should read "pInterface2->QueryInterface(IInterface3)". The book has IInterface2 as the parameter to QueryInterface, which is incorrect.

Page 157: A space is missing between "LPLPVOID" and "ppv" in the last parameter of the declaration CKoala::QueryInterface.

Page 161: The prototype for WEP should return an int, not VOID.

Page 197: The line of code under "if (IsEqual(riid, IID_IAnimal))" should read "*ppv=(LPVOID)m_pAnimal". In the text the underscore is mistakenly a space, dash, and space.

Page 199: A space is missing between "LPLPVOID" and "ppv" in the last parameter of the declaration CKoala::QueryInterface.

Page 243: The second sentence in the second paragraph should read:

"The Structured Storage definition of IStream allows streams to contain up to 2^64 (2 raised to the 64th power) addressable bytes of data."

The text incorrectly reads "264" instead of "2^64."

Pages 257-259: The code listings for CPages::DevModeSet, CPages::DevModeGet, and CPages::ConfigureForDevice should be changed to correct a problem with certain printer drivers in Patron. For more information on this problem, see the section "Patron Does Not Work with Some Printer Drivers" in the Knowledge Base article "SAMPLE: Corrections for Inside OLE 2 Sample Code."

Pages 289-290: Both LibMain and WEP functions should return an int. For more information, see the section "Incorrect Prototypes for LibMain() and WEP()" in the Knowledge Base article "SAMPLE: Corrections for Inside OLE 2 Sample Code."

Page 316: The implementation of CImpIEnumFormatEtc::Next is incorrect. For more information on this problem, see the section "Errors in Enumerator Samples" in the Knowledge Base article "SAMPLE: Corrections for Inside OLE 2 Sample Code."

Page 331: The description of IDataObject::GetCanonicalFormatEtc is incorrect. For correct details, see the Microsoft Object Linking and Embedding version 2.0 "Programmer's Reference."

Page 365: The "const LPRECTL" parameter to IViewObject::Draw should be of type "LPCRECTL" to match OLE 2.01. The book is correct for OLE 2.0. For more information, see the section "Compiler Errors on IViewObject::Draw()" in the Knowledge Base article "SAMPLE: Corrections for Inside OLE 2 Sample Code."

Page 368: The call to pIViewObject->Draw about should read:

"pIViewObject->Draw(dwAspect, -1, NULL, NULL, 0, hdcDraw, lprcBounds, NULL, NULL, 0);". In the book, the dwAspect and hDC parameters are incorrect.

Page 399: When CImpIDataObject::GetData calls AddRef() before returning, it should also call AddRef on any IStorage or IStream pointer

inside the STGMEDIUM, if that is the medium in use. For more information, see the section "Datatran Reference Counting Problem" in the Knowledge Base article "SAMPLE: Corrections for Inside OLE 2 Sample Code."

Page 437: The word "consumer" in the caption of Figure 8-6 should be "target."

Pages 544-546: In OLE 2.01 there is a new, additional parameter to OleUIAddVerbMenu called "idVerbMax", which comes immediately after "idVerbMin." This parameter was not present in OLE 2.0. For information on how this change affects the sample code, see the section "Compiler Errors on OleUIAddVerbMenu" in the Knowledge Base article "SAMPLE: Corrections for Inside OLE 2 Sample Code."

Page 554: Under OLE 2.01 the OleStdGetObjectDescriptorFromOleObject function (used in the Patron sample application) requires an additional LPSIZEL as the last parameter. For more information on how this change affects the sample code, see the section "Compiler Errors on OleStdGetObjectDescriptorFromOleObject()" in the Knowledge Base article "SAMPLE: Corrections for Inside OLE 2 Sample Code."

Pages 614-622: No ellipsis ("...") is necessary to separate functions in Listing 10-5. There is no code omitted from the listing in these places.

Page 630: The right curly brace just above the "else" at the bottom of the page should be indented to align with the call to ModifyMenu above it.

Pages 663-665: The bottom of page 663 mentions how a handler should hold onto open streams for low-memory save situations. This is important ONLY for elements that are entirely manipulated by the handler and never touched by the server. The handler should not cache or hold pointers to the same elements the server must access. For more information on this problem, see the section "Component Cosmo Fails to Release Polyline Object" in the Knowledge Base article "SAMPLE: Corrections for Inside OLE 2 Sample Code."

If a handler manipulates and changes data unaffected by the server, it must fulfill the IPersistStorage contract for only those formats. The server is responsible for all others that it modifies.

Page 668: The "const LPRECTL" parameter to IViewObject::Draw should be of type "LPCRECTL" to match OLE 2.01. The book is correct for OLE 2.0. For more information on how this change affects the sample code, see the section "Compiler Errors on IViewObject::Draw()" of the Knowledge Base article "SAMPLE: Corrections for Inside OLE 2 Sample Code."

Page 687: There should not be "//" before the large bold "IOLEOBJ.CPP"

in this code listing. The label is not part of the code.

Page 788: Same problem with OleStdGetObjectDescriptorFromOleObject as described above for page 554.

Page 814: Same problem with OleUIAddVerbMenu as described above for pages 544-546.

Page 817: Remove the second parameter to pTenantCur->Close. It should read "pTenantCur->Close(FALSE);". For more information on this problem, see the section "Patron GP Faults During Activate As" in the Knowledge Base article "SAMPLE: Corrections for Inside OLE 2 Sample Code."

Page 817: There should be a call to pIStorage->Commit(STGC_ONLYIFCURRENT) immediately after the call to OleStdDoConvert and before the call to pIStorage->Release. For more information, see the section "Patron Convert To Fails" of the Knowledge Base article "SAMPLE: Corrections for Inside OLE 2 Sample Code."

Page 819: Remove the second parameter to pTenantCur->Close. It should read "pTenantCur->Close(FALSE);". This matches the change necessary for a bug resolution described in the section "Patron GP Faults During Activate As" in the Knowledge Base article "SAMPLE: Corrections for Inside OLE 2 Sample Code."

Page 846: The parenthetical entry in the fourth line of the last paragraph should read "(linked objects are NOT allowed...)". The book reads "now" instead of "not".

Page 861: The second sentence of the third paragraph under "Active vs. UI Active and Inside-Out Objects" should read:

"The answer...as they are visible-end users DO NOT have to double-click..."

The DO NOT is important.

NOTE: Versions of "Inside OLE 2" printed after March 1994 (third printing or later) have been corrected.

Additional reference words: 2.01 mspress docerr
KBCategory: kbole kbdocerr
KBSubcategory: LeTwoOth

FIX: SR2TEST GP Faults During Object Shutdown

PSS ID Number: Q104141
Authored 08-Sep-1993 Last modified 16-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.0

SYMPTOMS

A general protection (GP) fault occurs when calling IOleObject::Close() on an SR2TEST object.

CAUSE

SR2TEST does not properly handle the case in which IOleObject::Close() is called after OleInPlaceObject::UIDeactivate() without first calling IOleInPlaceObject::InPlaceDeactivate().

RESOLUTION

Call IOleInPlaceObject::InPlaceDeactivate() before calling IOleObject::Close.

STATUS

Microsoft has confirmed this to be a problem in OLE version 2.0. This problem was corrected in OLE version 2.01.

Additional reference words: 2.00 gpf gp-fault
KBCategory: kbole kbfixlist kbbuglist
KBSubcategory: LeTwoTls

FIX: TYPE_E_CANTLOADLIBRARY Error on Win32s

PSS ID Number: Q131051
Authored 02-Jun-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.03
- Microsoft Win32s version 1.20

SYMPTOMS

A 32-bit OLE Automation server that implements dual or vtable binding interfaces finds that loading the type library or a typeinfo from the type library fails under Win32s version 1.20 with this error:

```
TYPE_E_CANTLOADLIBRARY
```

CAUSE

A problem in OpenFile in Win32s version 1.20 causes the search for STDOLE32.TLB to be done incorrectly.

STATUS

Microsoft has confirmed this to be a bug in Win32s version 1.20. This bug was corrected in Win32s version 1.25a, which can be obtained from the Microsoft Software Library by searching for PW1118.EXE.

Download PW1118.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for PW1118.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download PW1118.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the SOFTLIB\MSLFILES directory
  Get PW1118.EXE

Additional reference words: 2.0 2.00
KBCategory: kbprg kbole kbfixlist kbbuglist kbfile
KBSubcategory: LeTwoArc

FORMATETC for IDataObject::DAdvise Must be Validated

PSS ID Number: Q114600
Authored 08-May-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SUMMARY

Applications implementing IDataObject must validate the FORMATETC structure that is passed to IDataObject::DAdvise(). Some commercial applications, such as Microsoft Excel 5.0, will try to set up an Advise on their own native format before setting up an Advise on the format selected by the user in the Paste Special dialog box while creating a link.

MORE INFORMATION

When validating the FORMATETC structure, the server application needs to allow wildcard Advises to succeed. A Wildcard Advise is an Advise set up by the default link object so that it can properly maintain the time-of-last-change. A Wildcard Advise is requested by passing a FORMATETC with cfFormat = 0, ptd = NULL, dwAspect = -1L, and tymed = -1L. Code to validate the FORMATETC structure might look as follows:

```
   // if not a Wildcard Advise and the data format is not supported,
   // return failure.

   if( !(pfmtetc->cfFormat == NULL  &&
         pfmtetc->ptd      == NULL  &&
         pfmtetc->dwAspect == -1L   &&
         pfmtetc->lindex   == -1L   &&
         pfmtetc->tymed    == -1L)
      && FAILED(hres = QueryGetData(pfmtetc)) )
      return hres;

   // Now pass on to the Data Advise holder
```

For more information on the IDataObject::DAdvise() member function, please refer to pages 436-439 of the "OLE Programmer's Reference, Volume 1".

Additional reference words: 2.01 3.50 4.00
KBCategory: kbole kbprg
KBSubcategory: LeTwoSvr

How to Obtain Documentation for MS Graph 5.0 Automation

PSS ID Number: Q131049
Authored 02-Jun-1995 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Graph 5.0
- Microsoft OLE Libraries for Windows and Win32s, version 2.03
- Microsoft OLE libraries included with:
    - Microsoft Windows NT version 3.5
    - Microsoft Windows 95

Documentation of the Microsoft Graph version 5.0 automation properties and methods can be found in the VBAGRP.EXE self-extracting file in the library of the ADK/ADT section of the MSACCESS forum on Compuserve.

Additional reference words: 2.0 3.50 4.00 2.00
KBCategory: kbprg
KBSubcategory: LeTwoAto

How to Test OLE Applications with Outline

PSS ID Number: Q131154
Authored 04-Jun-1995 Last modified 07-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5
- Microsoft OLE Libraries for Windows and Win32s, versions 2.01, 2.02, and 2.03
- Microsoft OLE Libraries included with:
  - Microsoft Windows NT version 3.5
  - Microsoft Windows 95

SUMMARY

To understand how an OLE application should handle an OLE scenario, you may find it helpful to examine an existing application that handles that scenario correctly.

The OUTLINE series of OLE sample applications can provide a log of debug output. This log shows all of the OLE calls that the OUTLINE application handles and the OLE calls that it makes. Examining this log is a useful OLE problem-solving tool, and an excellent way to learn about OLE programming.

MORE INFORMATION

The OUTLINE series consists of four different applications: CNTROUTL (OLE container), SVROUTL (OLE server), ICNTROTL (in-place editing-capable OLE container), and ISVROTL (in-place editing-capable OLE server). The OUTLINE series is included as sample code with the OLE SDK and with the Win32 SDK.

Each of the four OUTLINE applications implements a Debug Level dialog box. The following table show the debugging information produced by each debug level value:

Debug Level	Debugging Information Produced
0	No debug messages (default).
1	Print message when a Document or Application is destroyed.
2	Trace every OLE API and method call.
3	Give a more OUTLINE app specific context to the OLE calls.
4	Track all memory reference counts (AddRef/Release).

The debug output forms a log of all the incoming and outgoing OLE method calls that OUTLINE handles. Use the DBWIN tool to capture and display this output. Note that although there are both 16-bit and 32-bit versions of the OUTLINE samples, DBWIN can currently only be used to capture output

from the 16-bit samples.

Benefits to Examining the Watch Log

The first benefit to examining the watch log produced by one of the OUTLINE samples is that it shows the series of calls that an OLE application needs to make in order to handle a given OLE scenario. For example, to see what calls an OLE container application needs to make when transitioning an embedded object from the loaded to the running state, double-click an object embedded into a CNTROUTL document to run that object; then examine the log that CNTROUTL produces.

The next benefit arises from the fact that the source code to the OUTLINE samples is included with Microsoft Visual C++. The watch log shows you the key calls that OUTLINE makes to handle some particular scenario. Using a GREP utility, you can search the OUTLINE source code for those calls, and then examine that code. The OUTLINE source code is a very thorough implementation of OLE functionality.

In addition, you can run the OUTLINE application under a debugger, setting break points before the key calls, and then stepping through that code.

Finally, the OUTLINE source code contains many source code comments labeled "OLE2NOTE." These comments contain in-depth discussions of some of the more detailed aspects of OLE programming.

Additional reference words: kbinf 2.01 2.02 2.03 3.50 4.00
KBCategory: kbole kbtshoot
KBSubcategory: LeTwoTls

How Visual Basic Automation Statements Map to OLE Calls

PSS ID Number: Q122288
Authored 01-Nov-1994 Last modified 21-Apr-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.02
- Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article lists the OLE API and interface calls to which the Visual Basic CreateObject and GetObject calls map. Each listing shows how to obtain the IDispatch of an OLE Automation object. You can then invoke a property or method of the OLE Automation object by using IDispatch::GetIDsOfNames and IDispatch::Invoke.

MORE INFORMATION

The code listed below does not do any error checking. When you implement it, you should add the code to do necessary error checking. See the OLE SDK documentation for more information about each function or interface. Each code listing shows how to obtain the pointer to IDispatch of an Automation object in the pdisp variable.

CreateObject(progID)

This creates a new automation object and returns a pointer to the IDispatch of that object.

```
CLSID clsid;
LPUNKNOWN punk;
LPDISPATCH pdisp;

OleInitialize(NULL), if OLE isn't already loaded.
CLSIDFromProgID(progID, &clsid);
CoCreateInstance(clsid, NULL, CLSCTX_SERVER,
                 IID_IUnknown, (LPVOID FAR*)&punk);
punk->QueryInterface(IID_IDispatch, (LPVOID FAR*)&pdisp);
punk->Release();
```

GetObject(filename, progID)

GetObject has three different semantics depending on the number of parameters passed. An automation object must implement IPersistFile to support GetObject(filename, progID).

```
CLSID clsid;
LPUNKNOWN punk;
```

```
LPDISPATCH pdisp;
LPPERSISTFILE pPF;

OleInitialize(NULL), if OLE isn't already loaded
CLSIDFromProgID(progID, &clsid);
CoCreateInstance(clsid, NULL, CLSCTX_SERVER,
                 IID_IUnknown, (LPVOID FAR*)&punk);
punk->QueryInterface(IID_IPersistFile, (LPVOID FAR*)&pPF);
punk->Release();

pPF->Load(filename, 0);
pPF->QueryInterface(IID_IDispatch, (LPVOID FAR*)&pdisp);
pPF->Release();
```

GetObject(filename,)

GetObject has three different semantics depending on the number of parameters passed. An automation object must implement IPersistFile to support GetObject(filename,).

```
LPBC pbc;
ULONG cEaten;
LPMONIKER pmk;
LPDISPATCH pdisp;

OleInitialize(NULL), if OLE isn't already loaded
CreateBindCtx(0, &pbc);
MkParseDisplayName(pbc, filename, &cEaten, &pmk);
BindMoniker(pmk, 0, IID_IDispatch, (LPVOID FAR*)&pdisp);
pmk->Release();
pbc->Release();
```

GetObject(, progID)

GetObject has three different semantics depending on the number of parameters passed. An automation object must call RegisterActiveObject to support GetObject(, progID).

```
CLSID clsid;
LPUNKNOWN punk;
LPDISPATCH pdisp;

OleInitialize(NULL), if OLE isn't already loaded
CLSIDFromProgID(progID, &clsid);
GetActiveObject(clsid, NULL, (LPVOID FAR*)&punk);
punk->QueryInterface(IID_IDispatch, (LPVOID FAR*)&pdisp);
punk->Release();
```

Additional reference words: 2.00 3.00
KBCategory: kbold kbprg kbcode
KBSubcategory: LeTwoAto

Insert Link from File Changes Current Directory

PSS ID Number: Q109553
Authored 04-Jan-1994 Last modified 21-Apr-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

The Insert Object dialog box allows a user to insert a linked object into the container document from a file residing on disk. This action changes the current working directory to the drive and directory in which the file resides. If this file is located on a network drive and the network drive becomes unavailable after the link is established, any subsequent attempt to insert an embedded object will fail until the working directory is changed to a drive and directory that is accessible.

The problem materializes when the OLE 2.0 libraries attempt to WinExec a server application for a subsequent embedded object. In this situation, the call to WinExec fails because the current working directory is invalid.

The Open common dialog box changes the current working directory unless OFN_NOCHANGEDIR is specified. The UI Change Source dialog box depends on the change directory functionality, however, and therefore the Insert Object dialog box cannot be modified to correct the problem.

Additional reference words: 2.01 OpenFile
KBCategory: kbole kbprg
KBSubcategory: LeTwoTls

Limits of VB 3.0 & Disptest as Automation Controllers

PSS ID Number: Q122287
Authored 01-Nov-1994 Last modified 21-Apr-1995

The information in this article applies to:

- Microsoft Visual Basic for Windows, version 3.0
- Microsoft OLE Libraries for Windows and Win32s, version 2.02

SUMMARY

Visual Basic version 3.0 for Windows and Disptest (the automation controller that shipped with OLE version 2.0) have the following limitations as automation controllers in addition to those described on pages 67-69 of the OLE 2 Programmer's Reference, Volume 2:

- Parameters cannot be passed by reference to Automation properties or methods.

- Array elements cannot be accessed.

- When an exception is raised by an automation object, the value of the wCode field of the EXCEPINFO structure is not used. Instead the value 440 is displayed.

MORE INFORMATION

Parameters Cannot Be Passed by Reference

Visual Basic version 3.0 for Windows and Disptest cannot pass parameters by reference to automation properties and methods.

This is a limitation of Visual Basic version 3.0 and Disptest, not of OLE Automation. OLE Automation allows building controllers that pass parameters by reference.

Here are three workarounds:

- Use return values of properties and methods to return a value instead of using byref parameters for this purpose. This approach cannot be used when a property or method returns multiple values.

  -or-

- Use third party controllers (see the list below) while you wait for a future version of Visual Basic, or use Visual Basic for Applications, which ships with Microsoft Excel version 5.0. Microsoft Visual Basic for Applications cannot be redistributed by or incorporated into products that are not manufactured by Microsoft. However, if user of your application has Microsoft Excel version 5.0 installed, you can use its

Visual Basic for Applications as a controller of any Automation server.

Here are two known third-party controllers:

NOTE: The products included here are manufactured by vendors independent of Microsoft; we make no warranty, implied or otherwise, regarding these products' performance or reliability.

Softbridge Basic Language
Mystic River Software, Inc. 125 CambridgePark Drive, Cambridge MA 02140
Tel:1-800-298-3500, 617-497-1585
Fax:617-864-7747

Summit BasicScript
Summit Software Company, 2844 Sweet Road, Jamesville, NY 13078
Tel: 315-677-9000
Fax: 315-677-3224
Internet: info@summsoft.com
Compuserve: 71211,3504

-or-

- Write a DLL that has exported functions that correspond to Automation methods and properties exposed by the automation server. Visual Basic version 3.0 can call these exported DLL functions, which in turn can call the corresponding automation properties or methods in the automation server. This approach works because Visual Basic version 3.0 can pass parameters by reference to DLL functions.

  The DLL functions can pass parameters by reference to the properties and methods of the automation method or property because OLE supports passing parameters by reference. This approach may not be suitable for some automation servers that implement nested objects.

Array Elements Cannot Be Accessed

Array elements cannot be accessed in Visual Basic version 3.0 or Disptest. As a result, code similar to the following code cannot be executed in Visual Basic version 3.0 or in Disptest if selection returns a safearray. This is because safearray elements cannot be accessed in Visual Basic version 3.0 or in Disptest.

```
Value = ObjVar.Selection(I)
ObjVar.Selection(I) = Value
```

Workarounds include using indexed properties, third-party controllers, Visual Basic for Applications in Microsoft Excel version 5.0, or waiting for a future release of Visual Basic.

Indexed properties are properties that take parameters. For example, in the example code, Selection could be made a property that takes an index parameter. Here is a description in the .ODL file of an indexed parameter called Value whose property type is VARIANT:

```
   [propget] VARIANT Value(long index);
   [propput] void Value(long index, VARIANT NewValue);
```

In a dispinterface, an indexed property should be used under the 'methods' keyword. A Microsoft Foundations Classes (MFC) implementation can use the DISP_PROPERTY_PARAM macro in the dispatch map to implement an indexed property.

The disadvantage of implementing array element access using indexed properties is the performance penalty caused by each indexed property access requiring the overhead of an RPC/LRPC call (for LocalServer Automation objects).

Value of EXCEPINFO.wCode Not Displayed on Exceptions

When an automation object raises an exception by returning DISP_E_EXCEPTION from IDispatch::Invoke and by filling the pexcepinfo parameter of this method, Visual Basic version 3.0 and Disptest will ignore the value returned in the wCode field of the EXCEPINFO structure, instead returning the value 440. Note that either the wCode or scode field of EXCEPINFO should be set to 0 -- both cannot be used.

Workarounds include using third-party controllers, Visual Basic for Applications in Microsoft Excel version 5.0, or waiting for a future release of Visual Basic.

Additional reference words: 2.00 3.00 5.00
KBCategory: kbole kbprg kb3rdparty
KBSubcategory: LeTwoAto

Object Creation Overview

PSS ID Number: Q104139
Authored 08-Sep-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.0
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

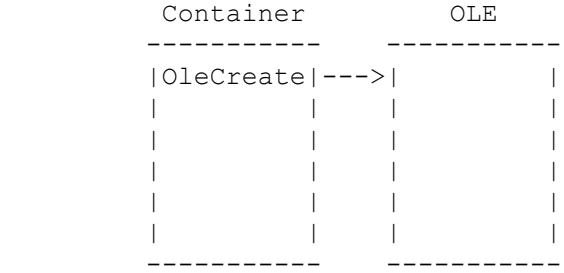
SUMMARY

The actual process of creating an OLE object can seem confusing when first learning OLE. To elucidate the object creation process, this article describes the steps taken by the OLE 2.0 libraries in response to a call to OleCreate(), with the OLERENDER_DRAW format passed as the renderopt parameter. This article gives a basic overview of the object creation process; it does not describe specific application programming interfaces (APIs) in detail.

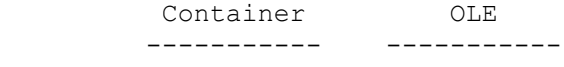
MORE INFORMATION

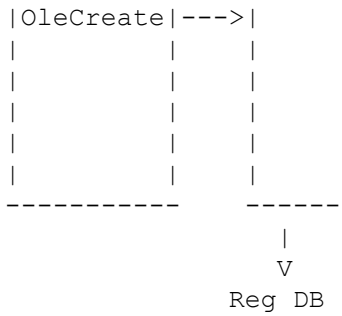
Typically, the container application presents the user with a dialog box containing the various types of objects that can be created on a particular system. These object types are enumerated from the Registration Database--each entry in the database has a name that the user can understand, as well as a unique identifier for the object type called a CLSID.

The container tells OLE that it wants to create an object. At this point, the container gives OLE the CLSID of the desired object, a pointer to a child storage within the container's compound file, and a pointer variable for OLE to return a pointer to a requested interface (typically IOleObject):

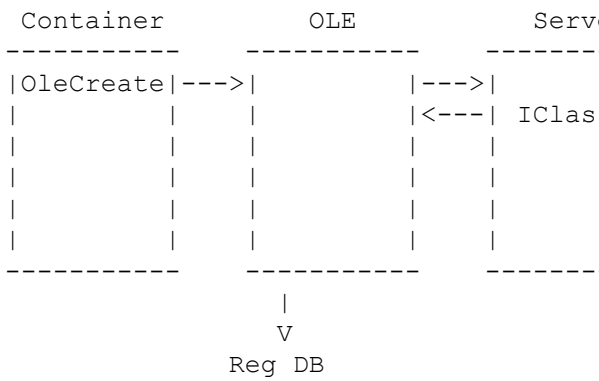


The OLE libraries then consult the registration database to find the name of the executable file associated with the CLSID:

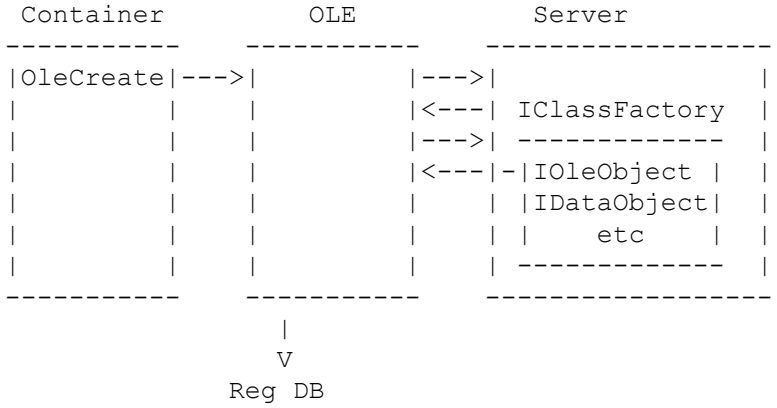




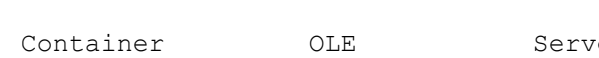
The server is then started with the -Embedding flag. When the server parses the command line and sees the -Embedding flag, it knows that it is being started on behalf of OLE and should remain hidden until it is explicitly told to become visible. At this point, the server registers a pointer to its IClassFactory interface with OLE:

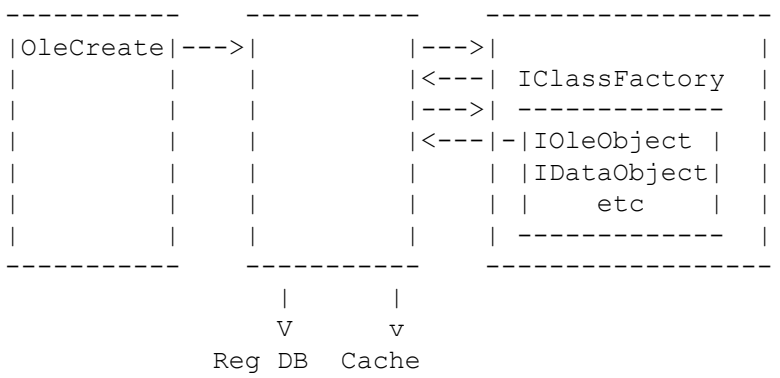


OLE calls the server through the IClassFactory interface and asks to instantiate an object, returning the pointer to the interface requested by the container in the creation routine:

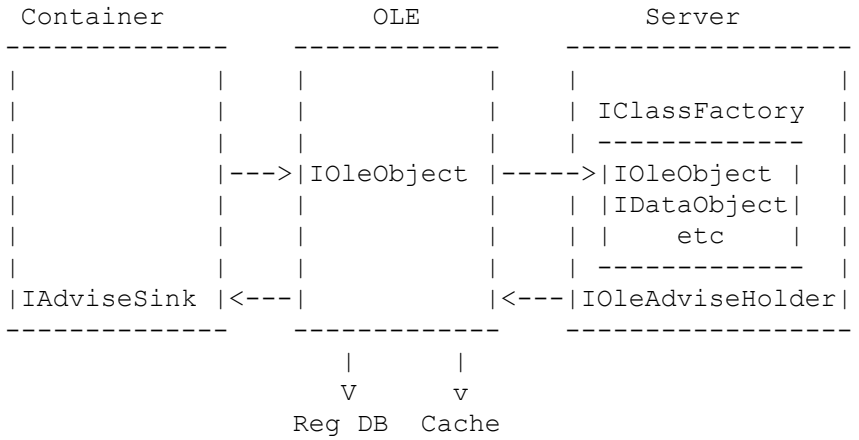


OLE queries the object for the IDataObject interface, and then calls GetData method to fill the presentation cache with a metafile, device-independent bitmap (DIB), or a bitmap. Later, when the container asks the object to draw itself, OLE will intercept the call and use one of the entries in the presentation cache:





At this point, the creation routine returns to the container. Now, the container will generally query the interface pointer returned for the IViewObject interface. The container uses the IViewObject interface to ask the object to notify the container (through the container's IAdviseSink interface) every time the object's view changes, so that the container can update its display. At this point, the server creates an advise holder to notify the object of its changes:



Note, the object still is not visible at this point. Now an explicit call to IOleObject::DoVerb() must be made with the verb OLEIVERB_SHOW. This tells the server to make the object visible for editing.

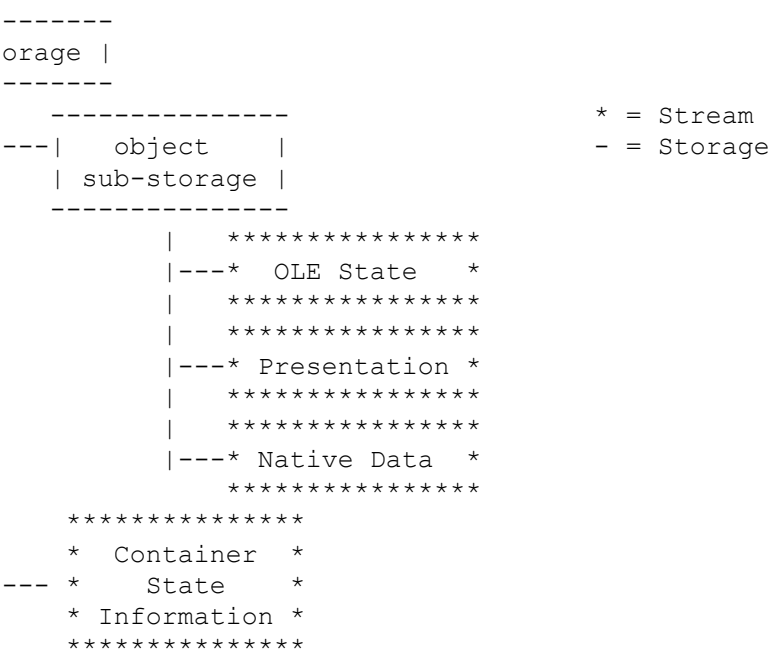
Every time the user makes a change to the object, the object calls OLE through the advise holder to notify the libraries that the view has been changed. The libraries then query the object for the new presentation(s) to fill the presentation cache. Then the advise sink of the container application is called to notify the container that its view is no longer up to date, and should be repainted.

The container then in turn tells OLE to draw the object, at which point the libraries render the information from the cache.

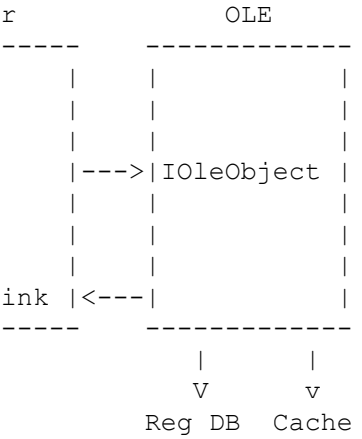
At some point, the user tells the server that he or she is finished editing the object. The server tells the container that it is now going away, and that the object needs to be saved into the container's persistent storage. The container responds by telling the object to save itself into a substorage of the container's compound file. The

OLE libraries "intercept" this call and save internal state information, as well as the presentation cache, into streams of this substorage. Then, the libraries tell the object to save itself into the container provided substorage.

After saving the object, the container's compound file resembles the following:



The container can continue to redraw the object even though the server application is no longer around because the object's presentation is stored in OLE's internal cache:



Now the user can save the document in the container application and exit. Upon exiting, the container queries the object for the IViewObject interface so the container can tell OLE to stop sending updates. Then the container asks the object to close.

If the user now reloads the document, or moves the document to another

machine, the OLE libraries can still render the object because the presentation is actually stored within the container's compound file. When the object is reloaded, the libraries can refill the presentation cache from the information stored within the compound file.

Additional reference words: 2.00 3.50 4.00
KBCategory: kbole kbprg
KBSubcategory: LeTwoCdt

Objects in .EXE Cannot be Aggregated

PSS ID Number: Q114598
Authored 08-May-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, versions 2.0 and 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

Attempting to create an aggregate object with a Non-Control object implemented in another .EXE is not possible. Passing a value for the punkOuter parameter of IClassFactory::CreateInstance() in this situation results in the error value CLASS_E_NOAGGREGATION.

The proxy for IClassFactory returns this error value immediately, without calling the CreateInstance() method in the object. According to the rules of aggregation, the Non-Control Object must not reference count the pointer to the controlling IUnknown. If the parts of the aggregate object reside in separate process spaces, the proxy for the controlling IUnknown will be freed prematurely due to the lack of this reference count. Without the interface proxy, the pointer to the controlling IUnknown stored by the Non-Control Object is no longer valid, causing problems when delegating calls to the outer IUnknown.

For more information on the process of Aggregation, please refer to the "OLE 2 Programmer's Reference", "Working with Windows's Objects".

Additional reference words: 2.01 3.50 4.00
KBCategory: kbole kbprg
KBSubcategory: LeTwoOh

OLE 2.02 Update for Windows and Win32s

PSS ID Number: Q121835
Authored 19-Oct-1994 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.02

SUMMARY

The file OLE202.EXE contains upgrades and additions to the 16-bit OLE files shipped with the Win32 SDK for Windows NT 3.5. This also includes the 32-bit counterparts for Win32s to the 16-bit tools shipped with the 16-bit OLE2 SDK. These files supercede the components of the OLE version 2.01 SDK for Windows 3.1x and Windows for Workgroups.

For additional information regarding Win32s availability, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID: Q122235
TITLE : Microsoft Win32s Upgrade

For additional information regarding the availability of updated OLE 2.02 libraries, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID: Q123087
TITLE : WW1116: OLE Version 2.02

MORE INFORMATION

Download OLE202.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for OLE202.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download OLE202.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \softlib\mslfiles directory
  Get OLE202.EXE

OLE202.EXE holds the following 16-bit updates and additions to the OLE202 SDK that shipped with Windows NT version 3.5:

LRPCSPY.EXE
DOBJVIEW.EXE
IROTVIEW.EXE
OLE2VIEW.EXE
COMPOBJ.DLL
OLENLS.H
VARIANT.H
DISPATCH.H
OLE2DISP.LIB
OLE2NLS.LIB
TYPELIB.LIB

and additions:

RPCSPY32.EXE
DOBJVW32.EXE
IROTVW32.EXE
OLE2VW32.EXE

NOTE: The tools contained in OLE202.EXE are designed to work with Windows 3.1 (16-bit) or Win32s 1.2, and the OLE 2.02 DLLs only. Do not mix the OLE 2.02 DLLs with the OLE 2.01 DLLs. You must install the OLE 2.02 DLLs over the OLE 2.01 DLLs. The tools include new automation features such as vtable binding and OLE32 tools for Win32s. Please see the Win32 OLE help file or the OLEHLP.TXT file for details about the new automation features.

Additional reference words: 2.01 2.02 3.50 3.10 1.20
KBCategory: kbole kbtool kbfile
KBSubcategory: letwomisc

OleCreateFromFile() Does Not Check for Reserved Names

PSS ID Number: Q111015
Authored 03-Feb-1994 Last modified 21-Apr-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

Before an OLE 2.0 container application calls OleCreateFromFile() to create an embedded object from the contents of a named file, it must make sure that the filename it passes as the lpszFile parameter to OleCreateFromFile() is not a reserved MS-DOS device name, such as COM1 or LPT1. This check is necessary because OleCreateFromFile() does not perform such a check itself.

A container application generally obtains the filename to pass to OleCreateFromFile() by presenting the user with the Insert Object dialog box. Many containers use the OLE2UI library implementation of the Insert Object dialog box, by calling OleUIInsertObject(). OleUIInsertObject() performs a check to make sure that the filename supplied by the user is not a reserved MS-DOS device name; if the user supplies a filename such as "COM1", OleUIInsertObject() brings up a modal dialog box stating "File COM1 Does Not Exist."

Therefore, container applications that use OleUIInsertObject() do not have to perform a check for reserved names.

Additional reference words: 2.01
KBCategory: kbole
KBSubcategory: LeTwoCdt

Passing Structures in OLE Automation

PSS ID Number: Q122289
Authored 01-Nov-1994 Last modified 21-Apr-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.02
- Microsoft Visual Basic for Windows, version 3.0

SUMMARY

OLE Automation does not allow structures to be passed as parameters. Automation servers that work with standard controllers can model a structure as an automation object. Automation servers that work with custom controllers can pass structures as a safearray of VT_UI1 or by passing a data transfer object that supports IDataObject.

MORE INFORMATION

OLE Automation does not support passing structures as parameters to automation methods and properties. Here are some ways to work around this:

NOTE: Standard controllers are products such as Visual Basic that can control any automation server. A custom controller is written to control a specific automation server.

If the Automation server is designed to work with standard controllers such as Visual Basic, the structure could be modeled as an automation object. For example, consider this structure:

```
struct point {
    short x;
    short y;
};
```

This could modeled as an automation object with two properties, x and y. Visual Basic code such as the following could be used to get and set the values of the properties. Note how the CreateLine method, which required two points as parameters, is implemented as taking references to two automation objects.

```
Dim Application As Object
Dim P1 As Object
Dim P2 As Object
Dim Line As Object
:
Set P1 = Application.CreatePoint()
P1.x = 3
P1.y = 4
Set P2 = Application.CreatePoint()
P2.x = 20
```

```
P2.y = 21
Set Line = Application.CreateLine(P1, P2)
```

The CreatePoint method returns a Point automation object whose x & y properties are set. The type of the value returned by CreatePoint is IDispatch* or a pointer to an object that supports automation. The CreateLine method takes two parameters of type IDispatch* or a pointer to an object that supports automation.

The disadvantage of this solution is that if the automation server is not an inproc server, each property access will result in the overhead of an LRPC/RPC call.

If the automation server is designed to work with custom controllers, the structure could be serialized into a safearray of VT_UI1 and the resultant binary data could be passed as a parameter of type SAFEARRAY(unsigned char). Another solution is to create a data transfer object that supports IDataObject. The IUnknown of this data transfer object could be passed in a parameter of type IUnknown*. The server could then use IDataObject::GetData with a private clipboard format to get the storage medium in which the structure was serialized. Both these solutions will not work with standard controllers such as Visual Basic.

NOTE: Serializing the structure into a BSTR will not work because the Unicode-ANSI conversions done by OLE's 16:32 bit interoperablity layer assumes that BSTRs contain strings and not binary data. Consequently, binary data passed in BSTRs can be corrupted by such conversions. However, serializing binary data into BSTRs will work if both controller and server are 16-bit. It will also work if both the controller and server are 32-bit and both support Unicode. This is because 32-bit OLE supports only Unicode, not ANSI. A safearray of VT_UI is preferred because of these limitations of passing binary data through BSTRs.

Additional reference words: 2.00 3.00
KBCategory: kbole kbprg kbcode
KBSubcategory: LeTwoAto

Passing Variant Parameters in OLE Automation

PSS ID Number: Q104960
Authored 04-Oct-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.0
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

A variant must be initialized using VariantInit after creation and before it is passed to a function. A variant with a valid value must be cleared using VariantClear before a new value is assigned to it.

A variant must be passed to or from a function in a valid state and the contents of the variant must be correctly freed. This requires that the following be done:

1. On creation of a variant, call VariantInit before passing it to a function.

2. A function that is passed a variant should clear it using VariantClear to free the previous contents before assigning it a new value.

VariantInit sets the vt field of the VARIANT (or VARIANTARG) structure to VT_EMPTY but does not free the contents of the variant. VariantClear frees the contents of the variant depending on the current value of the vt field and then sets the vt field to VT_EMPTY.

Passing an uninitialized variant to a function is a common error. This may cause problems when the VariantClear call in the function tries to free the contents of the uninitialized variant based on the undefined value of the vt field.

Additional reference words: 2.00 2.01 3.50 4.00
KBCategory: kbole kbprg
KBSubcategory: LeTwoAto

PRB: Calling IOleObject::InitFromData Returns E_NOTIMPL

PSS ID Number: Q108312
Authored 08-Dec-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE version 1.0
- Microsoft OLE Libraries for Windows and Win32s, versions 2.0 and 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

After an OLE 2.0 container successfully returns from a call to OleCreate(), a subsequent call to IOleObject::InitFromData() returns with a failure of E_NOTIMPL.

CAUSE

The object being manipulated is an OLE 1.0 object.

RESOLUTION

This is a limitation of an OLE 1.0 server application. Contact the manufacturer of the server application to find out the availability of an OLE 2.0-compliant version of the server.

MORE INFORMATION

OLE 1.0 does not have a feature equivalent to IOleObject::InitFromData, and therefore the OLE 2.0 libraries cannot synthesize this functionality in the OLE 1.0 compatibility layer.

If the object application is an OLE 2.0-compliant application, the error is returned by the object application's implementation of this method.

Additional reference words: 1.00 2.00 2.01 3.50 4.00
KBCategory: kbole kbprg kbprb
KBSubcategory: LeTwoArc

PRB: Compiler Doesn't Lay Out Overloaded Functions in Order

PSS ID Number: Q131104
Authored 04-Jun-1995 Last modified 07-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.03
- Microsoft OLE libraries included with:
  - Microsoft Windows NT version 3.5
  - Microsoft Windows 95
- Microsoft Visual C++ for Windows, version 1.52
- Microsoft Visual C++, 32-Bit Edition, version 2.1

SYMPTOMS

The Microsoft Visual C++ compiler does not lay out overloaded functions in the vtable in the order in which they are declared. This does not adhere to the COM binary standard. (See the OLE Specification section 3.1.1.1 for a discussion of the COM binary standard.) If there are no overloaded functions, the COM binary standard is followed.

In general, it is not a good idea to use overloaded methods in a COM interface because this precludes use of the interface from languages like C that do not support overloading.

CAUSE

All the overloads of a function are laid out together in successive entries in the reverse order of their declaration, overall in the order of the first overload of each name. Therefore given this declaration:

```
class A {
   virtual void a();
   virtual void b();
   virtual void a(int);
   virtual void b(int);
};
```

Entries in the vtable are laid out in this order: A::a(int), A::a(), A::b(int), A::b(). The COM binary standard requires that functions be laid out in the order in which they are declared.

STATUS

This behavior is built into the design. It cannot be modified because of legacy code that depends on this behavior.

Additional reference words: 2.03 1.52 4.00 3.50 2.10

KBCategory: kbprg kbole kbprb
KBSubcategory: LeTwoArc

PRB: Compound File Sharing Problems on Novell Netware

PSS ID Number: Q131050
Authored 02-Jun-1995 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.03

NOTE: Novell Netware is manufactured by a vendor independent of Microsoft; we make no warranty, implied or otherwise, regarding this product's performance or reliability.

SYMPTOMS

STGM_SHARE_DENY_WRITE is ineffective when opening compound files on a network drive when the computer is running Novell Netware version 4.1 or earlier.

For example with the following code, the open will succeed the first time as expected, but it will also succeed in subsequent openings, which should not happen because of STGM_SHARE_DENY_WRITE.

```
   hr = StgOpenStorage(szFileName, NULL, STGM_TRANSACTED | STGM_READWRITE |
                STGM_SHARE_DENY_WRITE, NULL, 0, &lpStg);
```

CAUSE

This problem is caused by incorrect range locking done by Novell Netware.

RESOLUTION

This problem doesn't occur if the computer is running the Microsoft Network.

STATUS

We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: 2.0 2.00
KBCategory: kb3rdparty kbprg kbnetwork kbprb
KBSubcategory: LeTwoPst

PRB: GetData Returns Outdated Data for OLE 1.0 Object

PSS ID Number: Q110716
Authored 27-Jan-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

If an OLE 2.0 container application document contains an embedded OLE 1.0 object, and the container calls IDataObject::GetData() to retrieve data for that object, the data returned is out of date, and does not reflect the OLE 1.0 server's most recent changes to the object.

CAUSE

OLE 1.0 embedded object servers only update the cache when the user chooses the Update command from the File menu. OLE 1.0 objects do not update the cache for every change to the object. Therefore, an OLE 1.0 object's cache often contains outdated data.

RESOLUTION

The container can obtain the latest object data by calling IOleObject::Update(). This function ensures that any data or view caches maintained inside the object are up to date.

REFERENCES

For more information on known idiosyncrasies of embedding OLE 1.0 objects into OLE 2.0 containers, see "Working with OLE 1 Servers" in the OLE Software Development Kit (SDK) version 2.01 Programmer's Reference.

Additional reference words: 2.01 3.50 4.00
KBCategory: kbole kbprg kbprb
KBSubcategory: LeTwoArc

PRB: GP Fault When Using 16-Bit Automation DLLs

PSS ID Number: Q131047
Authored 02-Jun-1995 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.03

SYMPTOMS

After dynamically loading and then unloading one of the 16-bit OLE Automation DLLs (OLE2DISP.DLL, LOE2NLS.DLL, or TYPELIB.DLL), you experience a general protection (GP) fault in COMPOBJ.DLL.

CAUSE

If any of the 16-bit OLE Automation DLLs (OLE2DISP.DLL, LOE2NLS.DLL, or TYPELIB.DLL) is dynamically loaded, it should not be unloaded until after OleUninitialize has been called. Otherwise a general protection (GP) fault can occur in COMPOBJ.DLL.

RESOLUTION

If you dynamically load any of the 16-bit OLE Automation DLLs (OLE2DISP.DLL, LOE2NLS.DLL, or TYPELIB.DLL), don't unload it until OleUninitialize has been called.

STATUS

This requirement is by design.

Additional reference words: GPF
KBCategory: kbprg kbole kbprb
KBSubcategory: LeTwoAto

PRB: IOleObject::IsUpToDate() and OLE 1.0 Link Objects

PSS ID Number: Q111613
Authored 14-Feb-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

An OLE 2.0 container application calls IOleObject::IsUpToDate to determine whether or not an OLE 1.0 link object is up to date. IsUpToDate() returns S_FALSE, even though the object is actually up to date.

CAUSE

The OLE 1.0 compatibility layer always considers OLE 1.0 link objects to be out of date. If the OLE 1.0 object is not running, the compatibility layer always returns S_FALSE to IOleObject::IsUpToDate().

MORE INFORMATION

This problem can occur whenever an OLE 2.0 container application loads a document that contains links to OLE 1.0 objects. As part of loading the document, the link container updates any out of date automatic links. To find the out of date links, the container calls IOleObject::IsUpToDate() on each link that has a link-update option of OLEUPDATE_ALWAYS.

As described above, IsUpToDate will always return S_FALSE for an OLE 1.0 object that is not running. The container application will then update the link object (which requires running the object's server), even if it was not actually out of date.

For more information on the known idiosyncrasies of embedding or linking OLE 1.0 objects into OLE 2.01 containers, see "Working with OLE 1 Servers" in the OLE SDK version 2.01 "Programmer's Reference."

Additional reference words: 2.01 3.50 4.00
KBCategory: kbole
KBSubcategory: LeTwoArc

PRB: IROT::IsRunning() Returns S_FALSE for OLE 1.0 Servers

PSS ID Number: Q109544
Authored 04-Jan-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

When an OLE 2.0 container application calls IRunningObjectTable::IsRunning() on the moniker for a linked OLE 1.0 object, IsRunning() returns S_FALSE even if that OLE 1.0 object is already running.

CAUSE

OLE 1.0 objects do not register themselves in the running object table while they are running.

RESOLUTION

Calling IMoniker::IsRunning() on the moniker for a linked OLE 1.0 object correctly returns S_OK if the object is indeed running.

As noted in the OLE SDK version 2.01 documentation for IRunningObjectTable::IsRunning(), clients of a moniker should not call IRunningObjectTable::IsRunning() directly; instead, they should call IMoniker::IsRunning().

Additional reference words: 2.01 3.50 4.00 irot
KBCategory: kbole
KBSubcategory: LeTwoLnk

PRB: LoadTypeLib Does Not Register Type Library

PSS ID Number: Q131055
Authored 02-Jun-1995 Last modified 12-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.03
- Microsoft OLE libraries included with:
  - Microsoft Windows NT version 3.5
  - Microsoft Windows 95

SYMPTOMS

LoadTypeLib will not register the type library if the path of the type library is specified in the szFileName parameter. For backward compatibility, LoadTypeLib will register the type library if the path is not specified.

RESOLUTION

RegisterTypeLib should be used to register a type library.

STATUS

This behavior is by design.

Additional reference words: 2.0 2.00
KBCategory: kbprg kbprb
KBSubcategory: LeTwoAto

PRB: Menu Mnemonics Not Working During In-Place Activation

PSS ID Number: Q104460
Authored 19-Sep-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.0
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

The menu mnemonics for an object do not operate properly, causing the system to hang.

CAUSE

The server application is not calling OleTranslateAccelerator inside of its message loop.

RESOLUTION

For menu commands to be dispatched properly during in-place editing, OLE 2.0 needs to have the server application call OleTranslateAccelerator, even if the server application does not have accelerator support. This OLE application programming interface (API) does the needed translation of key messages, and ensures that the appropriate action occurs.

To be more efficient, a server application need only pass "keystroke" messages to OleTranslateAccelerator. The following code demonstrates what the message loop for an OLE server application should look like:

Sample Code

```
 // Message loop for an OLE server.
 //
 while (GetMessage(&msg, NULL, NULL))
    {
    if (m_fInplaceActive)  // If currently active in place.
        if (msg.message >= WM_KEYFIRST && msg.message <= WM_KEYLAST)
            // If it is a "keystroke" message.
            if (OleTranslateAccelerator(...) == NOERROR)
                continue; // OLE handled the message

    TranslateMessage(...);
    DispatchMessage(...);
    }
```

MORE INFORMATION

When the object's server is a stand-alone .EXE and the in-place active object gets a keystroke message that is not a recognized accelerator, the object must check to see if the message is one that the container recognizes by calling the OleTranslateAccelerator function. If the container does not want the keystroke, then the OleTranslateAccelerator function will return FALSE. In this case, the object should continue using its normal TranslateMessage and DispatchMessage code.

If the container accepts the keystroke, OLE will call the container's IOleInPlaceFrame::TranslateAccelerator member function to translate the message. The container may call the Windows TranslateAccelerator and/or TranslateMDISysAccel functions to process the accelerator key, or do its own special processing.

Additional reference words: 2.00 3.50 4.00
KBCategory: kbole kbprg kbprb
KBSubcategory: LeTwoInp

PRB: Object Appears Larger During Visual Editing

PSS ID Number: Q104791
Authored 29-Sep-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.0
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

An OLE 2.0 object appears larger during visual editing compared to the loaded state.

CAUSE

The Windows application programming interface (API) LPtoDP is being used to convert the size of the object from MM_HIMETRIC to MM_TEXT. The object is then drawn with the MM_TEXT coordinates as the bounding rectangle.

RESOLUTION

LPtoDP is dependent on the current mapping mode to perform a conversion from logical points to device points. For a more accurate transformation, a function must be written to "manually" convert the units from MM_HIMETRIC to MM_TEXT by using GetDeviceCaps to get the logical pixels per inch for use in the conversion. The OLE2UI library, included with the OLE 2.0 toolkit, already includes several transformation functions to properly convert the units:

```
   XformWidthInPixelsToHimetric  - Converts an int width into HiMetric
units
   XformWidthInHimetricToPixels  - Converts an int width from HiMetric
units
   XformHeightInPixelsToHimetric - Converts an int height into HiMetric
                                   units
   XformHeightInHimetricToPixels - Converts an int height from HiMetric
                                   units
   XformRectInPixelsToHimetric   - Converts a rect into HiMetric units
   XformRectInHimetricToPixels   - Converts a rect from HiMetric units
   XformSizeInPixelsToHimetric   - Converts a SIZEL into HiMetric units
   XformSizeInHimetricToPixels   - Converts a SIZEL from HiMetric units
```

For more information on these functions, consult the "User Interface Dialog Help" help file (OLE2UI.HLP) included with the OLE 2.0 toolkit.

Additional reference words: 2.00 3.50 4.00

KBCategory: kbole kbprg kbprb
KBSubcategory: LeTwoPrs

PRB: OleCreate Problems with Borland Compiler

PSS ID Number: Q104461
Authored 19-Sep-1993 Last modified 21-Apr-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.0

SYMPTOMS

Using the OLE 2.0 application programming interface (API), OleCreate can cause unexpected behavior and return unexpected error codes when used in an application compiled with the Borland C++ compiler.

CAUSE

At link time, the Borland file IMPORT.LIB is included before the OLE 2.0 libraries. In the IMPORT.LIB file there is a reference to the OLE 1.0 OleCreate function.

RESOLUTION

Include OLE2.LIB before IMPORT.LIB when specifying the libraries for linking.

Additional reference words: 2.00 third party third-party 3rd
KBCategory: kbole kbprg kbprb
KBSubcategory: LeTwoApp

PRB: Paste Link Option Does Not Appear in OLE 1.0 Clients

PSS ID Number: Q98680
Authored 11-May-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE version 1.0
- Microsoft OLE Libraries for Windows and Win32s, version 2.0
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SYMPTOMS

The Paste Link option remains unavailable (grayed) in an OLE version 1.0 client when an OLE version 2.0 object places its data on the clipboard.

CAUSE

The CF_OBJECTDESCRIPTOR and CF_LINKSRCDESCRIPTOR formats do not have the OLEMISC_CANLINKBYOLE1 bit set in the dwStatus member of the OBJECTDESCRIPTOR structure.

RESOLUTION

Add the OLEMISC_CANLINKBYOLE1 flag to the dwStatus member of the OBJECTDESCRIPTOR structure.

MORE INFORMATION

OLE 1.0 clients do not support links to embedded objects. OLE 2.0 containers that support links to embedded objects should not set this flag if the object being placed on the clipboard could result in a link to an embedded object. Removing this flag prevents an OLE 1.0 client from attempting to make a link in this scenario.

Additional reference words: 1.00 2.00 3.50 4.00
KBCategory: kbole kbprg kbprb
KBSubcategory: LeTwoDdc

PRB: Problem with ScrollBar Control While Activated In-place

PSS ID Number: Q108942
Authored 20-Dec-1993 Last modified 21-Apr-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, versions 2.0 and 2.01

SUMMARY

It is not possible to use a ScrollBar child control inside a window that will be used as the in-place window in a container application.

SYMPTOMS

The ScrollBar control does not generate WM_VSCROLL or WM_HSCROLL messages to its parent window.

CAUSE

One way to understand this problem is to examine the case where you have an OLE container application that inserts an in-place capable object into itself using the Insert Object menu option.

When the object is created, and if the container application allows the object to embed itself in-place in the container application, it will set the parent of its editing window to be a window that belongs to the container.

Therefore, a situation occurs where two applications are running; that is, the container and the server. The server application created and owns the window procedure for a window that the object lives in; however, the parent of this window is a window of the container application.

Imagine now that the object's editing window (created by the server and parented by the container) contains a ScrollBar child control. When the user clicks the ScrollBar control by using the left mouse button, a WM_LBUTTONDOWN message is generated that gets handled internally by Windows's ScrollBar child control code.

This code makes a check to ensure that the message queue that has just delivered this message is also owned by the current active application. If they are not, the message will not be processed further and scroll bar messages will not be sent to the ScrollBar control's parent.

This is the case with the in-place scenario described above, and the root of the problem.

The active application is the container application but the message queue that delivered the scroll bar message belongs to the server application.

Also note that this problem does not apply to in-place editing windows that have scroll bars that were created using the WS_HSCROLL or WS_VSCROLL style. The message processing for these scroll bars is slightly different from ScrollBar child controls.

RESOLUTION

Instead of creating a child control with the "scrollbar" class, create a new window class (for example, MYSCROLL) whose window procedure forwards WM_VSCROLL and WM_HSCROLL messages to its parent.

Then you can create windows of this class with either WS_HSCROLL or WS_VSCROLL. Then position these windows the same way you would position ScrollBar controls. When you enable/disable the scroll bar, use the SB_HORZ, SB_VERT flags in the call to EnableScrollBar.

NOTE: You must have one row of pixels of the little window's client area visible or Windows will hide the horizontal scroll bar. This does not apply to the vertical scroll bar; that is, you do not need to have a column of pixels visible.

Additional reference words: 2.00 2.01
KBCategory: kbole kbprg kbprb
KBSubcategory: LeTwoApp

PRB: Property or Method Name Not Recognized on Some Machines

PSS ID Number: Q131053
Authored 02-Jun-1995 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.03

SYMPTOMS

If the sLangage and iCountry fields in the [intl] section of WIN.INI are missing or have incorrect entries, IDispatch::GetIDsOfNames fails (it returns the DISP_E_UNKNOWNNAME error) if it is implemented using DispGetIDsOfNames or CreateStdDispatch and a typeinfo created using an INTERFACEDATA structure.

CAUSE

If an automation server's IDispatch is implemented using a typeinfo created from the INTERFACEDATA structure, access of properties or methods may fail on some computers. These computers may have missing or incorrect sLanguage and iCountry fields in the [intl] section of WIN.INI. These fields are used by DispGetIDsOfNames and CreateStdDispatch's IDispatch::GetIDsOfNames to obtain locale information. These functions fail (DISP_E_UNKNOWNNAME) if the locale information cannot be obtained.

RESOLUTION

To solve this problem, modify the fields in WIN.INI to look similar to this:

```
[intl]
sLanguage=enu
iCountry=1
```

Where:

- enu means U.S. English.
- iCountry specifies the country code. This number matches the country's international telephone code, except for Canada, which is 2. The default is 1.

You can also change these fields by using the International Control Panel applet by selecting the appropriate Country and Language.

STATUS

This behavior is by design. This problem doesn't occur with typeinfos obtained from type libraries. Microsoft strongly recommends that type

libraries be used for the implementation of IDispatch. The INTERFACEDATA approach was a temporary solution to create typeinfos while type libraries were in Beta.

Additional reference words: 2.0 2.00
KBCategory: kbprg kbole kbprb
KBSubcategory: LeTwoAto

PRB: Some OLE Containers Do Not Call IViewObject::Draw

PSS ID Number: Q131155
Authored 04-Jun-1995 Last modified 07-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5
- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries included with:
  - Microsoft Windows NT version 3.5
  - Microsoft Windows 95

SYMPTOMS

A custom OLE rendering handler finds that its IViewObject::Draw method is never called by some OLE container applications.

CAUSE

Some OLE container applications never call IViewObject::Draw. These applications do not use the caching and drawing services provided by the object handler. Instead, they do their own caching and drawing of the presentations for OLE objects embedded in their documents.

Two examples of such container applications are Microsoft Word version 6.0 and Microsoft Excel version 5.0. They cache metafile representations of embedded OLE objects, and then do their own drawing of those metafiles.

STATUS

This behavior is by design. It is not possible to get Microsoft Word or Microsoft Excel to use a handler's drawing services.

Additional reference words: 2.01 3.50 4.00 5.00 6.00
KBCategory: kbole kbprb
KBSubcategory: LeTwoOh

PRB: Synchronous OLE Call Fails If in Inter-task SendMessage

PSS ID Number: Q131056
Authored 02-Jun-1995 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.03
- Microsoft OLE libraries included with:
  - Microsoft Windows NT version 3.5
  - Microsoft Windows 95

SYMPTOMS

A synchronous OLE call made by the recipient of an inter-task SendMessage fails.

CAUSE

The cause is discussed in detail in the "More Information" section of this article.

RESOLUTION

Use PostMessage instead of an inter-task SendMessage.

STATUS

This behavior is by design.

MORE INFORMATION

See the beginning of chapter 13 in the OLE 2 Programmer's Reference Volume 1 for the categories of OLE calls. An understanding of these categories is required for this article.

The majority of OLE calls are synchronous calls. A synchronous call to a different process yields to that process and waits for a reply from that process. In addition, OLE has input-synchronized calls that relate to the inplace-activation interfaces. Input-synchronized calls are implemented using an inter-task SendMessage.

16-bit Windows doesn't allow a task to yield while in an inter-task SendMessage because a system deadlock could occur. The deadlock occurs because a message for the sender could be present at the top of the shared system queue, and this prevents other tasks, including the recipient of the SendMessage, from retrieving their messages from the

system queue until the sender does. The sender cannot retrieve its message because it is waiting for the inter-task SendMessage to return.

In 32-bit Windows, each process has its own system queue and this architecture normally prevents deadlock problem from occuring. However, when one process is inplace active in another process's window, the system queues of the two processes are synchronized as in 16-bit windows, so the deadlock could occur. To prevent this, OLE stops synchronous OLE calls from being made while the caller is the recipient of an input-synchronized call.

OLE determines if the caller of the synchronous call is a recipient of an input-synchronized call by using the InSendMessage() API. This broad check prevents a synchronous call from being made if the caller is currently a recipient of any inter-task SendMessage.

Additional reference words: 2.0 2.00
KBCategory: kbprg kbole kbprb
KBSubcategory: LeTwoArc

Properties with Optional Parameters Not Supported

PSS ID Number: Q131048
Authored 02-Jun-1995 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.03
- Microsoft OLE libraries included with:
    - Microsoft Windows NT version 3.5
    - Microsoft Windows 95

Properties with optional parameters are currently not supported by the IDispatch::Invoke implementation provided by the OLE system (DispInvoke and CreateStdDispatch). For example, the following results in an error from MKTYPLIB.EXE:

```
[propput]
void MyProperty([in] BSTR bstrValue, [in, optional] VARIANT vIndex,);
[propget]
BSTR MyProperty([in, optional] VARIANT vIndex);
```

Properties with non-optional parameters are allowed. For example, the following is acceptable:

```
[propput] void MyProperty([in] short nIndex, [in] BSTR bstrValue);
[propget] BSTR MyProperty([in] short nIndex);
```

Additional reference words: 2.0 2.00 3.50 4.00
KBCategory: kbprg kbole
KBSubcategory: LeTwoAto

Reference Counting Rules

PSS ID Number: Q104138
Authored 08-Sep-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.0
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SUMMARY

In the component object model, an interface's lifetime is controlled through reference counting. The reference count for an interface is manipulated through the AddRef() and Release() member functions inherited from IUnknown. The AddRef() member increments an interface's reference count, and the Release() method decrements it. Once an interface's reference count goes to zero, there are no longer any valid pointers to that interface. If the reference count on all of an object's interfaces is zero, then the object can be freed because there are no longer any pointers to the object.

MORE INFORMATION

Reference Counting Rules

The following list is a copy of the reference counting rules (taken from pages 83 and 84 of the OLE 2.0 specification) that must be followed. Small code samples have been added in this article to help clarify the rules.

1. Every new copy of an interface pointer must be AddRef()'d, and every destruction of an interface pointer must be Release()'d except where subsequent rules explicitly permit otherwise.

   a. In-Out parameters to functions: The caller must AddRef() the actual parameter, because it will be Release()'d by the callee when the out-value is stored on top of it.

```
LPOLEOBJECT lpObject;
   .
   .  // Get pointer to IOleObject.
   .
LPVIEWOBJECT lpView = lpObject;

lpObject->AddRef()

// GetViewObject is a theoretical function that takes a
// pointer to anything derived from IUnknown, and then
```

```
   // returns a pointer to IViewObject in the same variable
   // passed as the parameter. The AddRef() above is needed so
   // that the original pointer to IOleObject is not freed.

   GetViewObject(lpView);
```

b. Fetching a global variable: The local copy of an interface pointer fetched from an existing copy of the pointer in a global variable must be independently reference counted because called functions might destroy the copy in the global while the local copy is still alive.

```
   void function()
   {
   // Get a pointer to IOleObject from a global variable.
   LPOLEOBJECT lpOleObject = glpObject;

   // This AddRef() is needed so that the interface
   // pointed to by the global variable, glpObject,
   // does not get released by a different part of
   // the applications code.

   lpOleObject->AddRef();
      .
      . // use lpOleObject;
      .
   lpOleObject->Release();
   }
```

c. New pointers synthesized out of "thin air": A function that synthesizes an interface pointer using special internal knowledge rather than obtaining it from some other source must do an initial AddRef() on the newly synthesized pointer. Important examples of such routines include instance creation routines, implementations of IUnknown::QueryInterface, and so forth.

```
   STDMETHDOIMP IUnknown::QueryInteface( REFIID iidInterface,
                                         LPVOID FAR *ppvObj)
   {
   *ppvObj = NULL;
   SCODE sc = E_NOINTERFACE;

   if (iidInterface == IUnknown)
       {
       *ppvObj = this;

       // This AddRef() is needed because a new pointer
       // was just created.

       AddRef();
      sc = S_OK;
       }

   return ResultFromScode(sc);
```

```
}
```

d. Returning a copy of an internally stored pointer: Once the pointer has been returned, the callee has no idea how its lifetime relates to that of the internally stored copy of the pointer. Thus, the callee must AddRef() the pointer copy before returning to it.

```
// m_lpOleObject is a private member variable of a C++ class.
// GetOleObject is a member function to return access to this
// pointer.

void GetOleObject (LPVOID FAR *ppObject)
{
    *ppObject = m_lpOleObject;

    // This AddRef() is needed due to this rule.

    m_lpOleObject->AddRef();
 }
```

2. Special knowledge on the part of a piece of code about the relationships of the beginnings and endings of the lifetimes of two or more copies of an interface pointer can allow AddRef()/Release() pairs to be omitted.

a. In-parameters to functions: The copy of an interface pointer that is passed as an actual parameter to a function has a lifetime that is nested in that of the pointer used to initialize the value. Therefore, the actual parameter need not be separately reference counted.

```
void function (LPOLEOBJECT lpOleObject)
{

// Can use lpOleObject in this function
// without doing AddRef() and Release().

}
```

b. Out-parameters from functions, including return values: To set the out parameter, the function itself by Rule 1 must have a stable copy of the interface pointer. On exit, the responsibility for releasing the pointer is transferred from the callee to the caller. Thus, the out parameter need not be referenced counted.

```
LPVIEWOBJECT lpView;

HERROR hErr = lpOleObject->QueryInterface(IID_IViewObject,
                                          (LPVOID FAR *)lpView);

if (hErr = NOERROR)
    {
    // The QueryInterface succeeded. lpView does not have
```

```
        // to be AddRef()'d because it has already been done
        // by the QueryInterface method.
        }
```

c. Local variables: A function implementation clearly has omniscient knowledge of the lifetimes of each of the pointer variables allocated on the stack frame. It can therefore use this knowledge to omit redundant AddRef()/Release() pairs.

```
void function()
{
LPOLEOBJECT lpTempObject;
   .
   .
   .
lpTempObject = lpObject;
   .
   .  // lpTempObject can be used
   .  // without reference counting as long as
   .  // it is known that the lifetime of lpObject
   .  // outside of this function call.
   .
}
```

d. Backpointers. Some data structures are of the nature of containing two components, A and B, each with a pointer to the other. If the lifetime of one component (A) is known to contain the lifetime of the other (B), then the pointer from the second component back to the first (from B to A) need not be reference counted. Often, avoiding the cycle that would otherwise be created is important in maintaining the appropriate freeing behavior.

Additional reference words: 2.00 3.50 4.00
KBCategory: kbole kbprg
KBSubcategory: LeTwoCom

Returning Floats and Doubles from Automation methods

PSS ID Number: Q122286
Authored 01-Nov-1994 Last modified 21-Apr-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.02

SUMMARY

Automation properties or methods implemented with the C calling convention in the Windows 16-bit operating system can't return type Float or Double. This includes the Date type, which is a floating-point type. To work around this, you can implement them using the Pascal calling convention.

MORE INFORMATION

Automation methods and properties can be marked as using one of the calling conventions described by the CALLCONV enumeration. This calling convention is used by the standard IDispatch implementation provided by CreateStdDispatch and by DispInvoke to call property accessor functions or methods. Note that the an automation controller does not directly use these calling conventions.

Because of a C Language implementation limitation, automation properties and methods that return Float or Double cannot use the C calling convention. You need to use the Pascal calling convention.

For example, describe a method or automation property that returns a Float or Double as follows in the .ODL file:

```
[propget] float pascal MyProperty();
float pascal MyMethod();
```

Additional reference words: 2.00
KBCategory: kbole kbprg
KBSubcategory: LeTwoAto

Rules for Freeing BSTRs in OLE Automation

PSS ID Number: Q108934
Authored 20-Dec-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SUMMARY

The callee frees a BSTR passed in as a by-reference parameter before assigning the parameter a new value. In all other cases, the caller frees the BSTR.

MORE INFORMATION

OLE Automation defines the BSTR data type to handle strings that are allocated by one component and freed by another. The rule for freeing a BSTR is as follows:

The callee frees a BSTR passed in as a by-reference parameter before assigning the parameter a new value. In all other cases, the caller frees the BSTR.

In other words, the caller handles BSTRs as follows:

1. Free BSTR returned by called function or returned through a by-reference parameter of called function.

2. Free BSTR passed as a by-value parameter to a function.

The callee handles BSTRs as follows:

1. Free BSTR passed in as a by-reference parameter before assigning a new value to the parameter. Do not free if a new value is not assigned to the by-reference parameter.

2. Do not free BSTR passed in as a by-value parameter.

3. Do not free BSTR that has been returned to caller.

4. If a BSTR passed in by the caller is to be stored by the callee, store a copy using SysAllocString(). The caller will release the BSTR that it passed in.

5. If a stored BSTR is to be returned, return a copy of the BSTR. The caller will release the BSTR that is returned.

Additional reference words: 2.01 3.50 4.00
KBCategory: kbole kbprg
KBSubcategory: LeTwoAto

SAMPLE: BINARY: Transfer Binary Data Using OLE Automation

PSS ID Number: Q131046
Authored 02-Jun-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.03
- Microsoft OLE libraries included with:

    - Microsoft Windows NT version 3.5
    - Microsoft Windows 95

BINARY is an OLE Automation server that you can obtain from the Microsoft Software Library. It demonstrates how to transfer binary data using a SAFEARRAY of unsigned char. The CTRL directory in this sample contains an automation controller that will obtain the binary data from the server.

Binary data can be transferred with OLE Automation by using a SAFEARRAY of unsigned char (VT_ARRAY|VT_UI1). Binary data can also be transferred by passing the IUnknown of a data trasfer object that supports IDataObject. This sample uses the first approach, not the second.

Standard controllers like Visual Basic cannot access binary data transferred through automation. For more information about this, please see the following article in the Microsoft Knowledge Base:

   ARTICLE-ID: Q122289
   TITLE : Passing Structures in OLE Automation

See the README.TXT in the sample for details on how to compile and run the sample. After you download BINARY.EXE, use the following command to run it:

    BINARY.EXE -d

This builds the directory structure for you.

Download BINARY.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
      GO MSL
      Search for BINARY.EXE
      Display results and download

- Microsoft Download Service (MSDL)
      Dial (206) 936-6735 to connect to MSDL
      Download BINARY.EXE

- Internet (anonymous FTP)
      ftp ftp.microsoft.com
      Change to the \SOFTLIB\MSLFILES directory
      Get BINARY.EXE

Additional reference words: 3.50 4.00
KBCategory: kbole kbprg kbcode kbfile
KBSubcategory: LeTwoAt

SAMPLE: DECODE16: OLE Error Code Decoder Tool

PSS ID Number: Q122956
Authored 16-Nov-1994 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.02

DECODE16 is a sample decoder of 16-bit OLE version 2.02 error codes. See the DECODE32 sample for a decoder of 32-bit OLE error codes. Enter the HRESULT in hexadecimal format, and press the Decode button.

Download DECODE16.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for DECODE16.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download DECODE16.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get DECODE16.EXE

Download DECODE32.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for DECODE32.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download DECODE32.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get DECODE32.EXE

Additional reference words: 2.02
KBCategory: kbole kbcode kbfile
KBSubcategory: LeTwoTls

SAMPLE: DECODE32: OLE Error Code Decoder Tool

PSS ID Number: Q122957
Authored 16-Nov-1994 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries, version 2.1, included with:
  - Microsoft Windows NT, version 3.5

DECODE32 is a sample decoder for 32-bit OLE version error codes. See the DECODE16 sample for a decoder for 16-bit OLE error codes. In addition to decoding the HRESULT, DECODE32 also uses FormatMessage() to provide a description of the error. Enter the HRESULT in hexadecimal format and press the Decode button.

Download DECODE32.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for DECODE32.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download DECODE32.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get DECODE32.EXE

Download DECODE16.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for DECODE16.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download DECODE16.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get DECODE16.EXE

Additional reference words: HRESULT SCODE DECODE
KBCategory: kbole kbcode kbfile
KBSubcategory: LeTwoTls

SAMPLE: DRGDRPS: OLE Drag-Drop Source

PSS ID Number: Q122955
Authored 16-Nov-1994 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.02
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

OLE version 2.02 and later provide drag-drop functionality that allows data to be dragged between and within applications. DRGDRPS is a sample that provides an OLE drag-drop Source that allows data of format CF_TEXT to be dragged from it.

The DRGDRPS sample implements IDataObject, IDropSource, and IEnumFormatEtc interfaces. It calls DoDragDrop when the user clicks on its main window to initiate a drag. It allows copying of data by dragging to a drag-drop target.

Download DRGDRPS.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for DRGDRPS.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download DRGDRPS.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get DRGDRPS.EXE

See the DRGDRPT sample for an example of an OLE drag-drop target. Download DRGDRPT.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for DRGDRPT.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download DRGDRPT.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com

```
      Change to the \SOFTLIB\MSLFILES directory
      Get DRGDRPT.EXE
```

See the SIMPDND sample in the OLE2 SDK for a more complex drag-drop example.

Additional reference words: 2.02 3.50 4.00
KBCategory: kbole kbcode kbfile
KBSubcategory: LeTwoDdc

SAMPLE: DRGDRPT: OLE Drag-Drop Target

PSS ID Number: Q122954
Authored 16-Nov-1994 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.02
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

OLE version 2.0 provides drag-drop functionality that allows data to be dragged between and within applications. DRGDRPT is a sample application for an OLE drag-drop target that allows data of format CF_TEXT to be dropped on it. DRGDRPT implements the IDropTarget interface. It registers its main window as a drop target using RegisterDragDrop.

Download DRGDRPT.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for DRGDRPT.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download DRGDRPT.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get DRGDRPT.EXE

For an example of an OLE drag-drop source, see the DRGDRPS code sample. Download DRGDRPS.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for DRGDRPS.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download DRGDRPS.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get DRGDRPS.EXE

See the SIMPDND sample in the OLE2 SDK for a more complex drag-drop

example.

Additional reference words: 2.02 3.50 4.00
KBCategory: kbole kbcode kbfile
KBSubcategory: LeTwoDdc

SAMPLE: MFCINP16: Inproc 16-bit MFC Automation Object

PSS ID Number: Q130843
Authored 30-May-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft OLE version 2.03
- Microsoft Foundation Classes (MFC) included with Microsoft Visual C++ for Windows, version 1.52

SUMMARY

MFC's App Wizard will not generate an inproc OLE Automation server. This article gives you the steps to follow to create an inproc automation server using MFC. MFCINP16, a sample available in the Microsoft Software Library, is a 16-bit inproc automation object that was created using the steps in this article.

Download MFCINP16.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for MFCINP16.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download MFCINP16.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get MFCINP16.EXE

MORE INFORMATION

MFC's App Wizard doesn't generate inproc (DLL) OLE servers because it is not possible for MFC to fully implement one that can open into a separate window. MFC needs to hook into the client's main message retrieval loop to translate accelerators of the separate window and to implement idle-time processing. OLE doesn't provide such a mechansim. It is entirely possible to implement an inproc server in MFC with no user interface or with a very simple user interface. This article shows you how.

Steps to Create a 16-Bit Inproc Automation Object in MFC

It is assumed that Visual C++ version 1.51 or greater and the Control Development Kit have been installed.

1. Use INPROC.CPP, INPROC.H, and STDAFX.H from the sample as the starting point. Copy these into a new directory. The names INPROC.CPP and INPROC.H can be changed to something more appropriate for your project.

2. Create a new project by choosing New from the Project menu and selecting the Windows dynamic-link library as the type. Add INPROC.CPP to this project. (The sample uses inproc for the project name.)

3. From the Options menu, choose Project and select Compiler. Add the following in the C/C++ CompilerOptions dialog box under the CustomOptions category in the OtherOptions edit control:

   /D "_USRDLL"

   From the Options menu, choose Project and select Linker. Add the following libraries in the LinkerOptions dialog box under the Input category in the Libraries edit control:

   ole2, compobj, ole2disp, typelib, ole2nls, mfcoleui, storage

4. Choose ClassWizard from the Project menu. ClassWizard will complain that the .CLW file does not exist. It will ask you to rebuild the .CLW file by opening the project in App Studio and running ClassWizard.

5. Open AppStudio by choosing it from the Tools menu. Save the resource file, and run ClassWizard from AppStudio by choosing ClassWizard from the Resource menu. The CLW file will now be built after you choose OK in the SelectSourceFiles dialog box. Complet the following steps:

   a. Select the OLE Automation tab in the MFC ClassWizard dialog.

   b. Choose the AddClass button, and add a class of type CCmdTarget.

   c. Select the OLEAutomation check box.

   d. Select the OLECreatable check box, and provide an ExternalName (progID) if this is a top-level automation object. This external name will be used by the automation controller/client to create the object. (The sample creates a class called TestObject, which has an external name Inproc.TestObject.)

6. Add the required automation properties and methods to the newly created class. (The sample creates a method called TestMethod that returns void and has no parameters. The method calls MessageBeep.)

7. Attempt to build the project. Visual C++ will ask if you want to create a default .DEF file. Edit the default .DEF file to export the following:

   DllGetClassObject
   DllCanUnloadNow
   DllRegisterServer

8. Build the project. Register the inproc Automation object by choosing RegisterControl from the Tools menu.

9. VB.MAK and VB.FRM in the sample are Visual Basic version 3.0 files that you can use to control the object.

Additional reference words: 2.03 1.52
KBCategory: kbole kbprg kbfile
KBSubcategory: LeTwoAto

SAMPLE: MFCINP32 Inproc 32-bit MFC Automation Object

PSS ID Number: Q130842
Authored 30-May-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft OLE version 2.03
- Microsoft Foundation Classes (MFC) included with:
  Microsoft Visual C++, 32-bit Edition, version 2.0

SUMMARY

MFC's App Wizard in Visual C++ version 2.0 will not generate an inproc OLE Automation server. This article gives steps you can follow to create an inproc automation server using MFC. You can also obtain a sample (MFCINP32) from the Microsoft Software Library that is a 32-bit inproc automation object created by following the steps in this article.

Download MFCINP32.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for MFCINP32.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download MFCINP32.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get MFCINP32.EXE

NOTE: The App Wizard in Visual C++ version 2.1 and above does generate an inproc OLE Automation server.

MORE INFORMATION

MFC's App Wizard doesn't generate inproc (DLL) OLE servers because it is not possible for MFC to fully implement one that can open into a separate window. MFC needs to hook into the client's main message retrieval loop to translate accelerators of the separate window and to implement idle-time processing. OLE doesn't provide such a mechansim. It is entirely possible to implement an inproc server in MFC with no user interface or with a very simple user interface. This article provides the instructions to do this.

Steps to Create 32-Bit Inproc Automation Object in MFC

It is assumed that Visual C++ version 2.0 and the Control Development Kit have been installed.

1. Use INPROC.CPP, INPROC.H, and STDAFX.H from the sample as the starting point. Copy these into a new directory. The names INPROC.CPP and INPROC.H can be changed to something more appropriate for your project.

2. Create a new project of type Dynamic-Link Library. (Do not select MFC AppWizard DLL.) Add INPROC.CPP to this project.

3. Create an .ODL file (INPROC.ODL) containing a modification of the following code, and add it to the project.

```
[ uuid(ABEBE5A0-0C69-11CE-B774-00DD01103DE1), version(1.0) ]
library inproc
{
     importlib("stdole32.tlb");
    //{{AFX_APPEND_ODL}}
};
```

   Don't use the same UUID as the one shown here. Instead generate a new one by running GUIDGEN.EXE, and use that value. The library name can be changed from inproc to a name more appropriate for your project. The version number can also be changed.

4. Create the .DEF file (INPROC.DEF) containing the following code, and add it to the project.

```
LIBRARY   INPROC
EXPORTS
          DllGetClassObject
          DllCanUnloadNow
          DllRegisterServer
```

   The library name can be changed from INPROC to a name more appropriate for your project.

5. Select Project Settings, as follows:

   - In the ProjectSettings dialog box under General, select Use MFC in a shared DLL.

   - In the ProjectSettings dialog box under C/C++, remove _AFXDLL, and add _USRDLL and _WINDLL under Preprocessor definitions.

   - In the ProjectSettings dialog under Link, add the following libraries to Object/Library modules:

     ole32.lib oleaut32.lib

6. Choose ClassWizard from the Project menu. ClassWizard will complain that the .CLW file does not exist. It will ask you to rebuild the .CLW file by opening the .RC file and running ClassWizard again.

7. Open your .RC file or create a new .RC file by choosing New from the

File menu and selecting Resource Script. Save the new .RC file (INPROC.RC). Now bring up ClassWizard. The .CLW file will now be built after you choose OK in the SelectSourceFiles dialog box. Follow these steps:

a. Select the OLE Automation tab in the ClassWizard dialog.

b. Choose the AddClass button, and add a class of type CCmdTarget.

c. Select the OLEAutomation check box.

d. Select the OLECreatable check box, and provide an ExternalName (progID) if this is a top-level automation object. This external name is used by the automation controller/client to create the object. (The sample creates a class called TestObject that has an external name Inproc.TestObject.)

8. Add the required automation properties and methods to the newly created class. (The sample creates a method called TestMethod that returns void and has no parameters. The method calls MessageBeep.)

9. Build the project. Register the inproc Automation object by using the Tools/RegisterControl menu.

10. Note that a 16 bit controller like Visual Basic version 3.0 cannot control a 32-bit inproc automation object because 16-bit to 32-bit interoperability is not supported with inproc automation objects. Instead, write a 32-bit controller to control this 32-bit inproc automation object.

Additional reference words: 2.00 3.00
KBCategory: kbole kbprg kbfile kbcode
KBSubcategory: LeTwoAto

SAMPLE: Multilingual OLE Automation Object

PSS ID Number: Q107698
Authored 24-Nov-1993 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.0
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SUMMARY

MULTLING demonstrates how to create an OLE automation object that supports multiple languages. This allows the controller of an automation object to access properties and methods using any of the languages that are supported.

Download MULTLING.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for MULTLING.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download MULTLING.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \softlib\mslfiles directory
  Get MULTLING.EXE

MORE INFORMATION

This multilingual OLE automation object sample checks the Locale ID (LCID) passed methods of the IDispatch interface to determine the language being used by the automation controller. The object supports access of properties and methods in English, French, and German.

One Type Library Per Language

The automation object registers three different type libraries in the registration database--one for each supported language. The type libraries have the same UUIDs but different locale attributes. Each type library is loaded at object creation and the ITypeInfo interface is obtained from each as follows (see LoadTypeInfo in MAIN.CPP):

```
   LoadRegTypeLib(LIBID_Hello, 1, 0, lcid, &ptlib);
   ptlib->GetTypeInfoOfGuid(IID_IHello, &ptinfo);
```

Also see ENGLISH.ODL, FRENCH.ODL, GERMAN.ODL, and HELLO.REG.

Interpret LCID in IDispatch Methods

The implementation of IDispatch::GetTypeInfo, GetIDsOfNames, and Invoke checks the value of the lcid parameter to determine the locale ID and uses the appropriate ITypeInfo for that language (See CHello::GetTypeInfo, GetIDsOfNames, and Invoke in HELLO.CPP).

To Run

The multilingual automation object exposes one VT_BSTR property (HelloMessage) and one method (SayHello).

ProgID : HelloMultiLingual.Hello

Method and Property Names:

English	French	German	Action
HelloMessage	SalutMessage	HalloNachricht	Sets or gets the HelloMessage string.
SayHello	DitSalut	SagHallo	Displays the HelloMessage in an edit control.

Use the AUTOCTRL sample to control the multilingual automation object. The AUTOCTRL automation controller allows the locale ID to be specified.

Update the path in HELLO.REG to the current location of the object and the type libraries.

To Compile

Requires OLE 2.01 or later.

Include device=vmb.386 in the [386Enh] section of SYSTEM.INI.
Note that vmb.386 can be found in \OLE2\BIN.
Run the WXSERVER.EXE from \OLE2\BIN before running the makefile.

Additional reference words: 2.00 3.50 4.00 multi-lingual softlib MULTLING.EXE
KBCategory: kbole kbfile
KBSubcategory: LeTwoAto

SAMPLE: OLE Automation '94 Documentation and Samples

PSS ID Number: Q124385
Authored 29-Dec-1994 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.02

SUMMARY

You can get a draft of the second edition of OLE 2 Programmer's Reference, Volume 2 and the samples it contains from two files (OA94DOC.EXE and OA94SAMP.EXE). The final version will be released in 1995. The samples demonstrate many of the new Automation features.

IMPORTANT: After downloading OA94SAMP.EXE, run it in an empty directory using the -d option to maintain the directory structure:

```
OA94SAMP.EXE -d
```

Download OA94DOC.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for OA94DOC.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download OA94DOC.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get OA94DOC.EXE

Download OA94SAMP.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for OA94SAMP.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download OA94SAMP.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory

Get OA94SAMP.EXE

MORE INFORMATION

OLE Automation '94 is the version of OLE Automation that shipped with version 2.02 of the OLE libraries.

Contents for OA94DOC

The OA94DOC.EXE self-extracting file contains a draft of the second edition of OLE 2 Programmer's Reference, Volume 2. It documents the following new features introduced in OLE Automation '94:

- VTBL binding and dual interfaces.
- Error handling mechanisms for interfaces that support vtable binding.
- New registration flags for RegisterActiveObject().
- New ODL Attributes.
- Multiple type libraries can now be stored in an executable (.EXE) file or DLL. These libraries can be loaded using LoadTypeLibFromResource.
- MkTypLib includes DEFINE_GUID macros in the generated header file for each element (interface, dispinterface, and so forth) defined in the type library.
- SafeArrayPtrOfIndex().
- VT_UI1 data type.
- Additions to ITypeInfo Structures and Enumerations.
- Additions to National Language Support LCTYPE constants in 16 bit.

You will fild a discussion of the new features in TECHNOTE.DOC, which is included in OA94DOC.EXE.

Contents for OA94SAMP

The OA94SAMP.EXE self-extracting file contains the samples referenced in OA94DOC. The following samples are included:

- HELLO - A simple automation object that implements a dual interface.
- HELLCTRL - A simple automation controller that controls HELLO using vtable binding.
- LINES - A more complex automation object that implements several dual interfaces and two collections.
- BROWSEH - (Browse Helper) An inproc (DLL) automation object that exposes properties and methods that allow easy browsing of a type library. This sample doesn't implement dual interfaces.
- BROWSE - An automation controller that uses late-binding to control BROWSEH. It displays the type library information obtained from BROWSEH. The INVHELP.CPP file in this sample contains helper functions that allow easy creation of any late-binding automation controller.

The WIN16 directory contains 16-bit versions of the samples, and the WIN32 directory contains a version that you can compile for 16-bit or 32-bit by

changing the dev option in the makefile to compile for 16- or 32-bit and changing the HOST option to compile a 32-bit version for Windows NT or for Windows 95. The .REG entries for the samples in the WIN32 directory are set up for 32-bit, so you need to change them to the 16-bit keys if you are compiling for the 16-bit environment.

The 32-bit version of the samples for NT use Unicode.

The 32-bit version of the samples for Windows 95 use Unicode with the OLE API and interfaces and use ANSI with the Windows 95 API. This is because Windows 95 uses ANSI and 32-bit OLE uses Unicode, regardless of the platform. The samples do the appropriate Unicode-to-ANSI conversions to support this.

See the README.TXT file for each sample for more details.

Additional reference words:
KBCategory: kbole kbprg kbcode kbfile
KBSubcategory: LeTwoAto

SAMPLE: OLE Automation Collection

PSS ID Number: Q107546
Authored 23-Nov-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.0
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SUMMARY

COLLECT demonstrates how to create an OLE automation collection object.

COLLECT can be found in the Software/Data Library by searching on the word COLLECT, the Q number of this article, or S14424. COLLECT was archived using the PKware file-compression utility.

MORE INFORMATION

The collection object implements the following methods and properties: Add Method, Count Property, Item Property, NewEnum Property, and Remove Method.

Error handling in the collection object involves raising exceptions. CreateStdDispatch does not allow exceptions to be raised, so the collection object implements the IDispatch interface using DispGetIDsOfNames and DispInvoke.

Add Method

Description:	Adds the indicated item to the collection. If an object is created, it should be returned.
Arguments:	Varies. It can be a pointer to the object that is to be added or it can the information required to create the object.
Return type:	Varies. If no object is created, the return type should be void. If an object is created, the return type should be VT_DISPATCH.

Note: The Add method is not appropriate for all collections, so it is not required. For many application-created collections, objects are automatically added to the collection for the user.

Count Property

Description:	Returns the number of items in the collection. Read-only property.
Arguments:	None.

Return type: VT_I4.

Item Property

Description: Returns the indicated item in the collection.
Argument: Specifies the index. Some collections allow various types of indexing. For example, this sample allows an integer or string to be specified as an index.
Return type: VT_DISPATCH.

Note: Item is the default value for the object, so it should have the special DISPID, DISPID_VALUE. MkTypLib automatically assigns this DISPID if the default attribute is specified in the ODL file.

_NewEnum Property

Description: Returns an enumerator that supports IEnumVariant for the items currently in the collection. Read-only property.
Arguments: None.
Return type: VT_UNKNOWN.

Note: NewEnum will not be accessible to users and must have the restricted attribute in the ODL file. The NewEnum method must have a special DISPID, DISPID_NEWENUM. The defining characteristic of a collection is the ability for a user to iterate over the items in it. Some languages will have built-in support for collections. The NewEnum method allows an OLE automation controller to support "for each" iteration over a collection:

```
For Each Item In Collection
Debug.Print Item.Text
Next Item
```

OLE automation controllers that support "for each" iteration will call the NewEnum method on the collection object and then QueryInterface on the resulting IUnknown to get the desired IEnumVariant.

Remove Method

Description: Removes the specified item from the collection.
Argument: Specifies the index. Some collections allow various types of indexing. For example, this sample allows an integer or string to be specified as an index.
Return type: void.

Note: The object is not deleted. It is simply removed from the collection. Remove should support the same kinds of indexing as the Item() method for the same collection. The Remove method is not appropriate for all collections, so it is not required. For many application-created collections, objects are automatically removed the collection for the user.

To Run

The collection sample application object exposes the following:

```
   ProgID :  Collection.Application

   Method and Property Names     Notes
   ------------------------------------------------------------
   [App Object]
   Collection  (read only prop)  Returns empty collection.
   New Item    (method)          Creates and returns new item.

   [Item Object]
   Text        (default prop)    Sets and returns string.

   [Collection Object]
   See above for exposed properties and methods.
```

Update the path in HELLO.REG to the current location of the object and the type libraries.

To Compile

Requires OLE version 2.01 or later.

Include device=vmb.386 in the [386Enh] section of SYSTEM.INI.
Note that vmb.386 can be found in \OLE2\BIN.
Run WXSERVER.EXE from \OLE2\BIN before running the makefile.

Additional reference words: 2.00 3.50 4.00 softlib COLLECT.EXE
KBCategory: kbole kbfile
KBSubcategory: LeTwoAto

SAMPLE: OLE Automation Controller Sample

PSS ID Number: Q106080
Authored 31-Oct-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.0
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SUMMARY

AUTOCTRL demonstrates how to create an OLE automation controller.

AUTOCTRL can be found in the Software/Data Library by searching on the word AUTOCTRL, the Q number of this article, or S14364. AUTOCTRL was archived using the PKware file-compression utility.

MORE INFORMATION

This OLE automation controller sample demonstrates how to create an OLE automation object and how to access its properties and methods.

Creating an Automation Object

[CreateObject() in file AUTOCTRL.CPP.]

The user is prompted for the ProgID of the automation object to be created. CLSIDFromProgID is used to obtain the CLSID of the object; CoCreateInstance is used to create the object.

Accessing Properties and Methods

IDispatch::GetIDsOfNames is used to obtain the ID of the property or method to be accessed; IDispatch::Invoke is used to access the property or method. This controller prompts the user for the locale ID; however, many controllers such as Visual Basic use the default system locale ID.

Setting a Property

[See SetProperty() in AUTOCTRL.CPP.]

The user is prompted for the property name, property value, property type and locale ID. This controller supports three types: VT_BSTR, VT_I2 and VT_R4. Setting the property requires an implicit named parameter, which represents the new value of the property. The DISPID

of this implicit named parameter is DISPID_PROPERTYPUT.

Getting a Property

[See GetProperty() in AUTOCTRL.CPP.]

The user is prompted for the property name, property type, and locale ID.

Invoking a Method

[See InvokeMethod() in AUTOCTRL.CPP.]

The user is prompted for the method name and the locale ID. This controller supports only methods without parameters.

Additional reference words: 2.00 3.50 4.00 softlib AUTOCTRL.EXE
KBCategory: kbole kbfile
KBSubcategory: LeTwoAto

SAMPLE: OLE Automation Inproc Object

PSS ID Number: Q107982
Authored 01-Dec-1993 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.0
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SUMMARY

AUTODLL demonstrates how to create a OLE automation inproc object.

Download AUTODLL.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for AUTODLL.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download AUTODLL.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the SOFTLIB\MSLFILES directory
  Get AUTODLL.EXE

MORE INFORMATION

AUTODLL implements a OLE automation inproc object (DLL server) called Hello that exposes one property, HelloMessage, and one method, SayHello. It uses an OLE provided IDispatch implementation that is created using CreateStdDispatch.

The AUTODLL dynamic-link library (DLL) uses the memory allocator used by the calling task. This is required of all inproc objects. The new and delete operators are redefined to use the calling task's memory allocator and all memory management is done using the redefined new and delete.

The interface that AUTODLL exposes is described using an object description language (HELLO.ODL). The mktyplib tool is used to create a type library (HELLO.TLB) from HELLO.ODL. CreateStdDispatch is then used to implement an IDispatch interface using the interface description in the type library.

AUTODLL exports DllGetClassObject, which is called by OLE to get the class factory. OLE uses this classfactory to create a Hello object. Automation

controllers use the IDispatch interface exposed by the Hello object to access its property and method. AUTODLL also exports DllCanUnloadNow.

The SIMPAUTO sample in the Software/Data library demonstrates the implementation of an automation object with the same methods and properties as AUTODLL, but which is an EXE. The main differences between the two samples are in the MAIN.CPP file.

To Run

The Hello object exposes the following:

ProgID : SimpleAutomationInProc.Hello

Method & Property Names		Notes
HelloMessage	(prop)	Sets and returns string.
SayHello	(method)	Displays HelloMessage in an edit control.

Update the path in HELLO.REG to the current location of the object and the type library.

To Compile

Requires OLE 2.01 or later.

Include device=vmb.386 in the [386Enh] section of SYSTEM.INI.
Note that vmb.386 can be found in \OLE2\BIN.
Run the WXSERVER.EXE from \OLE2\BIN before running the makefile.

Files

MAIN.CPP	Does initialization, redefines new and delete, implements DllGetClassObject, DllCanUnloadNow. and contains code that calls CreateStdDispatch.
HELLO.CPP	Implements the HELLO object.
HELLOCF.CPP	Implements the HELLO class factory.
HELLO.ODL	Object Description Language that describes the property and method that HELLO exposes.
HELLO.H	Defines the HELLO object and the class factory.
TLB.H	Header file created by mktyplib.
MAKEFILE	Makefile for project.
VB.MAK, VB.FRM	Visual Basic project files to control this sample.

Additional reference words: 2.00 3.50 4.00 server softlib AUTODLL.EXE
KBCategory: kbole kbfile
KBSubcategory: LeTwoAto

SAMPLE: Ole2View 1.33 Update Available in Software Library

PSS ID Number: Q122244
Authored 01-Nov-1994 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 3.5
- Microsoft Visual C++ for Windows, version 1.5
- Microsoft Visual C++, 32-bit Edition, version 2.0

SUMMARY

An update to the OLE2View sample application is available in the Microsoft Software Library (MSL) as a self-extracting file named OLE2V.EXE. This file contains both 16- and 32-bit versions of Ole2View version 1.33.

This new release of the sample application fixes several bugs (mostly of a cosmetic nature) found in the versions of Ole2View that shipped with Visual C++ versions 1.5 and 2.0 and with the Win32 Software Development Kit (SDK) for Windows NT version 3.5.

Download OLE2V.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for OLE2V.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download OLE2V.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \softlib\mslfiles directory
  Get OLE2V.EXE

MORE INFORMATION

Files Included in OLE2V.EXE

OLE2VIEW.EXE - Ole2View 1.33 (Win16)
DEFO2V.DLL - IDataObject and ITypeLib interface viewers (Win16)
OLE2VW32.EXE - Ole2View 1.33 (Win32)
DEFO2V32.DLL - IDataObject and ITypeLib interface viewers (Win32)
OLE2VIEW.HLP - Helpfile

Version 1.33 Changes

- The 32-bit version was made 100% compatible with Windows 95 in terms of using the right colors, and so on.

- A Bug that appears in the Japanese version of Windows (display of spurious characters) was fixed.

- DEFO2V32.DLL was changed to include only the ANSI versions of the interface viewers. Now DispplayIDataObjectA, DispplayIDispatchA, DispplayITypeInfoA, and DispplayITypeLibA are gone. DispplayIDataObject, DispplayIDispatch, DispplayITypeInfo, and DispplayITypeLib are now ANSI; they used to be Unicode. This change was made to simplify debugging DEFO2V and because the Unicode versions were never called.

Please report bugs to Microsoft and make suggestions by sending electronic mail to this alias:

ole2view@microsoft.com

Additional reference words: 2.00 1.50 visualc 3.50 softlib
KBCategory: kbother kbprg kbfile
KBSubcategory: letwomisc

SAMPLE: SAFEARAY: Use of Safe Arrays in Automation

PSS ID Number: Q131086
Authored 02-Jun-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.03
- Microsoft OLE libraries included with:
  - Microsoft Windows NT versions 3.5 and 3.51
  - Microsoft Windows 95

SUMMARY

SAFEARAY is an OLE Automation server application that demonstrates the use of safe arrays.

Download SAFEARAY.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services.

- CompuServe
  GO MSL
  Search for SAFEARAY.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download SAFEARAY.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get SAFEARAY.EXE

MORE INFORMATION

SAFEARAY implements the following methods (in addition to the standard application object methods).

SlowSort: Sorts an input safearray of BSTRs and returns the sorted array and the time required for the sort. The array is passed by reference because it needs to be modified by the sort. SlowSort uses SafeArrayGetElement and SafeArrayPutElement to access the array elements.

FastSort: Has the same functionality as SlowSort but uses SafeArrayAccessData to get a pointer to the array elements. This allows array elements to be accessed directly instead of using SafeArrayGetElement and SafeArrayPutElement. This accounts for the speed improvement over SlowSort.

Average: Finds the average of an input safe array of integers. The array is not passed by reference.

GetArray: Creates and returns a safe array of BSTRs.

See the README.TXT included for instructions to compile and run this sample.

Additional reference words: 3.50 4.00
KBCategory: kbprg kbfile
KBSubcategory: LeTwoAt

SAMPLE: Simple OLE 2.0 Container

PSS ID Number: Q99464
Authored 30-May-1993 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.0
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

There is a sample application called SIMPLE in the Software/Data Library that demonstrates how to implement an OLE version 2.0 container application. SIMPLE is intended to be the simplest OLE 2.0 container that is capable of visual editing. SIMPLE demonstrates the use of the OLE 2.0 libraries from a C++ application.

Download SIMPLE.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for SIMPLE.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download SIMPLE.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \softlib\mslfiles directory
  Get SIMPLE.EXE

Additional reference words: 2.00 3.50 4.00
KBCategory: kbole kbfile
KBSubcategory: LeTwoInp

SAMPLE: Simple OLE Automation Object Sample

PSS ID Number: Q107981
Authored 01-Dec-1993 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.0
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SUMMARY

SIMPAUTO demonstrates how to create a simple OLE automation object.

Download SIMPAUTO.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for SIMPAUTO.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download SIMPAUTO.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \softlib\mslfiles directory
  Get SIMPAUTO.EXE

MORE INFORMATION

SIMPAUTO implements a simple OLE automation object called Hello that exposes one property, HelloMessage, and one method, SayHello. It uses an OLE-provided IDispatch implementation that is created using CreateStdDispatch.

The interface that SIMPAUTO exposes is described using an object description language (HELLO.ODL). The mktyplib tool is used to create a type library (HELLO.TLB) from HELLO.ODL. CreateStdDispatch() is then used to implement an IDispatch interface using the interface description in the type library.

SIMPAUTO registers a class factory if it is started with the -Automation command-line switch. OLE uses this classfactory to create a Hello object. Automation controllers uses the IDispatch interface exposed by the Hello object to access its property and method.

The AUTODLL sample in the Software Library demonstrates the

implementation of an inproc automation object with the same methods and properties as SIMPAUTO, but which is a DLL. The main differences between the two samples are in the MAIN.CPP file.

To Run

The Hello object exposes the following:

ProgID : SimpleAutomation.Hello

Method and Property Names		Notes
HelloMessage	(prop)	Sets and returns string.
SayHello	(method)	Displays HelloMessage in an edit control.

Update the HELLO.REG file in two places to include the full path information to the HELLOEXE and the HELLO.TLB files. Change the lines from the following:

```
LocalServer = hello.exe -Automation
win16 = hello.tlb
```

to something similar to the following:

```
LocalServer = c:\source\simpauto\hello.exe -Automation
win16 = c:\source\simpauto\hello.tlb
```

To Compile

Requires OLE 2.01 or later.

Include device=vmb.386 in the [386Enh] section of SYSTEM.INI. Note that vmb.386 can be found in \OLE2\BIN. Run the WXSRVER.EXE from \OLE2\BIN before running the makefile.

Files

MAIN.CPP	Does initialization, creates main window, contains message retriever, and contains code that calls CreateStdDispatch.
HELLO.CPP	Implements the HELLO object.
HELLOCF.CPP	Implements the HELLO class factory.
HELLO.ODL	Object description Language that describes the property and method that HELLO exposes.
HELLO.H	Defines the HELLO object and the class factory.
TLB.H	Header file created by mktyplib.
MAKEFILE	Makefile for project.
VB.MAK, VB.FRM	Visual Basic project files to control this sample.

Additional reference words: 2.00 3.50 4.00 softlib SIMPAUTO.EXE
KBCategory: kbole kbfile

KBSubcategory: LeTwoAto

SAMPLE: TYPEBLD: How to Use ICreateTypeLib & ICreateTypeInfo

PSS ID Number: Q131105
Authored 04-Jun-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.03
- Microsoft OLE libraries included with:
  - Microsoft Windows NT version 3.5
  - Microsoft Windows 95

SUMMARY

The TYPEBLD sample demonstrates how to create an OLE Automation type library using the ICreateTypeLib and ICreateTypeInfo interfaces.

Download TYPEBLD.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for TYPEBLD.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download TYPEBLD.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get TYPEBLD.EXE

See the README.TXT file included in the sample for instructions on to compile and run this sample.

MORE INFORMATION

The type library that is created is called HELLO.TLB and corresponds to one that would have been built by MKTYPLIB.EXE if it had compiled the following .ODL file.

Sample .ODL File

```
[
  uuid(2F6CA420-C641-101A-B826-00DD01103DE1),           // LIBID_Hello
  helpstring("Hello 1.0 Type Library"),
  lcid(0x0409),
  version(1.0)
```

```
]
library Hello
{
#ifdef WIN32
    importlib("stdole32.tlb");
#else
    importlib("stdole.tlb");
#endif

    [
      uuid(2F6CA422-C641-101A-B826-00DD01103DE1),        // IID_IHello
      helpstring("Hello Interface")
    ]
    interface IHello : IUnknown
    {
        [propput] void HelloMessage([in] BSTR Message);
        [propget] BSTR HelloMessage(void);
        void SayHello(void);
    }
    [
      uuid(2F6CA423-C641-101A-B826-00DD01103DE1),        // IID_DHello
      helpstring("Hello Dispinterface")
    ]
    dispinterface DHello
    {
      interface IHello;
    }

    [
       uuid(2F6CA421-C641-101A-B826-00DD01103DE1),       // CLSID_Hello
       helpstring("Hello Class")
    ]
    coclass Hello
    {
        dispinterface DHello;
        interface IHello;
    }
}
```

Additional reference words: 3.50 4.00
KBCategory: kbole kbfile kbcode
KBSubcategory: LeTwoAt

SIMPSVR Implements IDataObject::GetData Incorrectly

PSS ID Number: Q114599
Authored 08-May-1994 Last modified 16-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.01

The simple server sample application (SIMPSVR.EXE) included with the OLE 2.01 SDK incorrectly implements the IDataObject::GetData() method. The code checks the cfFormat member of the FORMATEC structure to see if the format is CF_BITMAP. If so, the code continues, filling out the STGMEDIUM with a METAFILEPICT. The code should be comparing the cfFormat with CF_METAFILEPICT.

The sample code continues to work correctly because the STGMEDIUM structure is filled out appropriately for CF_METAFILEPICT. The default handler puts the information into the cache based on the value that is returned, rather than on the value that is requested.

The sample code included with Visual C++ for Windows version 1.5x has been corrected. It now compares against CF_METAFILEPICT.

Additional reference words: 2.01
KBCategory: kbole kbprg
KBSubcategory: LeTwoSvr

Sizing OLE 2.0 Objects and OLEMISC_RECOMPOSEONRESIZE

PSS ID Number: Q114014
Authored 21-Apr-1994 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, versions 2.0 and 2.01
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SUMMARY

Some server applications want to have their presentation displayed differently based on the size of the object in the container application. For example, if version 6.0 of Microsoft Word For Windows is used as a server application, Word for Windows will rewrap the text of an object based on the object size in the container. This is accomplished in a server application by setting the OLEMISC_RECOMPOSEONRESIZE bit in the MiscStatus bits.

However, for setting the OLEMISC_RECOMPOSEONRESIZE bit to work, the container application must properly implement its sizing code by honoring the OLEMISC_RECOMPOSEONRESIZE bit returned by IOleObject::GetMiscStatus().

MORE INFORMATION

To properly honor the OLEMISC_RECOMPOSEONRESIZE bit, the container needs to check the MiscStatus bits by calling IOleObject::GetMiscStatus(). Once the status bits are retrieved, the container can then check to see if the OLEMISC_RECOMPOSEONRESIZE bit is set. If the bit is set, then the server wishes to be notified every time the object size changes in the container. To inform the server application the object has been resized, the container needs to call IOleObject::SetExtent() to set the new size of the object. However, IOleObject::SetExtent() only works while the object is in the running state. The container application should check if the object is already in a running state by calling the OleIsRunning() function. This is important because the container needs to restore the state of the object once the operation is complete. If the object was not in the running state, OleRun() needs to be called such that the IOleObject::SetExtent() call takes effect. At this point the IOleObject::SetExtent() method call can be made, and should be followed with a call to IOleObject::Update() to update the presentation of the object in the container. Finally, if this section of code ran the server, then it needs to transition the server back to the loaded state by calling IOleObject::Close().

NOTE: Some container applications may want to come up with a more elaborate scheme for keeping servers running. Starting and stopping servers is an expensive process, so the container application might come up with a way to keep the last X (X being application specific) servers held in the running state.

The following code is a simple implementation for the container:

```
// Pass this function a pointer to the object's IOleObject interface,
// the Aspect that has changed size, and the new size of the object.
// returns TRUE if the object had the Recompose on Resize bit set.

BOOL fChkResize(LPOLEOBJECT lpObject, DWASPECT dwAspect, LPSIZEL lpsizel)
{
    DWORD dwStatus = 0; // For the status bits
    BOOL fShutdown = FALSE;  // don't shut the object down
    BOOL retval = FALSE;

    // Get the status bits.
    lpObject->GetMiscStatus(DVASPECT_CONTENT, &dwStatus);

    // is recompose on resize set?
    if ( dwStatus & OLEMISC_RECOMPOSEONRESIZE )
        {
        retval = TRUE;  // the bit was set

        // if the object isn't running, start the object and remember.
        if (!OleIsRunning(lpObject)
            {
            OleRun(lpObject);
            fShutdown = TRUE;
            }

        // set the extent
        lpObject->SetExtent(dwAspect, lpsizel);

        // update the cache
        lpObject->Update();

        // go back to the loaded state only if the object was in the
        // loaded state upon entry to this function.
        if (fShutdown)
            lpObject->Close(OLECLOSE_SAVEIFDIRTY);
        }
    return retval;
}
```

Additional reference words: 2.00 2.01 3.50 4.00
KBCategory: kbole kbprg
KBSubcategory: LeTwoPrs

Summary List of OLE Bugs

PSS ID Number: Q112036
Authored 28-Feb-1994 Last modified 21-Apr-1995

The information in this article applies to:

- Microsoft OLE version 1.0
- Microsoft OLE Libraries for Windows and Win32s, versions 2.0 and 2.01

SUMMARY

The information in this article is a summary of individual articles contained in the Microsoft Knowledge Base (as of 4/21/95). These articles can be found by querying on the article's ID number or on words in the title and text. The Microsoft Knowledge Base can be accessed on CompuServe (go mdkb) and on the Internet (through ftp.microsoft.com or www.microsoft.com).

BUG LISTING

- BUG: OleCreateLinkFromFile Fails on CD-ROM-Based File
- BUG: DIB Can Be Returned Only on TYMED_HGLOBAL
- BUG: SR2TEST GP Faults During Object Shutdown
- BUG: (I)CntrOutl Does Not Set Target Device Information
- BUG: Accelerator Causes Crash in ISvrOutl Embedded in C12Test
- BUG: Insert Object from Zero Length File Causes GP Fault
- BUG: Set Line Height on ISvrOutl Object Causes GPF in Cl2Test
- BUG: Deleting an Open Packager Ojbect Causes GP Fault
- BUG: Cannot Paste Link SR2TEST Object in OLE 1.0 Client
- BUG: SR2TEST Won't Close After Editing Link Object
- BUG: Paste Link Disabled Across the Network
- BUG: Object Packager GPFs w/Paths Greater Than 64 Characters
- BUG: Status Bar Not Redrawn With SR2TEST When In-Place
- BUG: Cursor Does Not Update with Sr2Test and ICntOutl
- BUG: OleCreate and IOleCache::Cache Fail with Multiple TYMEDs
- BUG: IROT::IsRunning() Returns S_FALSE for OLE 1.0 Servers
- BUG: Retaining Clipboard IDataObject Causes Unexpected Result
- BUG: SR2TEST Menu Items Enabled Incorrectly
- BUG: OLE 2.0 Containers & 1.0 Objects that Close w/out Saving
- BUG: CL2TEST Fails to Parse Filenames with Extended Characters
- BUG: Insertion of Large .WAV Object Fails
- BUG: OLE 2.01 Does Not Support CF_OWNERDISPLAY
- BUG: IOleCache::Cache Returns Incorrect Error Value
- BUG: OLERENDER_ASIS Results in Blank Embedded Object
- BUG: Invisible MSDRAW Object Retains Keyboard Focus
- BUG: Iconic OLE Object Prints as Black Rectangle on PostScript
- BUG: First Entry in Paste Special Dialog Is Blank
- BUG: IEnumUnknown Is Not Remoted
- BUG: SVROUTL Link Not Displayed Correctly in CNTROUTL
- PRB: Embedded Object's Size Changes When it Is Run
- BUG: Embedded Object's Size Changes When it is Run

- BUG: SVROUTL Link Not Displayed Correctly in CNTROUTL
- BUG: CL2TEST Does Not Display Prompt String from GetCurFile()
- BUG: CL2TEST Handles Icon Aspect Incorrectly
- BUG: CL2TEST Not Properly Activating Links to Embedded Objects
- BUG: IOleObject::Close(OLECLOSE_NOSAVE) and DoVerb()
- BUG: OLE 1.0 Server Launched for Paste Link
- BUG: OLE 2.0 Compatibility Layer Uses Document IDataObject
- BUG: CreateFromTemplate of OLE 2 Object into OLE 1 Container
- BUG: IROT::Register() and IOL::SetDisplayName() Inconsistency
- BUG: OLE Type Emulation for Previously Loaded Objects
- BUG: Relative Monikers and OLE 1.0 Link Objects
- BUG: OleConvertStorageToOLESTREAM() Fails When CLSID Is NULL
- BUG: Paste-Linking a 256-Color Paintbrush Object
- PRB: IOleObject::IsUpToDate() and OLE 1.0 Link Objects
- BUG: IOleCache::Cache(), ADVF_DATAONSTOP, and OLE 1.0 Objects
- BUG: IOleObject::IsUpToDate Returns Wrong Value for Manual Link
- BUG: Borland WINSIGHT Causes GP Faults w/Some OLE Sample Apps
- BUG: Windows OLE DLLs Do Not Convert Mac OLESTREAM
- BUG: Printing Does Not Work from CL2TEST.EXE

BUG: OleCreateLinkFromFile Fails on CD-ROM-Based File

Article ID: Q86408

In an OLE client application, the
OleCreateLinkFromFile function returns the value OLE_ERROR_DRIVE.

BUG: DIB Can Be Returned Only on TYMED_HGLOBAL

Article ID: Q98679

IDataObject::GetData returns DV_E_FORMATETC when requesting a
device-independent bitmap (DIB) from an OLE object.

BUG: SR2TEST GP Faults During Object Shutdown

Article ID: Q104141

A general protection (GP) fault occurs when calling IOleObject::Close()
on an SR2TEST object.

BUG: (I)CntrOutl Does Not Set Target Device Information

Article ID: Q108310

During the print process from (I)CntrOutl, a server application is asked to
render its display with the TARGETDEVICE structure set to NULL instead of

containing a valid structure. This implies that (I)CntrOutl wants a screen representation rather than a printed representation of the object.

BUG: Accelerator Causes Crash in ISvrOutl Embedded in C12Test

---

Article ID: Q108311

After embedding an ISvrOutl object in C12Test, pressing the ALT+BACKSPACE accelerator (Undo) results in a general protection (GP) fault within ISvrOutl.

BUG: Insert Object from Zero Length File Causes GP Fault

Article ID: Q108371

Creating an object from a zero length file that does not have a valid file class extension results in a general protection (GP) fault after displaying the message, "Not enough memory to perform this operation. Close one or more applications and try again."

BUG: Set Line Height on ISvrOutl Object Causes GPF in C12Test

---

Article ID: Q108930

In certain circumstances, editing the Line Height of an ISvrOutl object embedded within C12Test results in a general protection (GP) fault after displaying the following message:

> GDI - An error has occurred in your application. If you choose Ignore, you should save your work in a new file. If you choose Close, your application will terminate.

BUG: Deleting an Open Packager Ojbect Causes GP Fault

Article ID: Q108931

Deleting an object from within the Object Packager results in a general protection (GP) fault.

BUG: Cannot Paste Link SR2TEST Object in OLE 1.0 Client

Article ID: Q108932

The Paste Link option is not available in an OLE 1.0 client application after copying an unsaved SR2TEST object to the clipboard.

BUG: SR2TEST Won't Close After Editing Link Object

Article ID: Q108935

After editing a linked SR2TEST object from CL2TEST, SR2TEST fails to shut down after first displaying the following message box:

   Error Document not unlocked. Attempt Self-unlocking?

BUG: Paste Link Disabled Across the Network

Article ID: Q108939

The OleQueryLinkFromData application programming interface (API) function returns failure when clipbook data from another machine is copied to the local clipboard, even if the clipbook contains all of the data needed to create a link.

BUG: Object Packager GPFs w/Paths Greater Than 64 Characters

Article ID: Q109116

Packaging an object that has a fully qualified path of more than 64 letters results in a general protection (GP) fault in the Object Packager.

BUG: Status Bar Not Redrawn With SR2TEST When In-Place

Article ID: Q109541

After inserting an SR2TEST object into the in-place version of the OUTLINE sample application, the status bar will not be redrawn until the container application's window is resized.

BUG: Cursor Does Not Update with Sr2Test and ICntOutl

Article ID: Q109542

After inserting an Sr2Test object within the in-place Outline sample application, cursor display is not handled correctly when the cursor is positioned over the in-place object.

BUG: OleCreate and IOleCache::Cache Fail with Multiple TYMEDs
---

Article ID: Q109543

The OLE 2.01 creation functions [OleCreate(), OleCreateFromData(), and so forth] and IOleCache::Cache() fail if multiple values are specified for the TYMED field of the FORMATETC parameter. The functions fail even if the object server supports at least one of the TYMED values specified.

BUG: IROT::IsRunning() Returns S_FALSE for OLE 1.0 Servers

Article ID: Q109544

When an OLE 2.01 container application calls IRunningObjectTable::IsRunning() on the moniker for a linked OLE 1.0 object, IsRunning() returns S_FALSE even if that OLE 1.0 object is already running.

BUG: Retaining Clipboard IDataObject Causes Unexpected Result
---

Article ID: Q109545

An application holding on to a clipboard IDataObject pointer, and making repeated calls to IDataObject::EnumFormatEtc() through that pointer, may find that the set of formats returned by EnumFormatEtc() changes between calls.

BUG: SR2TEST Menu Items Enabled Incorrectly

Article ID: Q109546

During an in-place activation session, SR2TEST sometimes incorrectly enables menu items on its Edit menu.

BUG: OLE 2.0 Containers & 1.0 Objects that Close w/out Saving
---

Article ID: Q109547

When an OLE 1.0 object is inserted into an OLE 2.0 container document and then closed without an update being invoked, the correct streams for that object are not written to storage. Any subsequent attempt by the container to load the object will fail.

BUG: CL2TEST Fails to Parse Filenames with Extended Characters
---

Article ID: Q109548

When attempting to insert a new embedded object from a file, CL2TEST fails with the error STG_E_FILENOTFOUND if the filename or directory name provided contains an extended character.

BUG: Insertion of Large .WAV Object Fails

Article ID: Q109549

In some cases, inserting a large .WAV file object into either an OLE 1.0 or OLE 2.0 container will fail.

BUG: OLE 2.01 Does Not Support CF_OWNERDISPLAY

Article ID: Q109552

If an application places a data transfer object onto the clipboard and that data object enumerates CF_OWNERDISPLAY as one of the clipboard formats, the application will not receive the WM_PAINTCLIPBOARD message.

BUG: IOleCache::Cache Returns Incorrect Error Value

Article ID: Q110488

An OLE 2.0 container application calls IOleCache::Cache for a particular FORMATETC and is returned the value OLE_S_FORMATETC_NOTSUPPORTED. A subsequent call to IOleCache::Cache with the same FORMATETC returns CACHE_S_SAMECACHE.

BUG: OLERENDER_ASIS Results in Blank Embedded Object

Article ID: Q110714

If an OLE 1.0 client application contains an embedded object and copies the object to the clipboard and an OLE 2.0 container application then performs a paste operation by creating a new embedded object based on the clipboard data by calling OleGetClipboard() and then calling OleCreateFromData(), the resulting embedded object appears blank in the container's document if the OLE 2.0 container specifies OLERENDER_ASIS as the renderOpt parameter to OleCreateFromData().

BUG: Invisible MSDRAW Object Retains Keyboard Focus

Article ID: Q110715

When an invisible MSDRAW object is inserted as an OLE embedded object, the object retains the keyboard focus. This behaviour is incorrect. The MSDRAW object should not take the focus until it has been made visible.

This problem occurs whether the MSDRAW object is inserted into an OLE 1.0 client application or into an OLE 2.0 container application.

BUG: Iconic OLE Object Prints as Black Rectangle on PostScript

---

Article ID: Q110796

Calling OleDraw() or IViewObject::Draw() to print an iconic OLE object to a PostScript printer results in a black rectangle being drawn.

BUG: First Entry in Paste Special Dialog Is Blank

Article ID: Q110798

The first format entry in the Paste Special dialog box is blank after copying an OLE2Link object from an OLE 1.0 client to the clipboard.

NOTE: An OLE2Link object is created by the OLE 1.0 compatibility layer when an OLE 2.0 link is copied from an OLE 2.0 application to the clipboard, and then pasted into an OLE 1.0 client application.

BUG: IEnumUnknown Is Not Remoted

Article ID: Q110799

All calls to IOleContainer::EnumObjects() fail.

BUG: SVROUTL Link Not Displayed Correctly in CNTROUTL

Article ID: Q110871

Selecting the Paste Link menu item in the CNTROUTL sample application does not correctly display the contents of the link if the source application is the SVROUTL sample application.

This problem occurs only if the information created in SVROUTL is edited further after the Edit Copy operation in SVROUTL, but before the Paste Link operation in CNTROUTL. In this situation, the resulting link object is displayed as originally copied to the clipboard, and does not reflect the changes made after the copy operation.

PRB: Embedded Object's Size Changes When it Is Run

Article ID: Q110872

A non-running embedded object is resized in an OLE 2.01 container application. The next time the object is run, it snaps back to its previous size.

BUG: Embedded Object's Size Changes When it is Run

Article ID: Q111004

A non-running embedded object is resized in an OLE 2.01 container application. The next time the object is run, it snaps back to its previous size.

BUG: SVROUTL Link Not Displayed Correctly in CNTROUTL

Article ID: Q111006

Selecting the Paste Link menu choice in the CNTROUTL sample application does not correctly display the contents of the link if the source application is the SVROUTL sample application.

This problem occurs only if the information created in SVROUTL is edited further after the Edit Copy operation in SVROUTL, but before the Paste Link operation in CNTROUTL. In this situation, the resulting link object is displayed as originally copied to the clipboard, and does not reflect the changes made after the copy operation.

BUG: CL2TEST Does Not Display Prompt String from GetCurFile()

Article ID: Q111014

IPersistFile::GetCurFile() returns S_FALSE to indicate that the document has no currently associated file. In this case, GetCurFile() returns the default prompt string for the filename (as would be displayed in the Save As dialog box under the File menu) in the lplpszFileName parameter.

When CL2TEST calls IPersistFile::GetCurFile(), and GetCurFile() returns S_FALSE, CL2TEST fails to display the returned prompt string.

BUG: CL2TEST Handles Icon Aspect Incorrectly

Article ID: Q111339

Selecting the iconic aspect when inserting an object in CL2TEST results in

the object being created and displayed with the content aspect.

BUG: CL2TEST Not Properly Activating Links to Embedded Objects

---

Article ID: Q111340

Double-clicking a link to an embedded object in CL2TEST results in the activation remaining on CL2TEST. The correct action would be for the container of the embedded object to obtain the activation and edit the object visually (if possible).

BUG: IOleObject::Close(OLECLOSE_NOSAVE) and DoVerb()

Article ID: Q111577

An OLE 2.0 container application inserts a new embedded object into a document. The container calls IOleObject::Update() on the object to update the cache, then calls IOleObject::Close(OLECLOSE_NOSAVE) to transition the object back to the running state. Finally, the container tries to rerun the object by calling IOleObject::DoVerb().

In this particular scenario, DoVerb() will return STG_E_FILENOTFOUND, and the object will not be rerun.

BUG: OLE 1.0 Server Launched for Paste Link

Article ID: Q111578

An OLE 1.0 server application copies an object to the clipboard, then enters the blocked state. An OLE 2.0 container application then attempts to paste a link to the object from the clipboard. The OLE 2.0 application is frozen.

Next, the OLE 1.0 application is shut down. This unfreezes the OLE 2.0 container application, which proceeds to paste the link to the OLE 1.0 object. However, an instance of the OLE 1.0 server is launched. Normally, the server is not run in a paste-link scenario.

BUG: OLE 2.0 Compatibility Layer Uses Document IDataObject

Article ID: Q111585

Whenever the OLE 1.0 client compatibility layer uses IDataObject, it uses the OLE 2.0 server's document-level data object rather than the server's pseudo-object data object.

BUG: CreateFromTemplate of OLE 2 Object into OLE 1 Container

Article ID: Q111595

An OLE 2.01 object is inserted into an OLE 1.0 container using OleCreateFromTemplate(). The OLE libraries start the server, but then the object creation fails, or a blank object is created.

BUG: IROT::Register() and IOL::SetDisplayName() Inconsistency

Article ID: Q111607

IRunningObjectTable::Register() allows monikers that are not valid filenames to be registered as file monikers. However, if the display name string passed to IOleLink::SetSourceDisplayName() is not a valid filename, SetSourceDisplayName() returns STG_E_FILENOTFOUND.

BUG: OLE Type Emulation for Previously Loaded Objects

Article ID: Q111608

OLE type emulation is the process that allows the application user to specify that all objects of some particular type are henceforth to be activated as objects of some alternate, emulating type. When objects of the original type are subsequently run, the server for the emulating type is launched to serve them.

However, this emulation does not occur for objects of the original type that were already in the loaded state when the type emulation occurred. When such objects are subsequently run, they are run as the original type, not the emulating type. The original server is launched, not the server for the emulating type.

BUG: Relative Monikers and OLE 1.0 Link Objects

Article ID: Q111609

When a link object from an OLE 2.0 application is copied to the clipboard and then pasted into an OLE 1.0 application, the OLE 2.0 emulation layer preserves the object's relative moniker. If the OLE 1.0 application document is closed and then reopened by an OLE 2.0 version of the application, OLE 2.0 will create a full moniker for the linked object by appending the preserved relative moniker to the file moniker for the document. This full moniker will be incorrect, because it will point to the new document rather than to the first OLE 2.0 application's document that contains the original linked object.

However, OLE 2.0 will still be able to bind to the linked object by

following the absolute moniker that is also stored with the object (providing that the absolute location of the original linked object has not changed). At the time of this binding, OLE will then correct the full moniker for the object.

BUG: OleConvertStorageToOLESTREAM() Fails When CLSID Is NULL

Article ID: Q111611

The OleConvertStorageToOLESTREAM() API (application programming interface) call is used to convert the storage of an embedded object from the OLE 2.0 storage model to the OLE 1.0 storage model. However, if the OLE 2.0 object has a CLSID of NULL, then OleConvertStorageToOLESTREAM() fails with a return code of OLE_E_CLASS.

BUG: Paste-Linking a 256-Color Paintbrush Object

Article ID: Q111612

A 256-color bitmap is loaded into Microsoft Paintbrush version 3.11. The image is selected and then copied to the clipboard. An OLE 2.0 application then pastes a link to the object from the clipboard. To perform the paste-link operation, the container calls OleCreateLinkFromData(), specifying OLERENDER_FORMAT as the rendering option and CF_DIB as the desired format.

The resulting link object in the OLE 2.0 application has an incorrect appearance, such as being colored solid black.

PRB: IOleObject::IsUpToDate() and OLE 1.0 Link Objects

Article ID: Q111613

An OLE 2.0 container application calls IOleObject::IsUpToDate to determine whether or not an OLE 1.0 link object is up to date. IsUpToDate() returns S_FALSE, even though the object is actually up to date.

BUG: IOleCache::Cache(), ADVF_DATAONSTOP, and OLE 1.0 Objects

---

Article ID: Q111614

An OLE 2.0 container application document contains an embedded OLE 1.0 object. The container calls IOleCache::Cache() to control the cached presentation data for the object, and specifies ADVF_DATAONSTOP as the advise flag to Cache(). The user makes some changes to the object in the OLE 1.0 server, then attempts to update the object's presentation in the server by choosing the Update command from the File menu. Finally, the user closes the object server.

In this specific scenario, the object's presentation in the server is not updated. The object's native data, however, is correct.

BUG:IOleObject::IsUpToDate Returns Wrong Value for Manual Link

---

Article ID: Q111655

To determine whether an OLE link needs to be updated, an OLE 2.0 container application needs to call IOleObject::IsUpToDate. If this method returns S_FALSE, the link needs to be updated; otherwise, the link can be considered up to date. However, calling IOleObject::IsUpToDate on a manual link while the link server is not running results in an a return value of S_FALSE, even if the link is current. The result of this is that OLE 2.0 container applications may update links unnecessarily.

BUG: Borland WINSIGHT Causes GP Faults w/Some OLE Sample Apps

---

Article ID: Q112410

If Borland's WINSIGHT tool is running when you start some of the sample applications that have been included in the OLE 2.01 SDK, a general protection (GP) Fault will occur before the main application window appears.

The following OLE 2.01 SDK sample applications are affected:

- cl2test.exe
- sr2test.exe
- outline.exe
- cntroutl.exe
- svroutl.exe
- icntrotl.exe
- isvrotl.exe

BUG: Windows OLE DLLs Do Not Convert Mac OLESTREAM

Article ID: Q112412

The OLE 2.01 libraries will not convert a Mac OLESTREAM to a Windows OLESTREAM. Similarly, the Windows OLE dynamic-link libraries (DLLs) will not convert a Mac IStorage to a Windows OLESTREAM.

BUG: Printing Does Not Work from CL2TEST.EXE

Article ID: Q112413

Printing does not work correctly from CL2TEST.EXE. Choosing Print from the File menu results in a blank printout.

Additional reference words: 1.00 2.00
KBCategory: kbole kbbuglist
KBSubCategory: LeTwoOth

The Component Object Model

PSS ID Number: Q104140
Authored 08-Sep-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft OLE Libraries for Windows and Win32s, version 2.0
- Microsoft OLE Libraries, included with:
  - Microsoft Windows NT, versions 3.5 and 3.51
  - Microsoft Windows 95

SUMMARY

The Component Object Model is a specification that describes the process of communicating through interfaces, acquiring access to interfaces through the QueryInterface method, determining pointer lifetime through reference counting, and re-using objects through aggregation.

MORE INFORMATION

Object

An object is an item in the system that exposes interfaces to manipulate the data or properties of the object. An object is created by directly or indirectly calling the CoCreateInstance() application programming interface (API), which in turn creates a new instance of the object and returns a pointer to a requested interface. For more details, see pages 93 and 94 of the "OLE 2.0 Design Specification."

Interfaces

An interface is a group of related functions. Communication between two objects in a system occurs by calling the functions in an interface through a pointer to that interface. An interface pointer is originally obtained at the time the object is created.

A good example of an interface is a window that supports drag and drop. The window exposes an interface with methods that could be used during drag and drop. The object being dragged could communicate with the window through this interface. Such an interface might resemble the following:

```
interface IDropTarget : IUnknown {
    virtual HRESULT DragEnter() = 0; // Mouse entered the window.
    virtual HRESULT DragOver()  = 0; // Called each mouse move.
    virtual HRESULT DragLeave() = 0; // Mouse left the window.
    virtual HRESULT Drop()      = 0; // Item dropped on the window.
    };
```

For more information on interfaces, refer to pages 57-60 of the "OLE 2.0 Design Specification."

IUnknown

All interfaces used in the component object model are derived from a base interface called IUnknown. The methods contained within IUnknown are related because they deal with object maintenance. The IUnknown interface is defined as:

```
interface IUnknown {
    virtual HRESULT QueryInterface( REFIID, VOID FAR *) = 0;
    virtual ULONG   AddRef() = 0;
    virtual ULONG   Release() = 0;
};
```

IUnknown::QueryInterface is used for interface negotiation. The other methods are used for reference counting to control the life of the object.

More information on the IUnknown interface can be found on pages 81-83 of the "OLE 2.0 Design Specification."

Interface Negotiation

Given a pointer to a particular interface, an object can be queried for another interface. This is done by calling the QueryInterface() method in an interface. The following code demonstrates querying for the IOleObject interface:

```
// Assume that a pointer to an arbitrary interface, pint,
// exists.

LPOLEOBJECT pOleObject;
HRESULT hErr;

// Query the interface.
hErr = pint->QueryInterface(IID_IOleObject, (LPVOID FAR *)
                                            &pOleObject);

if (hErr == NOERROR)
    // Object supports this IOleObject. The IOleObject
    // methods can now be called through pOleObject.
else
    // Object does not support IOleObject.
```

Reference Counting

Interface lifetime is controlled through reference counting. To increment the reference count on an interface, call the AddRef() method. To decrement the reference count on an interface, call the

Release() method. Once an interface's reference count goes to zero, the pointer to that interface is no longer valid. If the reference count on all of an object's interfaces is zero, then the object can be freed because there are no longer any pointers to the object.

More information on reference counting can be found on pages 83 and 84 of the "OLE 2.0 Design Specification."

Aggregation

Aggregation is the ability of an object to be re-used or extended dynamically, without having to recompile the original object code. For more information on the process of aggregation, please refer to the "OLE 2.0 Design Specification," pages 61-63.

Additional reference words: 2.00 3.50 4.00
KBCategory: kbole kbprg
KBSubcategory: LeTwoCom

Using MKTYPLIB /h Option to Output C or C++ Style Header file

PSS ID Number: Q124597
Authored 05-Jan-1995 Last modified 21-Apr-1995

The information in this article applies to:

- The Type Library Generator (MKTYPLIB.EXE), included with:
  - Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5
  - Microsoft Visual C++, 32-bit Edition, version 2.0 on the following platform: x86

SUMMARY

In addition to generating type libraries, MKTYPLIB can output a C or C++ style header file if you use the optional /h option. When you use the /h option, MKTYPLIB creates a header file with the name specified following the /h option. Any existing file of the same name will be overwritten without warning.

MKTYPLIB always uses the "libraryname" from the LIBRARY keyword in the ODL file to create the #ifndef wrappers in the generated header file, regardless of the name specified for the file on the command line.

MORE INFORMATION

Consider this ODL file (MYPROJ.ODL):

```
// example ODL file
library MYPROJ
{
   ...
}
```

If you run MKTYPLIB on this ODL file by using this command line:

```
C:\> MKTYPLIB /h odlfile.h myproj.odl
```

the ODLFILE.H generated by MKTYPLIB looks like this:

```
#ifndef _MYPROJ_H_
#define _MYPROJ_H_
...
#endif
```

This can cause problems when compiling source files if another header file also uses _MYPROJ_H_ in its #ifndef wrappers. Only the first file to use _MYPROJ_H_ will actually be included in the source file.

If you are using the _<filename>_H_ convention for #ifndef wrappers, make

sure none of your header files have the same name as your type library. Alternatively, use a different naming convention for the #ifndef wrappers in your header files.

Additional reference words: 2.00 3.50
KBCategory: kbole kbtool
KBSubCategory: LeTwoTls

16-Bit App WNetGetCaps Call Return Value on Win32

PSS ID Number: Q120359
Authored 08-Sep-1994 Last modified 10-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.5, 3.51, and 4.0

16-bit Windows-based applications often call WNetGetCaps() to determine the capabilities of the installed network. When a Windows-based application running on Windows NT calls WNetGetCaps(), the return value is 0x8004, which corresponds to WNNC_NET_Multinet | WNNC_SUBNET_WinWorkgroups.

However, the Windows NT Windows on Windows (WOW) layer and Windows 95 do not support the Windows for Workgroups Multinet (MNet) APIs, so a call to one of these APIs returns a failed Dynalink error.

The return value of WNetGetCaps() may not seem technically correct for Windows for Workgroups. It was designed to be compatible with all existing 16-bit Windows-based applications.

If you need to determine whether a 16-bit Windows-based application is running on Windows NT or MS-DOS/Windows version 3.1, use GetWinFlags(). GetWinFlags() returns a WF_WINNT flag if the application is running under WOW on Windows NT.

GetWinFlags() is an existing function that was modified in WOW to add the following flag:

```
   #define WF_WINNT         0x4000
```

Additional reference words: 3.50 4.00
KBCategory: kbprg
KBSubcategory: SubSys

32-Bit Scroll Ranges

PSS ID Number: Q104311
Authored 13-Sep-1993 Last modified 16-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1 and 3.5

You can use 32-bit scroll ranges by calling GetScrollPos(); however, you cannot get 32-bit positions for notifications sent while thumb tracking, that is, via the SB_THUMBPOSITION message. This is because thumb position information is not queryable via an application programming interface (API). You only can obtain the 32-bit scroll information only before or after the scroll has taken place.

The scroll bar APIs allow setting a scroll range up to 0x7FFFFFFF via SetScrollRange(), and setting a scroll position within that range using SetScrollPos(). If the WM_HSCROLL or WM_VSCROLL message is processed, the information returned for scroll bar position, nPos, is only a 16-bit value. To obtain the 32-bit information, the GetScrollPos() API must be used.

Additional reference words: 3.10 3.50 scrollbar
KBCategory: kbprg
KBSubcategory: UsrCtl

Accessing Parent Window's Menu from Child Window w/ focus

PSS ID Number: Q92527
Authored 09-Nov-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

In an MDI-like application, the user must be allowed to pull down menus in the parent window by using menu mnemonics even though the child window or one of its children may have the focus. This can be done by creating child windows without a system menu or by processing the WM_MENUCHAR and WM_SYSCOMMAND/SC_KEYMENU messages to programatically pull down the parent's menu.

MORE INFORMATION

If a child window with a system menu has the focus and the user attempts to access the parent's menu with the keyboard using the menu mnemonic (ALT+mnemonic character), Windows will beep and the parent's menu will not be pulled down. This problem occurs because the parent window does not have the focus and because the window with the focus does not have a menu corresponding to the mnemonic. (Child windows cannot have menus other than the system menu.)

If the child window with the focus does not have a system menu, Windows assumes that the menu mnemonic is for the nearest ancestor with a system menu and passes the message to that parent. Consequently, it is possible to use menu mnemonics to pull down a parent's menu if the descendant windows do not have system menus.

If the child window with the focus has a system menu, Windows will beep if a menu mnemonic corresponding to a parent menu is typed. This can be prevented and the parent menu can be dropped down using the following code in the window procedure of the child window:

```
 case WM_MENUCHAR:
    PostMessage(hwndWindowWithMenu, WM_SYSCOMMAND, SC_KEYMENU, wParam);
    return(MAKELRESULT(0, 1));
```

WM_MENUCHAR is sent to the child window when the user presses a key sequence that does not match any of the predefined mnemonics in the current menu. wParam contains the mnemonic character. The child window posts a WM_SYSCOMMAND/SC_KEYMENU message to the parent whose menu is to be dropped down, with lParam set to the character that corresponds to the menu mnemonic.

The above code can also be used if the child window with the focus does not

have a system menu but an intermediate child window with a system menu exists between the child with the focus and the ancestor whose menu is to be dropped. In this case, the code would be placed in the intermediate window's window procedure.

Additional reference words: 3.10 3.00 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMen

Accessing the Application Desktop from a Service

PSS ID Number: Q115825
Authored 05-Jun-1994 Last modified 13-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.5 and 3.51

SUMMARY

Under Windows NT, version 3.1, if you want a service to have access to the application desktop, you must run the service in the LocalSystem account. A service process running in the LocalSystem account (or a process started from such a service) can display message boxes, windows, and dialog boxes. Processes that are running in the LocalSystem account are not terminated by the system during logoff. A number of changes were made to Windows NT, version 3.5, that affect the way Windows NT interacts with these services. In addition, Windows NT 3.51 has a richer set of desktop APIs.

NOTE: Running interactive services under the system account is a VERY dangerous practice. This is especially true of the command processor and batch files. A user who wants to control the system can just hit CTRL+C to get an interactive system command prompt.

MORE INFORMATION

The following are new features of Windows NT, version 3.5, that affect services:

- The account of the logged in user is the only account granted access to the application desktop. The LocalSystem no longer has access. Therefore, it is possible to get access to the desktop by impersonating the user before making any USER or GDI calls.

- Console and GUI applications started from a service process during a particular logon session are run on an invisible window station and desktop that are unique to that session. The window station and desktop are created automatically when the first application in the session starts; they are destroyed when the last application exits. There is no way to make these invisible desktops visible.

- If you want a service to interact with the logged-on user, specify the SERVICE_INTERACTIVE_PROCESS flag in the call to CreateService(). For example:

```
schService = CreateService(
   schSCManager,
   serviceName,
   serviceName,
   SERVICE_ALL_ACCESS,
   SERVICE_INTERACTIVE_PROCESS | SERVICE_WIN32_OWN_PROCESS,
```

```
        SERVICE_DEMAND_START,
        SERVICE_ERROR_NORMAL,
        lpszBinaryPathName,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL );
```

- If you use CreateProcess() to launch your process and you want your service to log onto the users desktop, assign the lpdesktop parameter of the STARTUPINFO struct with "WinSta0\\Default".

- Services that simply need a visible user notification can do this by calling MessageBox() with the MB_SERVICE_NOTIFICATION flag. Using the MB_DEFAULT_DESKTOP_ONLY flag works as well, but only if the user's desktop is active. If the workstation is locked or a screen saver is running, the call will fail.

  NOTE: If you are writing code for an application that can be run as either a service or an executable, you can't use MB_SERVICE_NOTIFICATION as well as a non-NULL hwndOwner.

- Any output done to a window is not displayed or made available to the application in any way. Attempts to read bits from the display results in a failure.

- GUI services do not receive WM_QUERYENDSESSION/WM_ENDSESSION messages at logoff and shutdown; instead, they receive CTRL_LOGOFF_EVENT and CTRL_SHUTDOWN_EVENT events. These services are not terminated by the system at logoff.

Additional reference words: 3.50
KBCategory: kbprg
KBSubcategory: BseService

Accessing the Event Logs

PSS ID Number: Q108230
Authored 07-Dec-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

Event logs are used to store significant events, such as warnings, errors, or information. There are five operations that can be performed on event logs through the event logging application programming interface (API): backup, clear, query, read, and write.

The default event logs are the Application event log, the Security event log, and the System event log. Access to these event logs is determined by which account the application is running under.

MORE INFORMATION

The following table shows which accounts are granted access to which logs and what type of access is granted under Windows NT 3.1:

Log	Account	Access Granted
Application	LocalSys	read write clear
	Admins	read write clear
	ServerOp	read write clear
	World	read write
Security	LocalSys	read write clear
	Admins	read clear
System	LocalSys	read write clear
	Admins	read clear
	ServerOp	read clear
	World	read

Table 1 - access granted in Windows NT 3.1

The Local System account (LocalSys) is a special account that may be used by Windows NT services. The Administrator account (Admins) consists of the administrators for the system. The Server Operator account (ServerOp) consists of the administrators of the domain server. The World account includes all users on all systems.

Changes made were for Windows NT 3.5:

Log	Account	Access Granted

Application	LocalSys	read write	clear
	Admins	read write	clear
	ServerOp	read write	clear
	World	read write	
Security	LocalSys	read write	clear
	Admins	read	clear
	World	read	clear *
System	LocalSys	read write	clear
	Admins	read write	clear **
	ServerOp	read	clear
	World	read	

Table 2 - access granted under Windows NT 3.5

* Users that have been granted manage auditing and security log rights can read and clear the Security log.

** Admins can write to the System log.

The following table shows which types of access are required for the corresponding event logging API:

Event Logging API	Access Required
OpenEventLog()	read
OpenBackupEventLog()	read
RegisterEventSource()	write
ClearEventLog()	clear

Table 3 - access required for event logging APIs

As an example, OpenEventLog() requires read access (see Table 2). A member of the ServerOp account can call OpenEventLog() for the Application event log and the System event log, because ServerOp has read access for both of these logs (see Table 1). However, a member of the ServerOp account cannot call OpenEventLog() for the Security log, because it does not have read access for this log (see Table 1).

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseMisc

Accessing the Macintosh Resource Fork

PSS ID Number: Q106663
Authored 11-Nov-1993 Last modified 02-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

Resource forks are implemented as NTFS streams named AFP_Resource. There is no structure to the fork; it is exactly whatever the Macintosh writes to it. The resource fork can be written to using standard Win32 application programming interfaces (APIs). Refer to the forks as <FileName>:AFP_Resource.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseMisc

Accurately Showing on the Screen What Will Print

PSS ID Number: Q75469
Authored 21-Aug-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

Many applications have an option where the screen display is set to closely correspond to the printed output. This article discusses some of the issues involved in implementing this feature.

MORE INFORMATION

If a screen font is available that exactly matches (or at least very closely corresponds to) the chosen printer font, then the process is very straightforward and consists of seven steps:

1. Retrieve a device context (DC) or an information context (IC) for the printer.

2. Call EnumFontFamilies() to obtain a LOGFONT structure for the chosen printer font. The nFontType parameter to the EnumFontFamilies() callback function specifies if a given font is a device font.

3. Get a DC for the screen.

4. Convert the lfHeight and lfWidth members of the LOGFONT structure from printer resolution units to screen resolution units. If a mapping mode other than MM_TEXT is used, round-off error may occur.

5. Call CreateFontIndirect() with the LOGFONT structure.

6. Call SelectObject(). GDI will select the appropriate screen font to match the printer font.

7. Release the printer device context or information context and the screen device context.

If a screen font that corresponds to the selected printer font is not available, the process is more difficult. It is possible to modify the character placement on the screen to match the printer font to show justification, line breaks, and page layout. However, visual similarity between the printer fonts and screen fonts depends on a number of factors, including the number and variety of screen fonts available, the selected printer font, and how the printer driver describes the font. For example, if the printer has a serifed Roman-

style font, one of the GDI serifed Roman-style fonts will appear to be very similar to the printer font. However, if the printer has a decorative Old English-style font, no corresponding screen font will typically be available. The closest available match would not be very similar.

To have a screen font that matches the character placement of a printer font, do the following:

1. Perform the seven steps above to retrieve an appropriate screen font.

2. Get the character width from the TEXTMETRIC structure returned by the EnumFonts function in step 2 above. Use this information to calculate the page position of each character to be printed in the printer font.

3. Allocate a block of memory and specify the spacing between characters. Make sure that this information is in screen resolution units.

4. Specify the address of the memory block as the lpDx parameter to ExtTextOut(). GDI will space the characters as listed in the array.

Additional reference words: 3.00 3.10 3.50 4.00 95 WYSIWYG
KBCategory: kbprg
KBSubcategory: GdiPrn

Action of Static Text Controls with Mnemonics

PSS ID Number: Q65883
Authored 26-Sep-1990 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

The text of a static control may contain a mnemonic, which is a character with which the user can access the control. A mnemonic is indicated to the user by underlining the character in the text of the control, and is created by preceding the desired character with an ampersand (&).

Mnemonic characters are used in conjunction with the ALT key to allow quick access to a control with the keyboard. When the user enters the key combination of the ALT key and the mnemonic character, Windows sets the input focus to the corresponding control and performs the same action as when the mouse is clicked on that control. Push buttons, option buttons, and check boxes all behave in this manner.

Because static text controls do not accept the focus, the behavior of a mnemonic in a static text control is different. When the user enters the mnemonic of a static text control, the focus is set to the next enabled nonstatic control. A static text control with a mnemonic is primarily used to label an edit control or list box. When the user enters the mnemonic, the corresponding control gains the focus.

In this context, the order in which windows are created is important. In a dialog box template, the control defined on the line following the static text control is considered to be "next."

When the user enters the mnemonic of a static text control and the next control is either another static text control or a disabled control, Windows searches for a control that is nonstatic and enabled. In some cases, it may be preferable to disable the mnemonic of a static text control when the control it labels is also disabled. For more information, please query in the Microsoft Knowledge Base on the following word:

mnemonic

MORE INFORMATION

The dialog box described by the following dialog box template might be displayed by an application when the user chooses Open from the File menu:

```
IDD_FILEOPEN DIALOG LOADONCALL MOVEABLE DISCARDABLE 9, 22, 178, 112
CAPTION "File Open..."
STYLE WS_CAPTION | DS_MODALFRAME | WS_SYSMENU | WS_VISIBLE | WS_POPUP
BEGIN
   CONTROL "File&name:", ID_NULL, "static",
           SS_LEFT | WS_GROUP | WS_CHILD, 5, 5, 33, 8
   CONTROL "", ID_NAMEEDIT, "edit",
           ES_LEFT | ES_AUTOHSCROLL | WS_BORDER | WS_TABSTOP
           | WS_CHILD | ES_OEMCONVERT, 40, 4, 90, 12
   CONTROL "Directory:", ID_NULL, "static", SS_LEFT | WS_CHILD,
           5, 20, 35, 8
   CONTROL "", ID_PATH, "static", SS_LEFT | WS_CHILD, 40, 20, 91, 8
   CONTROL "&Files:", ID_NULL, "static", SS_LEFT | WS_GROUP
           | WS_CHILD, 5, 33, 21, 8
   CONTROL "", ID_FILELIST, "listbox", LBS_NOTIFY | LBS_SORT
           | LBS_STANDARD | LBS_HASSTRINGS | WS_BORDER | WS_VSCROLL
           | WS_TABSTOP | WS_CHILD, 5, 43, 66, 65
   CONTROL "&Directories:", ID_NULL, "static", SS_LEFT | WS_GROUP
           | WS_CHILD, 75, 33, 49, 8
   CONTROL "", ID_DIRLIST, "listbox", LBS_NOTIFY | LBS_SORT
           | LBS_STANDARD | LBS_HASSTRINGS | WS_BORDER | WS_VSCROLL
           | WS_TABSTOP | WS_CHILD, 75, 43, 65, 65
   CONTROL "OK", IDOK, "button", BS_DEFPUSHBUTTON | WS_TABSTOP
           | WS_CHILD, 139, 4, 35, 14
   CONTROL "Cancel", IDCANCEL, "button", BS_PUSHBUTTON | WS_TABSTOP
           | WS_CHILD, 139, 23, 35, 14
END
```

In this dialog box, one static text control, with identifier ID_PATH, is used to display the current path. The other four static text controls label other controls, as follows:

```
   "File&name"         labels the ID_NAMEEDIT edit control
   "Directory"         labels the ID_PATH static control display
   "&Files"            labels the ID_FILELIST list box
   "&Directories"      labels the ID_DIRLIST list box
```

When the user enters the key combination ALT+N, Windows sets the focus to the edit control identified in the dialog template as ID_NAMEEDIT, because it is the next enabled nonstatic control. If that edit control was disabled by the EnableWindow function, pressing ALT+N would move the focus to the next enabled nonstatic control. This control would be the list box identified as ID_FILELIST.

Note that the static control "Directory" has no mnemonic; therefore, keyboard input does not affect it.

When the user enters ALT+F, the focus moves to the ID_FILELIST list box, if it is enabled. In the same manner, ALT+D moves the focus to the ID_DIRLIST list box.

If ID_DIRBOX is disabled, ALT+D moves the focus to the OK button, the next enabled nonstatic control. Windows treats this as if the user pressed and released the mouse button over the OK button. For more information on how to prevent this behavior, query the Microsoft

Knowledge Base on the following word:

mnemonic

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95 radio shortcut
KBCategory: kbprg
KBSubcategory: UsrCtl

Add-On Allows SystemParameterInfo() to Get/Set System

PSS ID Number: Q125695
Authored 01-Feb-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

The SystemParameterInfo() API provides functionality to set and get many Windows System Parameters based on the action flag if Windows extension #1 is installed. Here are some examples:

- The Full Window Drag system parameter can be retrieved or set by using SPI_GETDRAGFULLWINDOWS/SPI_SETDRAGFULLWINDOWS as the action flag in the SystemParameterInfo() API. When full drag windows is enabled, users can move entire windows instead of just moving the outline. This makes it easier to see how a window looks while being resized. This feature is not available with Windows, so calling SystemParameterInfo using the SPI_GETDRAGFULLWINDOWS or SPI_SETDRAGFULLWINDOWS flag will always fail. However, this feature will be available if an add-on product, Windows extension #1, is installed.

- The Font Smoothing setting can be retrieved or set by using SPI_GETFONTSMOOTHING or SPI_SETFONTSMOOTHING as the action flag in the SystemParameterInfo() API. Enabling this system setting tells Windows to draw smooth characters using font anti-aliasing. This feature is not available with Windows. Thus the call to SystemParameterInfo with the SPI_GETFONTSMOOTHING or SPI_SETFONTSMOOTHING flag will always fail. However, this Font Smoothing feature will be available if an add-on product, Windows extension #1, is installed.

- SystemParameterInfo(SPI_GETWINDOWSEXTENSION, 1, 0, 0) returns true if Windows extension #1 is installed otherwise false. The Second parameter denotes Windows extension #1.

Additional reference words: 4.00 95
KBCategory: kbprg
KBSubcategory: UsrWnd

Adding a Custom Template to a Common Dialog Box

PSS ID Number: Q86720
Authored 15-Jul-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, version 3.5

SUMMARY

Many applications developed for the Microsoft Windows environment using dialog boxes from the common dialogs library (COMMDLG.DLL) require custom dialog templates. An application generally uses a custom dialog box template to add controls to a standard common dialog box. The text below discusses the steps required to implement a custom dialog box template with a common dialog box.

A custom dialog box template is most often used in conjunction with a hook function. For details on using a hook function with one of the common dialog boxes, query on the following words in the Microsoft Knowledge Base:

steps adding hook function

MORE INFORMATION

CDDEMO, one of the advanced sample applications provided with version 3.1 of the Microsoft Windows Software Development Kit (SDK), demonstrates adding a hook function to the File Open dialog box. The five steps required to modify the CDDEMO application to use a custom dialog box template in its File Open dialog box are as follows:

1. Edit the FILEOPEN.DLG template in the Windows SDK advanced sample applications directory (by default, C:\WINDEV\SAMPLES\COMMDLG). All existing controls must remain in the dialog template; add additional controls, if desired. To demonstrate the process, make a copy of the FILEOPEN.DLG template and include it in the CDDEMO.RC file. Modify the title of the "Cancel" button to "CANCEL." Renaming the button minimizes the potential for error while demonstrating that the application loaded the custom dialog box template.

2. In the application, modify the Flags member of the OPENFILENAME data structure to include the OFN_ENABLETEMPLATE initialization flag.

3. Specify MAKEINTRESOURCE(FILEOPENORD) as the value of the lpTemplateName member of the OPENFILENAME data structure.

4. Specify ghInst as the value of the hInstance member of the OPENFILENAME data structure.

5. Use the #include directive to include DLGS.H in the CDDEMO.RC file.

If an application adds a hook function to a common dialog box, the hook receives all messages addressed to the dialog box. With the exception of the WM_INITDIALOG message, the hook function receives messages before its associated common dialog box does. If the hook function processes a message completely, it returns TRUE. If the common dialog box must provide default processing for a message, the hook function returns FALSE.

In the hook function, the application should process messages for any new controls added through the custom dialog box template. If the standard common dialog box template contains a control that is unnecessary in a particular application, hide the control when the hook function processes the WM_INITDIALOG message. Use the ShowWindow() API to hide a control; do not delete any controls from the common dialog box template. To indicate that the common dialog boxes DLL does not function properly if any controls are missing, the debug version of Windows displays FatalExit 0x0007.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: UsrCmnDlg

Adding a Hook Function to a Common Dialog Box

PSS ID Number: Q86721
Authored 15-Jul-1992 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

Many applications developed for the Microsoft Windows environment using dialog boxes from the common dialogs library (COMMDLG.DLL) require hook functions. A hook function for one of the common dialog boxes is similar to a subclass procedure for a standard Window control, such as an edit control. Through a hook function, an application can process all messages addressed to the dialog box. The text below discusses the steps required to implement a hook function with a common dialog box.

A hook function is most often used in conjunction with a custom dialog template. For details using a custom dialog template with one of the common dialog boxes, query on the following words in the Microsoft Knowledge Base:

steps add custom template

MORE INFORMATION

The hook function receives all messages addressed to a common dialog box. With the exception of the WM_INITDIALOG message, the hook function receives messages before its associated common dialog box does. If the hook function processes a message completely, it returns TRUE. If the common dialog box must provide default processing for a message, the hook function returns FALSE.

CDDEMO, one of the advanced sample applications provided with version 3.1 of the Microsoft Windows Software Development Kit (SDK), demonstrates adding a hook function to the File Open dialog box. The eight steps involved in this process are as follows:

1. Add the standard common dialog box to the application without the hook function.

2. In the application's module definition (DEF) file, list the hook procedure name (for example, MyHookProc) in the EXPORTS section.

3. Define a FARPROC variable (for example, lpfnHookProc)

4. In the application, before completing the OPENFILENAME data structure, call the MakeProcInstance function to create a procedure instance address for the hook procedure.

5. Set the lpfnHook member of the OPENFILENAME data structure to the procedure address of the hook function.

6. Specify OFN_ENABLEHOOK as one of the initialization flags in the Flags member of the OPENFILENAME structure.

7. Code the hook function to process messages as required. A sample hook function follows below.

8. After the user dismisses the common dialog box, call the FreeProcInstance function to free the procedure instance address.

The following code is a sample hook function:

```
BOOL FAR PASCAL MyHookProc(HWND hDlg, unsigned message,
                           WORD wParam, LONG lParam)
{
   switch (message)
   {
      case WM_INITDIALOG:
         OutputDebugString("Hello hook function!");
         return TRUE;

      case WM_COMMAND:
         switch(wParam)
         {
            case IDD_MYNEWCONTROL:
            // Perform appropriate processing here...
               return TRUE;

            default:
               break;
         }
         break;

      default:
         break;
   }
   return FALSE;
}
```

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCmnDlg

Adding Categories for Events

PSS ID Number: Q115947
Authored 07-Jun-1994 Last modified 03-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1, 3.5, and 3.51

SUMMARY

The ReportEvent() API accepts a category ID as one of the arguments. The "Microsoft Win32 Programmer's Reference" states that you can add your own categories for events. This article shows you how to add categories that will be recognized by the Event Viewer; however, the article assumes that you already know how to create a message file and add an event source to the registry. For information about basic event logging, please see the logging sample in the Q_A\SAMPLES\logging directory on the "Win32 SDK" CD.

NOTE: The logging example that comes with Visual C++ 2.0 does not bind the MESSAGES.RC to the MESSAGES.DLL unlike the logging example that accompanies the Win32 SDK or NSDN Level 2. Binding MESSAGES.RC to the MESSAGES.DLL can be accomplished by adding MESSAGES.RC to MESSAGES.MAK.

NOTE: If you notice that there are some entries that have a .DLL name and a driver name while you are attempting to read messages from the event log, this means that the event message source has more than one message file. This means you need to parse the string and load each message file.

MORE INFORMATION

Just like events, category IDs are simply IDs in message file resources. However, in order to use categories, the following two requirements must be met:

- The category IDs must be sequentially numbered, starting with a message ID of 1.

- The event source entry in the registry must specify the category message file and the number of categories in the message file.

The first requirement is simply a matter of setting the MessageID entries in the message file for the categories. If all of your categories are listed at the top of the message file, you can assign the ID of 1 to the first message. Each message after that automatically gets the next ID value unless you specify otherwise in the MessageID entry.

The category entries in the registry are made by adding values to your event key. Normally, your event log application key already contains EventMessageFile and TypesSupported entries. You should add the following two entries:

- CategoryMessageFile

- CategoryCount

The CategoryMessageFile entry is of type REG_EXPAND_SZ. It should be set to the full path to the message file that contains the categories.

The CategoryCount entry is a REG_DWORD type. You should set this entry to the number of categories in the message file specified in CategoryMessageFile.

REFERENCES

"Microsoft Win32 Programmer's Reference," Microsoft Corporation.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseMisc

Adding Custom Error Strings to an MCI Device Driver

PSS ID Number: Q76411
Authored 23-Sep-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

In the Microsoft Windows graphical environment, an application can use mciGetErrorString() to obtain the string associated with an error code returned from a media control interface (MCI) device driver.

An MCI driver can use a STRINGTABLE resource to store its error strings. The identifier constant for each string should correspond to the error value returned by the DriverProc function. This value must be greater than or equal to the constant MCIERR_CUSTOM_DRIVER_BASE to avoid confusion with the predefined MCI error codes.

MORE INFORMATION

For example, the Pioneer LaserDisc device driver, MCIPIONR.DRV, included with the Microsoft Device Development Kit (DDK) for Windows 3.1, contains the following declarations in its header file, MCIPIONR.H:

```
#define MCIERR_PIONEER_ILLEGAL_FOR_CLV  (MCIERR_CUSTOM_DRIVER_BASE)
#define MCIERR_PIONEER_NOT_SPINNING     (MCIERR_CUSTOM_DRIVER_BASE+1)
#define MCIERR_PIONEER_NO_CHAPTERS      (MCIERR_CUSTOM_DRIVER_BASE+2)
#define MCIERR_PIONEER_NO_TIMERS        (MCIERR_CUSTOM_DRIVER_BASE+3)
```

Its resource file, MCIPIONR.RC, contains the following STRINGTABLE definition:

```
STRINGTABLE
BEGIN
  MCIERR_PIONEER_ILLEGAL_FOR_CLV, "Illegal operation for CLV type disc."
  MCIERR_PIONEER_NOT_SPINNING, "The disc must be spun up to perform this \
operation."
  MCIERR_PIONEER_NO_CHAPTERS, "Chapters are not supported for this disc."
  MCIERR_PIONEER_NO_TIMERS, "All timers are in use. Cannot enable \
notification."
END
```

When mciGetErrorString() receives a value greater than or equal to MCIERR_CUSTOM_DRIVER_BASE, it looks in the driver's resource table for a string with the corresponding identifier.

Additional reference words: 3.00 3.10 3.50 4.00 95

KBCategory: kbmm kbprg
KBSubcategory: MMMisc

Adding Point Sizes to the ChooseFont() Common Dialog Box

PSS ID Number: Q99668
Authored 03-Jun-1993 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

When a TrueType font (or any other scalable font) is selected in the ChooseFont() common dialog box, a list of reasonable point sizes is displayed for selection. In some cases it is necessary to change this list to allow fewer or more selections.

The initial list of point sizes is hard-coded into COMMDLG.DLL but can be changed programmatically using a common dialog box hook function.

MORE INFORMATION

Most scalable fonts can be created at nearly any point size. Some TrueType fonts can be sized from 4 points to 999 points. A complete list of available point sizes for a font of this type would contain about 1000 elements, which can be prohibitively long and time-consuming to construct.

The ChooseFont() common dialog box attempts to limit the selection by listing only a few of the available point sizes in the size selection combo box.

In certain cases, it may be desirable to offer more point-size selections in this dialog box. In this case, a common dialog box hook procedure can be used to insert point sizes when a specific font is selected.

The following steps describe a technique that will allow new point sizes to be added to the size selection combo box. Pay special attention to step number 4:

1. In your common dialog box hook procedure, look for WM_COMMAND messages with wParam equal to the font name combo box (which is "cmb1" for Windows version 3.1).

2. When you get this message(s), check for the font name you are looking for (for example, you can compare the current selection to "Courier New"). If it's not the font you want, return.

3. If this is the font you want, post yourself a user-defined message. In response to this message, add the new point size to the Point Size combo box (which is "cmb3" for Windows 3.1). It's a good

idea to double-check here that the point size you are adding isn't already in the combo box (so you don't get duplicates).

4. Once you add the new point size, set the item data for the new item equal to the point size you are adding. For example, if you are adding the string "15" to the combo box, you need to set the item data of this new item to 15.

The following code fragment demonstrates the above steps:

```
// Common Dialog ChooseFont() hook procedure.

UINT CALLBACK __export FontHook(HWND hwnd, UINT wm, WPARAM wParam,
LPARAM lParam)
{
   char szBuf[150];
   DWORD dwIndex;

   switch(wm)
   {
      case WM_COMMAND:
      // See if the notification is for the "Font name" combo box.
         if (wParam == cmb1)
         {
            switch (HIWORD(lParam))
            {
               case CBN_SETFOCUS:
               case CBN_KILLFOCUS:
                  break;
               default:
               // Check to see if it's the font we're looking for.
                  dwIndex = SendDlgItemMessage(hwnd, cmb1, CB_GETCURSEL,
                                               0, 0);
                  if (dwIndex != CB_ERR)
                     SendDlgItemMessage(hwnd, cmb1, CB_GETLBTEXT, (WPARAM)
                                        dwIndex, (LPARAM) ((LPSTR) szBuf));

               // Compare list box contents to the font we are looking for.
               // In this case, it's "Courier New".
                  if (strcmp(szBuf, "Courier New") == 0)
                  // It's the font we want. Post ourselves a message.
                     PostMessage(hwnd, WM_ADDNEWPOINTSIZES, 0, 0L);
            }
         }
      break;

      case WM_ADDNEWPOINTSIZES:
      // First look to see if we've already added point sizes to this
      // combo box.
         if (SendDlgItemMessage(hwnd, cmb3, CB_FINDSTRING, -1,
                                (LPARAM)(LPCSTR)"6") == CB_ERR)
         {
         // Not found, add new point size.
            dwIndex = SendDlgItemMessage(hwnd, cmb3, CB_INSERTSTRING,
                                         0, (LPARAM)(LPSTR)"6");
```

```
         // Also set the item data equal to the point size.
            SendDlgItemMessage(hwnd, cmb3, CB_SETITEMDATA,
                               (WPARAM)dwIndex, 6);
         }
         return TRUE; // Don't pass this message on.
   }
   return FALSE;
}
```

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCmnDlg

Adding to or Removing Windows from the Task List

PSS ID Number: Q99800
Authored 08-Jun-1993 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1 and 3.5

There are no functions included in the Windows Software Development Kit (SDK) to add or remove windows from the task list. All top-level windows (that is, windows without parents) that are visible will automatically appear in the task list. A window can be removed from the task list by making it hidden. Call ShowWindow() with the SW_HIDE parameter to hide the window. To make it visible again, call ShowWindow() with the appropriate parameter such as SW_SHOW or SW_SHOWNORMAL.

Additional reference words: 3.00 3.10 3.50 tasklist
KBCategory: kbprg
KBSubcategory: UsrMisc

Additional Information for WIN32_FIND_DATA

PSS ID Number: Q120697
Authored 18-Sep-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

This article contains additional information about the WIN32_FIND_DATA members.

The WIN32_FIND_DATA structure contains three members that store the creation, last access, and last write time of a file. The time format for these three members (ftCreationTime, ftLastAccessTime, and ftLastWriteTime) are expressed in the Universal Time Convention (UTC). These three data members can be converted from UTC time to local time by calling the FileTimeToLocalFileTime api.

The WIN32_FIND_DATA structure contains two members that store the file size: nFileSizeHigh and nFileSizeLow. They are described as being the high and low order words of the size, but they are actually DWORDs. Therefore, nFileSizeHigh will be zero unless the file size is greater than 0xffffffff (4.2 Gig).

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseMisc

Additional Remote Debugging Requirement

PSS ID Number: Q106066
Authored 31-Oct-1993 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

The printed and online documentation for remote debugging with WinDbgRm fail to mention one requirement. The binaries must be in the same drive and directory on both the target machine and the development machine.

WinDbg also expects to find the source files in the same directory in which the the binary file was built, but will browse for the source if it is not found in this location. WinDbg will automatically locate the source if the files are specified to the compiler with fully qualified paths.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsWindbg

Administrator Access to Files

PSS ID Number: Q102099
Authored 28-Jul-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

A user that is a member of the Administrator group is not automatically granted access to any file on the local machine. For an administrator to access a file, permission must be specifically granted (as for any user) in the file's discretionary access control list (DACL).

If an administrator wants to access a file that he or she is not granted access to, the administrator must first take ownership of that file. Once ownership is taken, the administrator will have full access to the file. It is important to note that administrator cannot give ownership back to the original owner. If this were so, the administrator could take ownership of a file, examine it, and then assign it back to the original owner without that owner's knowledge.

NOTE: Because administrators have backup privileges, an administrator could back up a file (or entire volume) and restore it onto another system. The administrator could then take ownership of a file on this new system without the owner's knowledge. Please keep this in mind when thinking about file security.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

Advanced Graphics Settings Slider under Windows 95

PSS ID Number: Q127066
Authored 12-Mar-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

The following table shows exactly what the Advanced Graphics Settings Slider control does:

Slider Control	Hardware Cursor	Memory Mapped I/O	Accelerated Functions
None	No	No	Safe mode level 2
Basic	No	No	Safe mode level 1
Most	No	Yes	All
Full	Yes	Yes	All

NOTES:

- The hardware cursor adjusts the SWCursor switch in SYSTEM.INI [display] section. It applies only to S3 and WD drivers.

- The Memory Mapped I/O adjusts the MMIO switch in SYSTEM.INI [display] section. It applies only to the S3 driver.

- Accelerated Functions adjusts the SafeMode switch in WIN.INI [Windows] section. It applies to all DIB engine minidrivers. Level 1 allows basic, safe acceleration, such as srcCopy bitblt, patblt, and so on. Level 2 completely bypasses the driver for all operations (except cursor).

Additional reference words: 4.00 95 Video SYSTEM Applet
KBCategory: kbprg
KBSubcategory: UsrCtl

Advantages of Device-Dependent Bitmaps

PSS ID Number: Q94918
Authored 25-Jan-1993 Last modified 16-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

A DDB (device-dependent bitmap) is much faster than a DIB (device independent-bitmap) to BitBlt(). For this reason, it is often a good strategy under Win32 (as well as under Windows 3.1) to create a DDB from a DIB when caching or calling *Blt() functions.

The slight drawback of memory overhead for the DDB is handled well by Win32. Under Windows 3.1, the DDB memory could be marked as discardable. Under Win32, the memory will be paged out if system resources become tight (at least until the next repaint); if the memory is marked as PAGE_READONLY, it can be efficiently reused, [see VirtualProtect() in the Win32 application programming interface (API) Help file].

However, saving the DDB to disk as a mechanism for transfer to other applications or for later display (another invocation) is not recommended. This is because DDBs are driver and driver version dependent. DDBs do not have header information, which is needed for proper translation if passed to another driver or, potentially, to a later version of the driver for the same card.

MORE INFORMATION

Windows 95 and Windows NT 3.5 and later support DIBSections. DIBSections are the fastest and easiest to manipulate, giving the speed of DDBs with direct access to the DIB bits. NOTE: Win32s does not support DIBSections.

Additional reference words: 3.10 3.50 4.00
KBCategory: kbprg
KBSubcategory: GdiBmp

Allocating and Using Class and Window Extra Bytes

PSS ID Number: Q34611
Authored 22-Aug-1988 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The WNDCLASS structure contains two fields, cbClsExtra and cbWndExtra, which can be used to specify a number of additional bytes of memory to be allocated to the class structure itself or to each window created using that class.

MORE INFORMATION

Every application that uses class extra bytes and window extra bytes must specify the appropriate number of bytes before the window class is registered. If no bytes are specified, an attempt to store information in extra bytes will cause the application to write into some random portion of Windows memory, causing data corruption.

Windows version 3.1 will FatalExit if extra bytes are used improperly.

If an application does not use class extra bytes or window extra bytes, it is important that the cbClsExtra and cbWndExtra fields be set to zero.

Class and window extra bytes are a scarce resource. If more than 4 extra bytes are required, use the GlobalAlloc fucntion to allocate a block of memory and store the handle in class or window extra bytes.

Class Extra Bytes

For example, setting the value of the cbClsExtra field to 4 will cause 4 extra bytes to be added to the end of the class structure when the class is registered. This memory is accessible by all windows of that class. The number of additional bytes allocated to a window's class can be retrieved through the following call to the GetClassWord function:

```
nClassExtraBytes = GetClassWord(hWnd, GCW_CBCLSEXTRA);
```

The additional memory can be accessed one word at a time by specifying an offset, in BYTES (starting at 0), as the nIndex parameter in calls to the GetClassWord function. These values can be set using the SetClassWord function.

The GetClassLong and SetClassLong functions perform in a similar manner and get or set four bytes of memory respectively:

```
   nClassExtraBytes = GetClassLong(hWnd, GCL_CBCLSEXTRA);
```

NOTE: A Win32-based application should use GetClassLong and SetClassLong, because the GCW_ indices are obsolete under Win32.

Window Extra Bytes

Assigning a value to cbWndExtra will cause additional memory to be allocated for each window of the class. If, for example, cbWndExtra is set to 4, every window created using that class will have 4 extra bytes allocated for it. This memory is accessible only by using the GetWindowWord and GetWindowLong functions, and specifying a handle to the window. These values can be set by calling the SetClassWord or SetClassLong functions. As with the class structures, the offset is always specified in bytes.

An example of using window extra bytes would be a text editor that has a variable number of files open at once. The file handle and other file-specific variables can be stored in the window extra bytes of the corresponding text window. This eliminates the requirement to always consume memory for the maximum number of handles or to search a data structure each time a window is opened or closed.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrWndw

AllocConsole() Necessary to Get Valid Handles

PSS ID Number: Q89750
Authored 29-Sep-1992 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

If a graphical user interface (GUI) application redirects a standard handle, such as stderr or stdout, and then spawns a child process, the output of the child process will not be seen unless the AllocConsole() application programming interface (API) is called before the standard handle is redirected.

If an application spawns a child process without calling AllocConsole() first, the child's console window will appear on the screen and the GUI application will not be able to control this window (for example, it cannot minimize the child window). In addition, users can terminate the child process by choosing Close from the console window's Control (system) menu. This causes users to think that only the window is closed, when in actuality, the entire application is terminated. This can cause the user to lose data in the console window.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseCon

Allowing Only One Application Instance on Win32s

PSS ID Number: Q124134
Authored 19-Dec-1994 Last modified 21-Feb-1995

The information in this article applies to:

- Microsoft Win32s, versions 1.1, 1.15, 1.15a, and 1.2

SUMMARY

The entry point for both Windows-based and Win32-based applications is:

```
int WinMain( hInstance, hPrevInstance, lpszCmdLine, nCmdShow )

HINSTANCE hInstance;        /* Handle of current instance  */
HINSTANCE hPrevInstance;    /* Handle of previous instance */
LPSTR lpszCmdLine;          /* Address of command line     */
int nCmdShow;               /* Show state of window        */
```

You can allow only one instance of your Windows-based application to run at a time by using hPrevInstance to determine if there is already an existing application instance; then exit the process if there is one. If there is no previous instance, hPrevInstance is NULL.

However, in a Win32-based application, hPrevInstance is always NULL. Therefore, you cannot determine if another instance of your application has been started simply by examining hPrevInstance. This article gives you a method you can use.

MORE INFORMATION

Use one of the following four methods to determine if there is an existing application instance on Win32s:

- Synchronize with a named object, such as a file mapping.

  -or-

- Synchronize with a global atom.

  -or-

- Synchronize with a private message.

  -or-

- Use FindWindow() to check for the application.

Using a File Mapping

Using a file mapping works well on any Win32 platform. The global atom is a cheaper resource, whereas a file mapping will cost a page of memory. A private message is good if you want to inform the first instance that the user attempted to start a second instance, and then let it handle the request -- post a message, become the active application, and so on.

NOTE: You need to clean up before terminating the second instance. FindWindow() doesn't require cleanup, but this method will not work as reliably in a preemptive multitasking environment, such as Windows NT, because you can get in a race condition.

The following code fragment demonstrates how a file mapping can be used to allow only one instance of a Win32-based application. This code should avoid any race conditions. Place this code at the beginning of WinMain().

The code creates a file mapping named MyTestMap using CreateFileMapping(). If MyTestMap already exists, then you know that there is already a running instance of this application. A similar technique would be used with a global atom.

Sample Code

```
   HANDLE hMapping;

   hMapping = CreateFileMapping( (HANDLE) 0xffffffff,
                                 NULL,
                                 PAGE_READONLY,
                                 0,
                                 32,
                                 "MyTestMap" );
   if( hMapping )
   {
      if( GetLastError() == ERROR_ALREADY_EXISTS )
      {
      //
      // Display something that tells the user
      // the app is already running.
      //
         MessageBox( NULL, "Application is running.", "Test", MB_OK );
         ExitProcess(1);
      }
   }
   else
   {
   //
   // Some other error; handle error.
   //
      MessageBox( NULL, "Error creating mapping", "Test", MB_OK );
      ExitProcess(1);
   }
```

Additional reference words: 1.10 1.20
KBCategory: kbprg kbcode
KBSubcategory: W32s

Alternative to PtInRegion() for Hit-Testing

PSS ID Number: Q121960
Authored 24-Oct-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

It may be useful to perform hit-testing on an object that is defined by a polygon. To accomplish this, you could call CreatePolygonRgn() to create a region from the polygon, and then call PtInRegion() to determine if the point falls within the region. However, this method can be expensive both in terms of GDI resources, and in terms of speed. If a polygon is complex, CreatePolygonRgn() will often fail due to lack of memory in Windows because regions are in GDI's heap.

The code below provides a better method. Use it to determine if a point lies within a polygon. It is fast and does not use regions. The trick lies in determining the number of times an imaginary line drawn from the point you want to test crosses edges of your polygon. If the line crosses edges an even number of times, the point is outside the polygon. If it crosses an odd number of times it is inside. The line is drawn horizontally from the point to the right.

MORE INFORMATION

The following code is based on an algorithm presented in "Algorithms" by Robert Sedgewick, Addison-Wesley, 1988, 2nd ed. ISBN 0201066734. The algorithm is on p.354, in the section "Inclusion in a Polygon" in the chapter "Elementary Geometric Methods." It is also discussed in "Computer Graphics" by Foley, van Dam, Feiner and Hughes, Addison-Wesley, 1990, 2nd ed. ISBN 0201121107, chapter 2, section 1, p.34.

Sample Code

```
#include "windows.h"
#include "limits.h"
BOOL G_PtInPolygon(POINT *rgpts, WORD wnumpts, POINT ptTest,
```

```
                    RECT *prbound) ;
BOOL G_PtInPolyRect(POINT *rgpts, WORD wnumpts, POINT ptTest,
                     RECT *prbound) ;
BOOL Intersect(POINT p1, POINT p2, POINT p3, POINT p4) ;
int  CCW(POINT p0, POINT p1, POINT p2) ;

/*********************************************************************

 * FUNCTION:   G_PtInPolygon
 *
 * PURPOSE
 * This routine determines if the point passed is in the polygon. It uses

 * the classical polygon hit-testing algorithm: a horizontal ray starting

 * at the point is extended infinitely rightwards and the number of
 * polygon edges that intersect the ray are counted. If the number is odd,

 * the point is inside the polygon.
 *
 * RETURN VALUE
 * (BOOL) TRUE if the point is inside the polygon, FALSE if not.
 *********************************************************************/

BOOL G_PtInPolygon(POINT *rgpts, WORD wnumpts, POINT ptTest,
                    RECT *prbound)
{
   RECT   r ;
   POINT  *ppt ;
   WORD   i ;
   POINT  pt1, pt2 ;
   WORD   wnumintsct = 0 ;

   if (!G_PtInPolyRect(rgpts,wnumpts,ptTest,prbound))
      return FALSE ;

   pt1 = pt2 = ptTest ;
   pt2.x = r.right + 50 ;

   // Now go through each of the lines in the polygon and see if it
   // intersects
   for (i = 0, ppt = rgpts ; i < wnumpts-1 ; i++, ppt++)
   {
      if (Intersect(ptTest, pt2, *ppt, *(ppt+1)))
         wnumintsct++ ;
   }

   // And the last line
   if (Intersect(ptTest, pt2, *ppt, *rgpts))
      wnumintsct++ ;

   return (wnumintsct&1) ;
}

/*********************************************************************
```

```
 * FUNCTION:   G_PtInPolyRect
 *
 * PURPOSE
 * This routine determines if a point is within the smallest rectangle
 * that encloses a polygon.
 *
 * RETURN VALUE
 * (BOOL) TRUE or FALSE depending on whether the point is in the rect or

 * not.
 ************************************************************************/

BOOL  G_PtInPolyRect(POINT *rgpts, WORD wnumpts, POINT ptTest,
                     RECT *prbound)
{
   RECT r ;
   // If a bounding rect has not been passed in, calculate it
   if (prbound)
      r = *prbound ;
   else
   {
      int   xmin, xmax, ymin, ymax ;
      POINT *ppt ;
      WORD  i ;

      xmin = ymin = INT_MAX ;
      xmax = ymax = -INT_MAX ;

      for (i=0, ppt = rgpts ; i < wnumpts ; i++, ppt++)
      {
         if (ppt->x < xmin)
            xmin = ppt->x ;
         if (ppt->x > xmax)
            xmax = ppt->x ;
         if (ppt->y < ymin)
            ymin = ppt->y ;
         if (ppt->y > ymax)
            ymax = ppt->y ;
      }
      SetRect(&r, xmin, ymin, xmax, ymax) ;
   }
   return (PtInRect(&r,ptTest)) ;
}

/************************************************************************

 * FUNCTION:   Intersect
 *
 * PURPOSE
 * Given two line segments, determine if they intersect.
 *
 * RETURN VALUE
 * TRUE if they intersect, FALSE if not.
 ************************************************************************/
```

```
BOOL Intersect(POINT p1, POINT p2, POINT p3, POINT p4)
{
   return ((( CCW(p1, p2, p3) * CCW(p1, p2, p4)) <= 0)
        && (( CCW(p3, p4, p1) * CCW(p3, p4, p2)  <= 0) )) ;
}

/*************************************************************************

 * FUNCTION:   CCW (CounterClockWise)
 *
 * PURPOSE
 * Determines, given three points, if when travelling from the first to
 * the second to the third, we travel in a counterclockwise direction.
 *
 * RETURN VALUE
 * (int) 1 if the movement is in a counterclockwise direction, -1 if
 * not.
 *************************************************************************/

int CCW(POINT p0, POINT p1, POINT p2)
{
   LONG dx1, dx2 ;
   LONG dy1, dy2 ;

   dx1 = p1.x - p0.x ; dx2 = p2.x - p0.x ;
   dy1 = p1.y - p0.y ; dy2 = p2.y - p0.y ;

   /* This is basically a slope comparison: we don't do divisions because

    * of divide by zero possibilities with pure horizontal and pure
    * vertical lines.
    */
   return ((dx1 * dy2 > dy1 * dx2) ? 1 : -1) ;
}

/*********************************************
 * The above code might be tested as follows:
 *********************************************/
void PASCAL TestProc( HWND hWnd )
{
    POINT rgpts[] = {0,0, 10,0, 10,10, 5,15, 0,10};
    WORD wnumpts = 5;
    POINT ptTest = {3,10};
    RECT prbound = {0, 0, 20, 20};
    BOOL bInside;

    bInside = G_PtInPolygon(rgpts, wnumpts, ptTest, &prbound);

    if (bInside)
       MessageBox(hWnd, "Point is inside!", "Test", MB_OK );
    else
       MessageBox(hWnd, "Point is outside!", "Test", MB_OK );
}
/* code ends */
```

Additional reference words: 3.00 3.10 3.50 4.00 95 hittest hit-test fails
KBCategory: kbprg kbcode
KBSubcategory: GdiMisc

Alternatives to Using GetProcAddress() With LoadLibrary()

PSS ID Number: Q92862
Authored 16-Nov-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

When loading a DLL with LoadLibrary(), an alternative to calling GetProcAddress() for each of your DLL entry points is to have the DLL initialization function initialize a global structure or array containing the addresses of these DLL entry points, then call a DLL function from your executable which will return the address of this structure or array to your executable. You can then call your DLL functions via the function pointers in this structure or array.

MORE INFORMATION

The best place to initialize this structure or array of function pointers would be in the DLL_PROCESS_ATTACH code of your DLL's main entry point. The structure or array containing these function pointers must be declared as either a global variable or as dynamically allocated memory (malloc(), GlobalAlloc(), etc.) in your DLL in order for the executable to be able to address this memory properly.

It is also possible, though not as clean, to export the global structure or array of function pointers so that your executable can use the structure or array by name directly in your executable. For more information on how to declare and export global data in a Win32 DLL, please see the following article(s) in the Microsoft Knowledge Base:

```
ARTICLE-ID: Q90530
TITLE     : Exporting Data From a DLL
```

Be careful not to call these DLL functions via the function pointers after the DLL is unloaded via FreeLibrary(). After FreeLibrary() is called, these function pointer addresses are invalid and calling them will result in an access violation.

This technique of returning pointers to DLL entry points is a supported technique and will work on all hardware platforms that Windows NT supports.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseDll

An Efficient Animation Algorithm

PSS ID Number: Q75431
Authored 20-Aug-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

An application that shows an animated image cannot rely solely on Windows graphical device interface (GDI) functions because they will be too slow. Instead, the application must create its own set of bitmaps. This article discusses the process required and provides tips to improve performance and memory use.

This information applies to any type of animation or fast drawing, from painting the game pieces in Reversi to updating the time each second in Clock.

MORE INFORMATION

There are three major steps to this process:

1. Allocate the bitmap.

It is preferable to allocate a single bitmap to store all the different "cels"--the components of the animated scene. The contents of the bitmap should be arranged in a column that is wide enough to hold a single cel; the height is determined by the number of cels. To improve memory usage, the bitmap should be discardable.

For example, given the definitions of the three constants below, the following code allocates the correct size bitmap:

```
X_SIZE = width of the cel
Y_SIZE = height of the cel
NUM_CELS = number of cels in the animated sequence

HBITMAP hbm;

hbm = CreateDiscardableBitmap(hDC, X_SIZE, NUM_CELS * Y_SIZE);
if (!hbm)
    {
    // error - could not allocate bitmap
    }
```

2. Prepare the bitmaps.

To draw into the bitmap, it must be selected into a display context (DC). Allocate a (temporary) compatible DC for this purpose:

```
if (hTmpDC = CreateCompatibleDC(hDC))
    {
    HBITMAP hOldBm;

    hOldBm = SelectObject(hTmpDC, hbm);
    // and so forth
    }
```

In many cases, all cels will share the same background. Rather than drawing this background several times onto the bitmap, draw it once onto the first cel and copy it to the other cels, as the following code demonstrates:

```
// GDI calls to draw to hbm from (0, 0) to (X_SIZE, Y_SIZE)

for (i = 1; i < NUM_CELS; i++)  // Perform the copy
    BitBlt(hTmpDC, 0, i * Y_SIZE, X_SIZE, Y_SIZE, hTmpDC, 0, 0,
               SRCCOPY);
```

After the background is copied, draw the foreground on each cel, using regular GDI calls (in TRANSPARENT drawing mode). The coordinates for cel "i" in bitmap hbm are:

```
x_pos: 0 to (X_SIZE - 1)
y_pos: (i * Y_SIZE) to (((i + 1) * Y_SIZE) - 1)
```

If the cels in the bitmap contain sequential images, animating to the screen is simplified.

To finish this step, release the temporary DC.

```
SelectObject(hTmpDC, hOldBm);
DeleteDC(hTmpDC);
```

3. Animate.

A temporary, off-screen DC is required to allow the application to select the bitmap. Note that selecting the object may fail if the bitmap has been discarded. If this has occurred, another bitmap must be allocated (if memory allows) and the bitmap must be initialized (as outlined in step 2, above).

```
if ((hTmpDC = CreateCompatibleDC(hDC)) != NULL)
    {
    HBITMAP hOldBm;

    if (!(hOldBm = SelectObject(hTmpDC, hbm))
        // must re-allocate bitmap. Note that this MAY FAIL!!!
```

At this point, call the BitBlt() function to copy the various stages of the animation sequence to the screen. If the cels in the bitmap contain sequential images, a simple loop will do the job nicely, as the following code demonstrates:

```
        for (i = 0; i < NUM_CELS; i++)
            {
            BitBlt (hDC, x_pos, y_pos, X_SIZE, Y_SIZE, hTmpDC, 0,
                        i * Y_SIZE, SRCCOPY);

            // Some form of delay goes here. A real-time wait, based on
            // clock ticks, is recommended.
            }
```

When the drawing is done, delete the temporary DC:

```
   SelectObject(hTmpDC, hOldBm);
   DeleteDC(hTmpDC);
```

It is important to cancel the selection of the bitmap between passes through the for loop. This allows the bitmap to be discarded if the system runs low on memory.

Additional reference words: 3.00 3.10 3.50 4.00 95 animation
KBCategory: kbprg
KBSubcategory: GdiBmp

Application Can Allocate Memory with DdeCreateDataHandle

PSS ID Number: Q85680
Authored 17-Jun-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

An application can use the DdeCreateDataHandle function to create a handle to a block of data. The application can use the handle to pass the data to another application in a dynamic data exchange (DDE) conversation using the Dynamic Data Exchange Management Libraries (DDEML). All DDEML functions refer to blocks of memory using data handles.

An application can allocate memory and manually create a data handle associated with the memory (using method 1 below), or automatically by using the DdeCreateDataHandle function (method 2 below).

Method 1
----

1. Obtain a block of memory using the GlobalAlloc or LocalAlloc function or by declaring a variable in your application.

2. Fill the block of memory with the desired data.

3. Call the DdeCreateDataHandle function to create a data handle associated with the block of memory.

Method 2
----

1. Call the DdeCreateDataHandle function with the lpvSrcBuf parameter set to NULL, the cbInitData parameter set to zero, and the offSrcBuf parameter set to the number of bytes of memory required.

2. To retrieve a handle to the memory block, specify the data handle returned by DdeCreateDataHandle as the hData parameter of the DdeAccessData function. This operation is similar to calling the GlobalLock function on a handle returned from GlobalAlloc.

3. Use the pointer to fill the memory block with data.

4. Call DdeUnaccessData to unaccess the object. This operation is similar to calling the GlobalUnlock function on a handle returned from GlobalAlloc.

The following code fragment demonstrates method 2:

```
   // Retrieve the length of the data to be stored
   cbLen = lstrlen("This is a test") + 1;

   // Create the data handle and allocate the memory
   hData = DdeCreateDataHandle(idInst, NULL, 0, cbLen,
                               hszItem, wFmt, 0);

   // Access the data handle
   lpstrData = (LPSTR)DdeAccessData(hData, NULL);

   // Fill the block of memory
   lstrcpy(lpstrData, "This is a test");

   // Unaccess the data handle
   DdeUnaccessData(hData);
```

When an application obtains a data handle from DdeCreateDataHandle, the application should next call DdeAccessData with the handle. If a data handle is first specified as a parameter to a DDEML function other than DdeAccessData, when the application later calls DdeAccessData, the application receives only read access to the associated memory block.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDde

Application Exception Error Codes

PSS ID Number: Q101774
Authored 21-Jul-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

Many exception errors are not processed by applications. The most common exception error is EXCEPTION_ACCESS_VIOLATION (c0000005). It occurs when a pointer is dereferenced and the pointer points to inaccessible memory or a write operation is attempted on read-only memory. If an application does not trap an exception, the Win32 module, UnhandledExceptionFilter, will do one of the following: display a message box, invoke Dr. Watson, or attach your application to a debugger.

The following are standard exception errors:

```
EXCEPTION_ACCESS_VIOLATION
EXCEPTION_ARRAY_BOUNDS_EXCEEDED
EXCEPTION_BREAKPOINT
EXCEPTION_DATATYPE_MISALIGNMENT
EXCEPTION_FLT_DENORMAL_OPERAND
EXCEPTION_FLT_DIVIDE_BY_ZERO
EXCEPTION_FLT_INEXACT_RESULT
EXCEPTION_FLT_INVALID_OPERATION
EXCEPTION_FLT_OVERFLOW
EXCEPTION_FLT_STACK_CHECK
EXCEPTION_FLT_UNDERFLOW
EXCEPTION_ILLEGAL_INSTRUCTION
EXCEPTION_IN_PAGE_ERROR
EXCEPTION_INT_DIVIDE_BY_ZERO
EXCEPTION_INT_OVERFLOW
EXCEPTION_INVALID_DISPOSITION
EXCEPTION_NONCONTINUABLE_EXCEPTION
EXCEPTION_PRIV_INSTRUCTION
EXCEPTION_SINGLE_STEP
EXCEPTION_STACK_OVERFLOW
```

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseExcept

Application Version Marking in Windows 95

PSS ID Number: Q125705
Authored 01-Feb-1995 Last modified 16-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

Applications designed for Windows 95, whether 16- or 32-bit, should be marked for Windows version 4.0 so they receive the full benefit of new user interface features in Windows 95. Applications marked as being designed for earlier versions of Windows will display behavior consistent with the Windows version 3.1 user interface, which is not always identical to Windows 95 behavior.

Executables marked for Windows version 4.0 will load on Windows 95, Windows NT version 3.5, Win32s version 1.15, and later versions, but not on earlier versions.

NOTE: This article deals solely with marking executable files as compatible with a particular Windows version. This is different from the version resources (VS_VERSION_INFO) that may be contained in an executable.

MORE INFORMATION

The Microsoft Visual C++ version 2.1 linker defaults to marking executables for Windows version 4.0 while the version 2.0 linker defaults to 3.1. To override the default, the Microsoft Visual C++ linkers accept the following syntax for the /SUBSYSTEM option:

```
/SUBSYSTEM:WINDOWS,4.0
```

In the development environment, you can change the /SUBSYSTEM option by going to the Project menu, selecting Settings, selecting either Win32 Debug or Win32 Release, choosing the Link tab, and editing the Project Options.

You may need to perform a full link for this to take effect, but subsequent incremental linking with this switch will work correctly.

To mark a 16-bit executable as Windows version 4.0 compatible, use the 16-bit resource compiler (RC.EXE) from the Windows 95 SDK to bind the resources into the executable file. By default, this version of RC marks the executable for version 4.0, but this can be overridden by using the -30 or -31 switch.

An application will display several behavioral differences depending on which Windows version the application is compatible with:

1. All standard control windows owned by a version 4.0 application are

drawn with a chiseled 3D look. The same effect can be obtained for dialogs owned by a version 3.1 application by using the DS_3DLOOK dialog style. (This style is ignored on other Windows platforms.)

2. Thunks created using the Windows 95 SDK Thunk Compiler will not work unless the 16-bit thunk DLL is marked for version 4.0.

3. Windows version 3.1 allowed 16-bit applications to share GDI resources such as font or bitmap handles. For backwards compatibility, Windows 95 does not clean up objects left undeleted by a 16-bit version 3.x application when that application terminates because these objects may be in use by another application. Instead, such objects remain valid as long as there are any 16-bit applications running. When all 16-bit applications are closed, these objects are freed.

   On the other hand, is is assumed that 16-bit version 4.0 applications follow the Windows 95 guidelines, and do not share objects. Thus, all objects owned by a 16-bit version 4.0-based application are freed when the application terminates.

   Win32-based applications cannot freely share GDI objects, so all owned objects are freed when a Win32-based application terminates regardless of the version of the application.

4. New window messages, such as WM_STYLECHANGED, are only sent to windows owned by a version 4.0 application.

5. Printer DCs are reset during StartPage() in applications marked with version 4.0

Additional reference words: 4.00
KBCategory: kbprg
KBSubcategory: UsrMisc

Assigning Mnemonics to Owner-Draw Push Buttons

PSS ID Number: Q67716
Authored 12-Dec-1990 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

An application that uses owner-draw push buttons is always responsible for the appearance of the buttons. It might seem that in doing so, the ability to assign a mnemonic character to an owner-draw button is lost because text containing the mnemonic may not be displayed.

Fortunately, this is not the case. If an owner-draw button should be activated by ALT+X, place "&X" into the button text. NOTE: You have to use DrawText() to get the & character to underline the next character. Using TextOut() will not cause the & character to underline the next character in the string.

When the ALT key is pressed in combination with any character, Windows examines the text of each control to determine which control, if any, uses that particular mnemonic. With an owner-draw button, the text exists, but may not necessarily be used to paint the button.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Associating Data with a List Box Entry

PSS ID Number: Q74345
Authored 16-Jul-1991 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

In the Microsoft Windows graphical environment, an application can use the LB_SETITEMDATA and LB_GETITEMDATA messages to associate additional information with each entry in a list box. These messages enable an application to associate an arbitrary LONG value with each entry and to retrieve that value. This article documents how an application uses these messages.

MORE INFORMATION

In this example, the application will associate a 64-byte block of data with each list box entry. This is accomplished by allocating a global memory block and using the LB_SETITEMDATA message to associate the handle of the memory block with the appropriate list box item.

During list box initialization, the following code is executed for each list box item:

```
   if ((hLBData = GlobalAlloc(GMEM_MOVEABLE, 64)))
      {
      if ((lpLBData = GlobalLock(hLBData)))
         {
         // Store data in 64-byte block.

         GlobalUnlock(hLBData);
         }
      }
// NOTE: The MAKELONG is not needed on 32-bit platforms.
   SendMessage(hListBox, LB_SETITEMDATA, nIndex, MAKELONG(hLBData, 0));
```

To retrieve the information associated with a list box entry, the following code can be used:

```
// NOTE: The return from LB_GETITEMDATA is a long on 32-bit platforms.
   if ((hLBData = LOWORD(SendMessage(hListBox, LB_GETITEMDATA,
            nIndex, 0L))))
      {
      if ((lpLBData = GlobalLock(hLBData)))
         {
         // Access or manipulate the data or both.
```

```
         GlobalUnlock(hLBData);
         }
      }
```

Before the application terminates, it must free the memory associated with each list box item. The following code frees the memory associated with one list box item:

```
   if ((hLBData = LOWORD(SendMessage(hListBox, LB_GETITEMDATA,
         nIndex, 0L))))
      GlobalFree(hLBData);
```

These techniques can be used to associated data with an entry in a combo box by substituting the CB_SETITEMDATA and CB_GETITEMDATA messages.

Additional reference words: 3.00 3.10 3.50 4.00 95 combobox listbox
KBCategory: kbprg
KBSubcategory: UsrCtl

Authoring Windows Help Files for Performance

PSS ID Number: Q74937
Authored 05-Aug-1991 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The Windows Help Compiler allows an application developer to create hypertext documentation, richly annotated with color and graphics. This article discusses methods to author help files to achieve maximum performance when the file is used. These comments apply to Windows Help versions 3.06 and later.

This information is subject to change in future versions of the Help Compiler and of the Help application as new technology is incorporated into these products.

There are four major suggestions:

1. Use OPTCDROM=1 for all files destined for CD-ROM (compact-disk read-only memory), and potentially on files where up to an additional 10K of size is not significant.

2. Use bitmaps placed with data for small bitmaps that are referenced only once or infrequently in the help file.

3. Use bitmaps not placed with data for large bitmaps, all bitmaps referenced frequently, or bitmaps referenced by two or more topics that generally will be viewed in succession.

4. Use segmented hypergraphics to generate graphics with multiple hot spots, rather than several bitmaps positioned next to each other.

Coincidentally, suggestions 2, 3, and 4 are also generally space-saving techniques as well.

MORE INFORMATION

1. Use OPTCDROM=1 for files destined for CD-ROM.

   When OPTCDROM is placed in the [OPTIONS] section of the .HPJ file, the topic information in the help file is aligned on 2K boundaries. This option is aimed at maximizing performance on CD-ROM drives, where reading aligned information can be significantly faster.

   Estimates indicate that sequential reads from CD-ROM can be up to twice as fast when aligned. While reads are not always sequential,

a high percentage can be, depending on how the help file is authored. Minor improvements have also been noted on magnetic media.

This option can cause up to 10K of additional file space to be used.

2. Store small bitmaps with data.

   Placing bitmaps with data keeps the graphical and textual information in the same location in the help file. This avoids reading from different locations on disk to display a topic. Seeks to different locations are exceptionally time consuming on CD-ROM, and can be time consuming on magnetic media. Bitmaps with data also help maximize the effects of the OPTCDROM option.

   Up to 12K of compressed topic text is buffered in memory. Since bitmaps are kept with the topic text, they take advantage of this buffering. Thus, topics under 12K in compressed size generally do not need to be reread when the window is resized or redrawn.

   Note that large bitmaps kept with data may cause performance to become worse. Topics may be laid out twice as part of scroll bar removal. Topics over 12K in size will be read, laid out, and then reread as they are laid out a second time before display. Bitmaps are often the cause of exceeding the 12K size. Bitmaps NOT kept with data are buffered elsewhere (see #3 below), and if the remaining textual data is less than 12K, the topic will be read from disk only once.

   Bitmaps with data cost space only when the same bitmap is used more than once in the help file. If a bitmap is used frequently, not placing it "with data" may be more appropriate, unless the performance benefit is determined to outweigh the size hit.

3. Store large and frequently accessed bitmaps separately.

   The 50 most recently accessed bitmaps not stored with data are cached in memory. This means that the bitmap may have to be read from disk only once if two successively displayed help topics reference it. Unlike bitmaps stored with data, these cached bitmaps only have to be reread if they are bumped out of the cache by the subsequent display of 50 more bitmaps, or by low memory conditions.

   Cached bitmaps not stored with data cost some speed when they are typically displayed by the user only once in a session. Since they are stored in a different portion of the help file from the topic text, an additional seek is required to locate them. This cost, if incurred, is generally negligible on disk, and high on CD-ROM. Note, however, that this cost MAY be less than the cost of reading the topic twice, if the topic is laid out twice as a result of not needing scroll bars, and is larger than 12K.

4. Use segmented hypergraphics.

Segmented hypergraphics is the term used to describe the ability to take a single bitmap, and define several regions that are hot spots. Hot spots can overlap, and can each perform different actions.

The primary benefit of using segmented hypergraphics is that a single bitmap can be used. Previous techniques utilizing several bitmaps carefully placed in the topic text to generate a single image have the drawback of requiring several bitmap-locating operations at display time, which can translate to several disk seeks and reads. On CD-ROM, especially, this can be quite significant.

The only cost involved in using segmented hypergraphics is that the segmented hypergraphics editor must be used to define the hot spots within the bitmaps.

In summary, there are a few simple authoring techniques that can improve performance of help files. While especially significant for CD-ROM hosted help files, they are of benefit to disk-based help files.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsHlp

Availability of Microsoft Network SDKs

PSS ID Number: Q115604
Authored 31-May-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), version 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0
- Windows for Workgroups SDK, version 3.11
- Microsoft LAN Manager SDK

SUMMARY

The following SDKs are available for programmers writing network applications on Microsoft platforms:

Microsoft Windows for Workgroups SDK

This is available on the April 1994 (and later) Microsoft Developer Network (MSDN) Level 2 CDs.

The SDK, documentation, debug kernel, and other files for network programmers on Windows for Workgroups, version 3.11 are available on the April 1994 MSDN. This SDK includes APIs for the network extensions to Windows for Workgroups, version 3.11.

Microsoft LAN Manager SDK

This is available on CompuServe:

Forum: msnet
Section: 3 (LM on OS/2)

The libraries in this section contain the SDK.

The SDK for LAN Manager programming on MS-DOS, Windows, and OS/2 is available on CompuServe.

The documentation is available in two books:

- "Microsoft LAN Manager, A Programmer's Guide," Ralph Ryan, Microsoft Press, ISBN 1-55615-166-7.

- "Microsoft LAN Manager Programmer's Reference," Microsoft Corporation, Microsoft Press, ISBN 1-55615-313-9.

This SDK includes APIs for writing distributed applications and administration programs for Microsoft LAN Manager.

LAN Manager APIs for Windows NT

The file DOC\SDK\MISC\LMAPI.HLP on the CD describes the ported LAN Manager APIs APIs. Windows NT supports 32-bit equivalents of most of the LAN Manager APIs.

The LAN Manager APIs are included in the header file LMACCESS.H and in the import library NETAPI32.LIB. These APIs are described in the Win32 API help file.

Windows Sockets APIs

The Windows Sockets APIs are available through the SDKs. The file WINSOCK.HLP gives the details.

The files needed to support Windows Sockets APIs (conforming to the Windows Sockets specifications) for Microsoft LAN Manager may be obtained in the following locations:

- CompuServe (lib 10 in the MSNETWORKS forum, titled LMSOCK.ZIP).

  -or-

- Internet (FTP.MICROSOFT.COM, ADVSYS\LANMAN\SUP-ED\WINSOCK).

The Windows Sockets specifications, libraries, header files and samples may also be obtained from the Internet location mentioned above, in the ADVSYS\WINSOCK directory.

Microsoft Remote Procedure Call (RPC)

The support for RPC is included in the Win32 SDKs. The Win32 API help file includes the RPC APIs. The RPC.HLP file gives the details of RPC support.

RPC support files for MS-DOS and Microsoft Windows may be obtained from the Win32SDK CD-ROM. The directory MSTOOLS\RPC_DOS has the required files.

NOTE: Microsoft also provides other kinds of network APIs. For example, the NetBIOSCall() API provides a mechanism to write NetBIOS applications on Microsoft Windows; on Windows NT this is accomplished with the Netbios() API. Information on these other APIs may be obtained from the Windows and Win32 SDK documentation.

Additional reference words: 3.10 3.50 4.00 95 3.11 msdnl2
KBCategory: kbref
KBSubcategory: NtwkMisc

Avoid Calling SendMessage() Inside a Hook Filter Function

PSS ID Number: Q74857
Authored 31-Jul-1991 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

A hook filter function should not call SendMessage() to pass intertask messages because this behavior can create a deadlock condition in Windows. If a hook filter function is called as a result of an intertask SendMessage(), and if the hook function then yields control with an intertask SendMessage(), a message deadlock condition may occur. For this reason, hook filters should use PostMessage() rather than SendMessage() to pass messages to other applications.

NOTE: A hook filter can use SendMessage() to pass a message to the current task because this will not yield the control.

Section 1.1.5 of the "Microsoft Windows Software Development Kit Reference Volume 1" from the Windows SDK version 3.0 documentation has more information on message deadlocks.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrHks

Background, Foreground, and System Palette Management

PSS ID Number: Q72386
Authored 23-May-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

On a device that supports the Microsoft Windows palette management APIs, an application can create a logical palette, select the palette into a device context (DC), and realize the palette. Realizing a logical palette maps its colors to the system (hardware) color palette. The GetDeviceCaps() API is available to inform an application whether the device is capable of supporting palette management functions and, if so, the size of its system palette. This article discusses the different types of logical palettes and the effect of each on the system palette when a logical palette is realized.

MORE INFORMATION

When a logical palette is selected into a DC, it can be selected as either a foreground or a background palette. Setting the bForceBackground parameter of the SelectPalette() API to TRUE selects the palette as a background palette. If this parameter is FALSE, the palette can be selected as a foreground palette. A palette will be selected as a foreground palette only if the DC into which the palette is selected is one of the five cached DCs managed by the GetDC() API and the DC is retrieved on behalf of the active window. If the DC is returned by CreateDC() or CreateCompatibleDC() or if the window is not the active window, the palette will be forced into the background.

The status as a foreground or a background palette affects how the colors in the logical palette are mapped into the system palette when the logical palette is realized.

When a foreground palette is realized, every entry in the system palette that can be modified by an application is accessible to the logical palette. Logical palette entries are mapped into the system palette starting at the first available entry. Because a logical palette entry that exactly matches a reserved system palette entry is mapped to that system entry, it does not consume a separate palette slot. If the logical palette has more entries than available slots in the system palette, the available slots are filled, in order, from the logical palette. The remaining logical palette entries are mapped to the closest colors already present in the system palette. There is one exception to this rule: if a logical palette entry is marked with the PC_RESERVED flag, no colors will be mapped to that entry. If all available system palette entries are reserved, additional

colors will not be mapped to any entry and will be displayed as black on the screen.

A palette entry marked as PC_NOCOLLAPSE will always take a separate slot if available, just as for PC_RESERVED. Unlike a PC_RESERVED color, if no slots are available, it will map to the nearest color, and other colors may map onto it.

The first available entry in the system palette is the first palette entry not marked as used. For example, assume a device with 256 palette entries, 20 of which are reserved for the system. An application realizes a palette of 36 colors on this device; therefore, the first 36 entries are marked used. Another application realizes a 100-entry palette; therefore, the next 100 entries are marked used. If a third application receives the input focus and realizes a foreground palette with 236 entries, Windows maps the first 100 colors into the remainder of the system palette. Each of the remaining 136 colors of the logical palette is mapped into the closest color available in the system palette.

When a background palette is realized, any empty positions in the system palette are filled. Any colors that remain are mapped to the closest color in the system palette. A background palette entry cannot overlay a foreground entry in the system palette; however, a foreground palette entry can overlay a background entry in the system palette.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiPal

Background Colors Affect BitBlt() from Mono to Color

PSS ID Number: Q41464
Authored 22-Feb-1989 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

When using BitBlt() to convert a monochrome bitmap to a color bitmap, GDI transforms all white bits (1) to the background color of the destination device context (DC). GDI transforms all black bits (0) to the text (or foreground) color of the destination DC.

When using BitBlt() to convert a color bitmap to a monochrome bitmap, GDI sets to white (1) all pixels that match the background color of the source DC. All other bits are set to black (0).

These features are mentioned in the BitBlt() documentation.
manual.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiBmp

BeginPaint() Invalid Rectangle in Client Coordinates

PSS ID Number: Q19963
Authored 17-Dec-1987 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

The BeginPaint() function returns a pointer to a PAINTSTRUCT data structure through its second parameter. The rcPaint field of this structure specifies the update rectangle in client-area coordinates (relative to the upper-left corner of the window client area).

This update rectangle also serves as the clipping area for painting in the window, unless the invalid area of the window is expanded using the InvalidateRect() function.

Additional reference words: 3.00 3.10 3.5 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrPnt

Binary and Source Compatibility Under Windows NT

PSS ID Number: Q93213
Authored 01-Dec-1992 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

There are currently three hardware platforms for which Win32-based applications can be written; they are Intel (x86), MIPS, and DEC Alpha. Binary compatibility across these hardware platforms is not a viable alternative. Therefore, Windows NT offers source compatibility. This means that developers may create versions of their Win32 applications for each CPU by recompiling.

For example, suppose that you have written a Win32-based application in C and have used a 32-bit compiler that targets the Intel platform. In order to produce a Win32-based application that runs on DEC Alpha, you would purchase a 32-bit compiler that targets Alpha and recompile your code.

MORE INFORMATION

It is important not to confuse source compatibility across hardware platforms with binary compatibility across application execution environments. Windows NT and Win32s do support binary compatibility between different application execution environments. For example, the typical 16-bit Windows-based application can be run without modification on any Windows NT machine. On the MIPS and Alpha platforms, this is achieved through emulation. In addition, it is possible to design an x86 Win32-based application so that it runs on Windows 3.1. This is achieved through Win32s.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: SubSys

Binary Compatibility Basics

PSS ID Number: Q95162
Authored 02-Feb-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

Because of differing instruction sets between processors, object code is not binary-compatible across platforms. To enable object code compiled for processor X to run on processor Y, a virtual machine must be created on processor Y to emulate processor X's instructions, which results in a considerable performance hit. For this reason, Windows NT offers a strong source compatibility, which makes it a relatively simple matter to compile the same code into native object code for each platform.

The Hardware Abstraction Layer (HAL), smooths over differences between hardware implementations. All access to the hardware from the operating system (OS) goes through the HAL, which makes changing of both the hardware and the OS itself much simpler. The HAL, however, does not emulate the instruction set of the platform; a common misconception.

MORE INFORMATION

Source code is compiled into executable or object code for the instruction set of a specific processor or family (x86, 680x0, R3000/4000, and so forth). This code runs natively only on that type of processor. Remember, this reduces to 1's and 0's, hence binary, so even if two processors have a large overlap in the available instructions (and they do; MOVs, XORs, or whatever at the assembly language level), what is actually executed is not even relatively "high level" human-readable assembly code but merely a series of bytes that by convention/definition only are assigned the meanings that are human-readable at the assembly language level.

For example, probably every microprocessor has an OR instruction. On the Intel x86, the OR instruction may be represented in the instruction stream by the bytes "09 [effective address]" (see page 456 of Microsoft's 80386/80486 Programming Guide, 2nd Ed.). On the R4000, however, it's nearly guaranteed to be something different.

Thus, if you want to run an executable compiled for Intel on a MIPS chip, you must run a program that behaves as an Intel instruction interpreter (similar to a Basic interpreter, but much more complex). Such a program is called an emulator, and will scan through the Intel object code and then in turn execute equivalent commands in MIPS form on the processor. But the emulator must do much more than that; it must also create a "virtual machine," a complete software execution environment that behaves similar to the original hardware/software environment that the executable was

originally compiled for.

Note that even in an ideal case, every processor X instruction requires about 5-20 instructions on processor Y. The object code interpreter must examine the next instruction/byte, compare its value to known instruction values via some kind of table (depending on the implementation), and then execute the equivalent native instruction. There is room for optimization but it will never be very fast (relative to native code).

Therefore, run non-native object code only if you absolutely must. Below is a binary compatibility table for Windows NT:

System	Binary-Compatible on NT?
Win16 apps	Yes (via Insignia's x86 emulation code)
Win32 apps	No (must re-compile)
POSIX apps	No (POSIX is a source-code standard)(1)
OS/2 1.x apps	No (Don't run on non-x86 machines at all)(2)

Notes: (1) POSIX 1003.1 is a C-language source-level standard for basic operating system services. POSIX applications don't need to be binary-compatible even on the same instruction set! That is, a POSIX application compiled and linked for SCO on x86 will NOT run on x86 Windows NT POSIX or x86 Sun/Interactive POSIX. (2) OS/2 1.x support is technically feasible but would have required more work on the non-Intel platforms, and was not considered a high priority.

The Hardware Abstraction Layer (HAL)

A common misconception is that the HAL should allow binary compatibility. The Windows NT HAL has absolutely nothing to do with the issue discussed above; that is, running code complied for processor X on processor Y. Nor is the HAL akin to a "virtual machine" implementation; it does not simulate anything. Rather, the HAL is simply an example of a good modular software design that deals with the issue of differences in hardware design between machines with the same instruction set (or between instruction sets).

The HAL provides a set of services to the rest of the Windows NT executive that abstracts and "hides" the differences between low- level hardware-software interfaces, such as with DMA controllers, programmable interrupt controllers, clocks and timers, and so forth. In a typical pre-Windows NT operating system, there is lots of code embedded throughout the operating system (particularly in device drivers) that is specifically tied to particular implementations of hardware services (a particular PIC, a particular DMA chip, and so forth). If one of these hardware pieces is changed, lots of code scattered throughout the system will break. As a result, the hardware never changed and a typical 486/66 machine today uses the same low-function hardware devices that appear in the IBM AT 286 (if not the IBM PC itself).

In Windows NT, any other part of the OS (including the kernel and all device drivers) that needs to deal with those low-level devices now uses HAL services, and is therefore isolated from changes in the hardware. If

you change those hardware pieces you only need to change the HAL. This results in at least the following two advantages:

- A cleaner, easier to write and debug design for the OS and device drivers.
- The real possibility for innovation and change in the underlying hardware.

But the HAL does not provide an abstract instruction set, or the services necessary for running object code from processor X on processor Y. It is possible to write an such as OS if you are willing to take the huge performance hit, but Windows NT isn't it. And because a well-designed OS such as Windows NT can provide a very complete level of source-code compatibility across instruction sets, and therefore a relatively painless way of getting native-code versions of Win32-based applications, probably no one will be willing to take that performance hit in a mainstream operating system, no matter how fast the hardware.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: SubSys

Broadcasting Messages Using PostMessage() & SendMessage()

PSS ID Number: Q64296
Authored 28-Jul-1990 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

When SendMessage() is used to send a broadcast message (hwnd = 0xFFFF or hwnd = -1), the message is sent to all top-level windows. A message broadcast by PostMessage() is only sent to top-level windows that are visible, enabled, and have no owner.

You might observe the effect of the difference when, for example, the top-level window of your application calls DialogBox() to present a modal dialog box. While the modal dialog box exists, its owner (your top-level window) will be disabled. Messages broadcast using PostMessage() will not reach the top-level window because the window is disabled, and will not reach the dialog box because the dialog box has an owner. Messages broadcast using SendMessage() will reach both the top-level window and the dialog.

In Windows 3.1, PostMessage() will broadcast to invisible and disabled windows just like SendMessage() already does.

Both PostMessage() and SendMessage() actually broadcast using the same broadcast procedure. This procedure does some additional screening to make sure that pop-up menus, the task manager window, and icon title windows are insulated from broadcast messages.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMsg

BUG: Bad Characters in 32-bit App on Win32s on Russian

PSS ID Number: Q126865
Authored 06-Mar-1995 Last modified 24-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5
- Microsoft Win32s versions 1.2 and 1.25

SYMPTOMS

When compiling, linking, and running a Russian application which was built using Microsoft Visual C++ version 2 under on Windows NT version 3.5 the application runs correctly. If this application is run on Russian Windows version 3.1 with the Win32s libraries version 1.2 or 1.25 installed, some parts of the user interface appear as meaningless set of characters.

CAUSE

The strings in the resources are stored in UNICODE format. Yet you must pass all strings to Windows 3.1 in ANSI format. The drawing of the resources is done by Windows 3.1. Win32s simply reads the 32-bit resources, converts them to the 16-bit format equivalent, and passes the resources to Windows 3.1. One stage of the conversion is to convert the UNICODE strings into ANSI strings. This conversion for the Russian language is broken.

STATUS

Microsoft has confirmed this to be a problem in the product(s) listed at the beginning of this article. This bug will be fixed in the next release of Win32s.

Additional reference words: Cyrillic Russia 2.00 garbage corruption
KBCategory: kbbuglist
KBSubcategory: WIntlDev W32s

BUG: Console Applications Do Not Receive Signals on Windows

PSS ID Number: Q130717
Authored 25-May-1995 Last modified 30-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SYMPTOMS

Console applications call SetConsoleCtrlHandler() to install or remove application-defined callback functions to handle signals. On Windows 95, the signal handler function only gets called for the CTRL_C_EVENT and CTRL_BREAK_EVENT signals; the signal handler function is never called for the CTRL_SHUTDOWN_EVENT, CTRL_LOGOFF_EVENT, and CTRL_CLOSE_EVENT signals.

CAUSE

Windows 95 sends CTRL_C_EVENT and CTRL_BREAK_EVENT signals to console applications that have installed signal handers, but does not send CTRL_SHUTDOWN_EVENT, CTRL_LOGOFF_EVENT, or CTRL_CLOSE_EVENT signals.

STATUS

Microsoft has confirmed this to be a bug in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: Win95 4.00 console event signal
KBCategory: kbprg kbbuglist
KBSubcategory: BseCon

BUG: CreateDC Does Not Thunk DEVMODE Structure Correctly

PSS ID Number: Q128701
Authored 06-Apr-1995 Last modified 07-Apr-1995

The information in this article applies to:

- Microsoft Win32s, versions 1.2 and 1.25

SYMPTOMS

Under Win32s version 1.2 and 1.25, when a Win32-based application displays a printer setup dialog box that calls PrintDlg with the PD_PRINTSETUP flag and the printing orientation is changed from Portrait to Landscape, the DEVMODE structure obtained from the PrintDlg call to CreateDC is passed and creates a printer DC. If the printer driver is a postscript driver, anything printed using this DC is still in Portrait mode.

CAUSE

The thunking layer for CreateDC does not thunk the DEVMODE structure for the postcript driver correctly.

RESOLUTION

To work around the problem, avoid calling CreateDC with a DEVMODE structure. Instead, directly create the pinter DC by calling PrintDlg with the PD_RETURNDC flag. Then change the printing orientation from within the print dialog box.

STATUS

Microsoft has confirmed this to be a bug in the Microsoft products listed at the beginning of this article. We are reasearching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: 1.20 1.25 3.10 3.50
KBCategory: kbprg kbbuglist
KBSubcategory: GdiPrn

BUG: ESC/ENTER Keys Don't Work When Editing Labels in

PSS ID Number: Q130691
Authored 24-May-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.51 and 4.0
- Microsoft Win32s version 1.3

SYMPTOMS

When editing labels in a TreeView control, you should be able to press the ESC key to cancel the changes or press the ENTER key to accept the changes. However, when the TreeView control is contained in a dialog box, IsDialogMessage processes the ESC and ENTER keystrokes and does not pass them on to the edit control created by the TreeView control, so the keystrokes have no effect.

CAUSE

The TreeView control creates and subclasses the edit control used for in-place editing. The subclass procedure does not process the WM_GETDLGCODE and WM_CHAR messages for the edit control properly.

RESOLUTION

To work around the problem, subclass the edit control and return DLGC_WANTALLKEYS in response to the WM_GETDLGCODE message. Then process the WM_CHAR messages for VK_ESCAPE and VK_RETURN.

To subclass the edit control, obtain the handle to the edit control by using the TVM_GETEDITCONTROL message in response to the TVN_BEGINLABELEDIT notification. Remove the subclassing when the TVN_ENDLABELEDIT notification is received.

In response to the WM_CHAR|VK_ESCAPE message, have the application send the TVM_ENDEDITLABELNOW with fCancel = TRUE message to cancel the edit. In response to the WM_CHAR|VK_ENTER message, have the application send the TVM_ENDEDITLABELNOW with fCancel = FALSE message to accept the edit.

All other WM_CHAR messages should be passed on to the default edit control window procedure.

STATUS

Microsoft has confirmed this to be a bug in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: 1.30 4.00 95
KBCategory: kbprg kbui kbbuglist
KBSubcategory: UsrCtl W32s

BUG: FindFirstFile() Does Not Handle Wildcard (?) Correctly

PSS ID Number: Q130860
Authored 30-May-1995 Last modified 12-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 4.0

SYMPTOMS

In Windows 95, the FindFirstFile() function interprets a wildcard (?) as "any character" instead of "zero or one character," its true meaning. This incorrect interpretation causes some searches to return invalid results. For example, if the files, TEMP.TXT and TEMPTEMP.TXT, are in the same directory, the following code finds the TEMPTEMP.TXT file, but not the TEMP.TXT file:

```
HANDLE hFind;
WIN32_FIND_DATA findData = {0};

hFind = FindFirstFile ("TEM?????.???", &findData);

if (hFind == INVALID_HANDLE_VALUE)
   MessageBox (hwnd, "FindFirstFile() failed.", NULL, MB_OK);
else
{
   do
   {
      MessageBox (hwnd, findData.cFileName, "File found", MB_OK);
   }
   while (FindNextFile(hFind, &findData));

   CloseHandle (hFind);
}
```

Windows NT correctly finds both the TEMP.TXT and TEMPTEMP.TXT files.

WORKAROUND

To work around this problem, choose an alternative wildcard search and apply further processing to eliminate files that are found by the alternative search, but do not match the original search. For example, the code above could be changed to search for TEM*.* instead of TEM?????.???. Then you could make an additional test for filenames that are up to 8 characters in length, followed by a ".", followed by up to 3 more characters (8.3).

STATUS

Microsoft has confirmed this to be a bug in the Microsoft products

listed at the beginning of this article. We are researching this bug and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: win95 4.00 regular expression wild
KBCategory: kbprg kbbuglist
KBSubcategory:

BUG: GetKerningPairs Sometimes Fails on Win32s Version 1.2

PSS ID Number: Q125872
Authored 07-Feb-1995 Last modified 13-Jun-1995

The information in this article applies to:

- Microsoft Win32s versions 1.2 and 1.25a

SYMPTOMS

GetKerningPairs will sometimes fail on Win32s version 1.2 causing the 32-bit application to exit mysteriously. The problem may only occur once in a while with many successful runs interrupted by a single unsuccessful run.

CAUSE

The thunking layer for GetKerningPairs contains a bug in the code that allocates a temporary buffer passed to the 16-bit version of GetKerningPairs. The errant code executes whenever the number of kerning pairs requested is equal to or less than 128. Requesting GetKerningPairs to return 129 or more kerning pairs forces the thunking layer to use an alternative buffer allocation method.

RESOLUTION

To work around the problem, ensure that the number of kerning pairs requested from GetKerningPairs is greater than 128.

Typically, kerning pairs are retrieved with two calls to GetKerningPairs. The first call retrieves the number of kerning pairs available. A buffer is allocated based on the number of pairs returned. Then the second call to GetKerningPairs retrieves the kerning pairs into the buffer.

To avoid the bug in GetKerningPairs, follow these steps:

1. Retrieve the number of kerning pairs available from GetKerningPairs.

2. Check that this value is greater than 128. If it is less than or equal to 128, reset the variable to an arbitrary value greater than 128 -- like 129.

3. Use the new value to allocate the buffer of KERNINGPAIRS and pass this new value with the buffer to the second GetKerningPairs call.

STATUS

Microsoft has confirmed this to be a bug in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes

available.

Additional reference words: 1.20 font kerning
KBCategory: kbprg kbbuglist
KBSubcategory: GdiFnt

BUG: MDI Child Window's "Minimize" System Menu Disabled

PSS ID Number: Q110795
Authored 28-Jan-1994 Last modified 10-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, version 3.1

SYMPTOMS

After creating a new multiple document interface (MDI) child window in an MDI application, the child window's Minimize System menu option will be disabled after the following actions:

1. The MDI child window is created and in the topmost z-order.

2. The user accesses the MDI child window's System menu through the shortcut keys: ALT+SPACE, RIGHT ARROW.

These actions will cause the child window's System menu to appear with the Minimize menu option disabled. This behavior can be demonstrated with File Manager. From File Manager's Window menu, choose New Window. Next, press ALT+SPACE and then press the RIGHT ARROW key to bring up the new window's System menu. The Minimize option will be disabled.

NOTE: The Minimize menu item is not disabled if the System menu is accessed through the ALT+"-" (minus) shortcut key or the mouse. Also, the Minimize menu item is not disabled if the window is moved.

RESOLUTION

This problem may be avoided if the MDI parent calls GetSystemMenu() after creating the MDI child window. If the parent calls GetSystemMenu (hwndMDIChild, TRUE) for the MDI child window, the System menu is reset to the Windows default state, thus eliminating the problem.

STATUS

Microsoft has confirmed this to be a bug in Windows version 3.1 and Windows NT version 3.1. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

MORE INFORMATION

Although MDI is supported, it is discouraged. New users find it complicated to understand, so Microsoft is encouraging ISVs to use SDI or greatly simplify their interfaces to act like an SDI (there are some advantages to

coding using MDI).

Additional reference words: buglist3.10 3.10
KBCategory: kbprg kbbuglist
KBSubcategory: UsrMdi

BUG: Menu Bar Covered By Main Window Client Area

PSS ID Number: Q109539
Authored 04-Jan-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, version 3.1

SYMPTOMS

In an application, the lower half of the menu bar is covered by the client area of the main window.

CAUSE

This problem can be caused by a command with a null menu string or separator at the end of the menu bar. For example, if the following menus are used in a window, they will display the problem:

```
   PROBLEMMENU MENU DISCARDABLE
   BEGIN
      POPUP "&Popup 1"
         BEGIN
            MENUITEM "&Menu Item 1", 1
         END
      MENUITEM "", 2
   END
```

-or-

```
   PROBLEMMENU MENU DISCARDABLE
   BEGIN
      POPUP "&Popup 1"
         BEGIN
            MENUITEM "&Menu Item 1", 1
         END
      MENUITEM SEPARATOR
   END
```

RESOLUTION

To work around this problem, remove the command with a null menu string or separator at the end of the menu bar.

STATUS

Microsoft has confirmed this to be a bug in the Microsoft products listed

at the beginning of this article. We are researching this bug and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: utoff overlayed view toolbar
KBCategory: kbprg kbbuglist
KBSubcategory: UsrMen

BUG: Pressing SHIFT+ESC Doesn't Generate WM_CHAR on Windows

PSS ID Number: Q129861
Authored 08-May-1995 Last modified 08-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SYMPTOMS

Pressing SHIFT+ESC in an application running under Windows 95 doesn't generate a WM_CHAR message even though it does generates a WM_CHAR message for applications running under Windows version 3.x and Windows NT.

CAUSE

The key table that TranslateMessage on Windows 95 uses to generate the WM_CHAR messages doesn't include the SHIFT+ESC key combination.

RESOLUTION

If you need to use this key combination, use the WM_KEYDOWN or WM_KEYUP messages.

STATUS

Microsoft has confirmed this to be a bug in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available. This bug is scheduled to be corrected in a future version of Windows 95.

Additional reference words: 4.00
KBCategory: kbui kbbuglist
KBSubcategory: UsrInp

BUG: Race Between 2 Threads Sharing a Socket Causes Problem

PSS ID Number: Q126346
Authored 21-Feb-1995 Last modified 01-May-1995

The information in this article applies to:

- Microsoft Win32 Device Development Kit (DDK) for Windows NT version 3.5

SYMPTOMS

Packets between the TCP, UDP, and IP layers are lost.

CAUSE

There appears to be a problem with a race condition between two threads that share a socket where one is closing a socket while the other tries to call recvfrom() on the same socket. This causes problems the next time a socket is bound to the same UDP port.

RESOLUTION

The vendor should implement a workaround within the application so that this race condition does not occur.

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

MORE INFORMATION

This Sockets/UDP problem was discovered while testing the TX1000 NCPI driver for Windows NT. Here are some notes showing what appears to be happening:

```
Thread 1                        Thread 2
--------                        ---------
Create DGRAM socket x
Bind socket x to:
        IPADDR = ANY
        PORT = 1571
Create thread 2 ----------------->
                                RecvFrom on socket x
                                ...
```

```
                                Packet received on x (recvfrom completes)
                <---------------signal main thread
Process rec'd packet            wait for main thread to consume buffer
Signal buffer available--------->
                                RecvFrom on socket x
                                ...  { repeats many times }
```

Normal Shutdown Sequence (on Last Packet)

```
                                Packet received on x (recvfrom completes)
                <---------------signal main thread
Process rec'd packet            wait for main thread to consume buffer
Signal buffer available--------->
Application done                (1) RecvFrom on socket x
(2) close socket x
                                (3) RecvFrom fails with expected error
                                Thread terminates
```

Usually, events occur in sequence (1, 2, then 3). In this normal case, the socket is cleared correctly, and everything works the next time the application runs.

Shutdown Sequence that Causes Problems (on Last Packet)

```
                                Packet received on x (recvfrom completes)
                <---------------signal main thread
Process rec'd packet            wait for main thread to consume buffer
Signal buffer available--------->
Application done
(1) close socket x              (2) RecvFrom on socket x
                                (3) RecvFrom fails with expected error
                                Thread terminates
```

In this case, the sequence is slightly different. The closesocket() function from main thread starts, but does not complete, before thread 2 runs. While thread 1 is suspended awaiting completion of closesocket(), thread 2 runs and posts next recvfrom() on same socket. The closesocket() function completes successfully, and recvfrom() fails as in normal case. But the next time the application runs and binds to the same UDP port, the following occurs:

- All socket calls (bind(), recvfrom()) are successful

- All incoming packets to that UDP port are discarded before reaching the application -- recvfrom() never completes. The following command shows that the "receive errors" count has been incremented once for each incoming packet that was lost:

  netstat -s -p udp

The conclusion is that in this case the socket was not cleaned up properly due to the race condition between the closesocket() and the recvfrom() functions.

Additional reference words: 3.50
KBCategory: kbprg
KBSubcategory: NtwkWinsock

BUG: SetWindowPlacement and ptMin.x or ptMax.x = -1

PSS ID Number: Q110793
Authored 28-Jan-1994 Last modified 13-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, 4.0

SYMPTOMS

Passing an X coordinate value of -1 to SetWindowPlacement causes the parameter to be ignored. If ptMinPosition.x is set to -1, SetWindowPlacement won't reset the minimized window coordinate; this is also true for ptMaxPosition.x. This was undocumented in the Windows 3.1 SDK documentation.

CAUSE

This problem is caused by the use of -1 as a special value. A value of -1 in the X coordinate causes the API (application programming interface) to use the window's current coordinate for the specified parameter.

RESOLUTION

Microsoft has confirmed this to be a bug in the products list above.

This behavior may be a problem for application developers because they may want to set the maximized or minimized horizontal coordinate of a window to -1. To avoid this problem, the developer should trap values of -1, and use a value of -2 or 0 (zero) as appropriate.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbbuglist
KBSubcategory: UsrWndw

BUG: SNMP Service Produces Bad "Error on getproc(InitEx) 127"

PSS ID Number: Q130699
Authored 25-May-1995 Last modified 30-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SYMPTOMS

When the SNMP service is started with debug level 2 or greater, it returns this error message:

```
error on getproc(InitEx) 127
```

RESOLUTION

This error message should be ignored.

STATUS

Microsoft has confirmed this to be a bug in the Microsoft products listed at the beginning of this article. We are reasearching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

MORE INFORMATION

How to Start the SNMP Service

The SNMP service can be started from either the control panel or from a console window.

Type "net help start snmp" in a console box to see how to configure error logging of the SNMP agent.

The syntax of the command is:

```
net start snmp [/logtype: type] [/loglevel: level]
```

where:

- /LOGTYPE: type determines where the log will be created. The possible values are 2 for file, 4 for eventlog, and 6 for both. The default is 4. The file option creates a file under \WINNT\SYSTEM32 called SNMPDBG.LOG.

- /LOGLEVEL: level determines the debug level. The higher the number, the

more the detail obtained. The default is 1 (minimum), and the range is from 1 to 20.

Here is an example:

```
net start snmp /logtype:6 /loglevel:10
```

This starts the SNMP service with loglevel 10, and logs events in the eventlog as well as in SNMPDBG.LOG.

The SNMP service can also be started from a console window without typing "net start." This makes all error messages go to the console window and can be used to help in debugging when writing SNMP applications. For example, the following starts the SNMP service:

```
cmd-prompt> snmp
```

Additional reference words: 3.10 3.50
KBCategory: kbnetwork kbbuglist
KBSubcategory: NtwkSnmp

BUG: Using WM_SETREDRAW w/ TreeView Control Gives Odd Results

PSS ID Number: Q130611
Authored 23-May-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.51 and 4.0

SYMPTOMS

If a program uses the WM_SETREDRAW message to turn off updating of a TreeView control before adding items, the TreeView control can behave strangely.

For example, if the item being added to the control uses the TVI_FIRST style to insert it to the top of the tree and the top of the tree is scrolled above the top of the visible window, it may be impossible to bring the item into view.

Another possible symptom is that the TreeView control doesn't repaint itself at all. These problems occur only if the program has used the WM_SETREDRAW message to turn off updating the TreeView control.

RESOLUTION

Don't use the WM_SETREDRAW message with the TreeView control while adding items to the control.

STATUS

Microsoft has confirmed this to be a bug in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: 4.00 95
KBCategory: kbprg kbbuglist
KBSubcategory: UsrCtl

BUG: Win32 SDK Ver. 3.5 Bug List for Win32 SDK and Win32 API

PSS ID Number: Q121907
Authored 23-Oct-1994 Last modified 03-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 3.5

SUMMARY

The following is a list of bugs in the Win32 SDK and Win32 API (version 3.5) that were known at the time of its release. The list is divided into four sections: Base, GDI, Networking, and User.

MORE INFORMATION

Base

- SetLastErrorEx() does not raise a RIP_INFO debug event.

- Overlapped I/O does not work on CD-ROM drives.

- CDECL should be defined to __cdecl in WINDEF.H.

- EVENTLOGRECORD strings are incorrectly given as type WCHAR in winnt.h. The online help correctly gives the strings as type TCHAR.

- SetComputerName(""); sets computer name to an empty string at reboot, which the logon service cannot handle. The Network applet does not allow this computer name. Your code should check for this case as well.

- Registry allows remote access to certain performance data, but not local access.

- RegCreateKeyEx() puts the class string in all created subkeys.

- Call FileTimeToLocalFileTime() with a structure containing 0, then pass the result to FileTimeToSystemTime(). FileTimeToSystemTime() returns a bad value in SystemTime.wHour.

- DLL does not receive DLL_PROCESS_DETACH if last thread in process calls ExitThread().

GDI

- Return values for certain GetBitmapBits() calls are not the same on Windows and Windows NT.

- Return values for certain StretchBlt() calls are not the same on

Windows and Windows NT.

- When a transparent window is present and full drag is enabled, the display is not refreshed correctly when a window is dragged.

- GetCharABCWidths() returns twice the width for character 0x92 that GetTextExtentPoint() returns using Arial font.

- Use a True Type font off the network, produce a few glyphs, and kill the network connection. You get an in page memory exception if you try to use the font with different (not cached) glyphs. Restore the network connection and try to use the font. The font was tagged as invalid, so all subsequent TextOut() calls fail.

- CreateDC() returns ERROR_MOD_NOT_FOUND instead of ERROR_ACCESS_DENIED for a printer with no access rights. The problem is that GDI does not check for ERROR_ACCESS_DENIED when the open printer fails; GDI directly tries to do a LoadLibrary() on the pszDevice.

- EndDoc() reports success when given an information context (IC).

- AbortDoc() reports success when given an information context (IC).

- glGet() doesn't generate GL_INVALID_OPERATION error code when it is called within glBegin()/glEnd().

Networking

- For MS-DOS clients, NetAccessGetInfo() returns code 53. (ERROR_BAD_NETPATH) for the local server.

- For MS-DOS clients, NetServerEnum2() with a bad domain should return error 2320, but it returns Windows NT error code 6118.

- For MS-DOS clients, NetShareGetInfo() level 2 fails with error 87.

- For MS-DOS clients, NetShareGetInfo() level 3 returns NERR_Success, but level 3 is not a valid level.

- NetWkstaGetInfo() returns a platform id of PLATFORM_ID_OS2 for a downlevel Windows for Workgroups machine.

- NetUserAdd() remoted from a 16-bit application to a server with a long name fails with error 59 (unexpected network error).

- NetServiceEnum() resumehandle and prefMaxLen fields are ignored.

- NetGetDCName() with domain name of "." or "?" returns error 2102 (NERR_NetNotStarted), not 2453 (NERR_DCNotFound).

- NetUseEnum() with a preferred buffer size less than 40 bytes returns ERROR_MORE_DATA, rather than NERR_BufTooSmall.

- NetServerEnum() with a zero length buffer returns ERROR_MORE_DATA,

rather than NERR_BufTooSmall.

- NetScheduleJobEnum() with a small preferred buffer size returns ERROR_MORE_DATA, rather than NERR_BufTooSmall.

- NetStatisticsGet() remoted to a downlevel OS/2 server does not work for the service "LanmanWorkstation".

- NetAccessEnum() with NULL base path causes an access violation.

- NetUserModalsGet() level 2 returns error 87 (ERROR_INVALID_PARAMETER).

- Network API doc errors: NetUserModals() supports a level 3 structure in Windows NT 3.5, NetGroupSetUsers() also supports level 1, NetAccessCheck() is in docs but not the header files, and the NetAuditClear()/NetAuditRead() service parameter should be marked as a reserved field that must be NULL.

- NetConfigGet() and NetConfigGetAll() return error 2146 when remoted to Windows NT. This API should only work from Windows NT to a downlevel machine. It should return ERROR_NOT_SUPPORTED or NERR_InvalidApi is remoted to Windows NT.

- NetWkstaUserSetInfo() level 0 returns error 124 (invalid level).

- NetReplImportDirUnlock() returns error 87 (NERR_Invalid_Parameter) if the directory is not locked.

- NetReplExportDirEnum() ignores suggested buffer size.

- NetUseEnum() level 2 remoted to an OS/2 machine returns error 50 (ERROR_NOT_SUPPORTED), not error 124 (ERROR_INVALID_LEVEL).

- NetConfigSet() returns success when remoted to Windows NT.

- NetWkstaUserSetInfo() called with an incorrect level returns error 87 (INVALID_PARAMETER), rather than error 124 (ERROR_INVALID_LEVEL).

- NetMessageNameEnum() with a zero length buffer returns NERR_Success, rather than NERR_BufTooSmall.

User

- DDESPY frees locked global objects when filtering out WM_DDE_INITIATE messages.

- GetTabbedTextExtent(), TabbedTextOut(), TextOut(), and DrawText() GP fault with bad string length values.

- Journal Recording causes improper icon title painting.

- With a WH_MSGFILTER installed, MSGF_NEXTWINDOW is not sent when the user Alt-Tabs to the next window.

- Active Screen Saver doesn't allow WH_JOURNALPLAYBACK hook.

- 122-key kbd: WSCtrl, Jump, Finish keys are VK_NONAME.

- Using of NumLock key for PF1 in terminal emulation programs does not work.

- Calling GetKeyState()/GetKeyboardState() from a thread that is not reading from an input queue causes a QEVENT_UPDATEKEYSTATE to be posted to the queue, but it is never removed. If the input queue is being read, the key state is one update event out of date.

- System hang during journalling if a hung application exists and many system events are posted to the hung application.

- Windows NT does not wait for all applications to return their response from WM_QUERYENDSESSION before starting to destroy an application. This can cause data loss in OLE.

- WM_GETMINMAXINFO not used when window has WS_CAPTION and WS_THICKFRAME.

- Edit control clips italic text.

- Hotkeys are not checked during journal playback.

- OLE 2.01 server creates a window and uses SetParent() on the window of the container. When the container is resized to a smaller window than the server, the container can no longer be resized with the mouse over the server window. This is a side effect of calling SetParent() on a window that does not have style WS_CHILD.

- GetSystemMetrics() does not check parameter to see if it is within range and returns garbage for out of range values.

- Windows applications with focus can be overlapped.

- In large font video mode, AdjustWindowRect() returns bad width if the window does not have a menu and a bad height if the window has a menu.

- Prototypes for SetSystemCursor() and LoadCursorFromFile() are missing from WINUSER.H.

Additional reference words: 3.50 buglist3.50
KBCategory: kbprg kbnetwork kbbuglist
KBSubcategory: BseMisc GdiMisc NtwkMisc UsrMisc

BUG: Win32 SDK Version 3.5 Bug List - OLE

PSS ID Number: Q122679
Authored 09-Nov-1994 Last modified 10-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 3.5

SUMMARY

The following lists the bugs in the Win32 SDK and OLE API that were known when the SDK was released.

MORE INFORMATION

- Packager will report that there is not enough memory when you try to package a file which is in use.

- VB can't activate Word object if Word is running in separate VDM.

- MFC 2.5 OLE 2.0 Automation Server fails initialization.

- If a client thread terminates without calling CoUnitialize(), its servers in the same process are not released.

- OLE 2.0 hides application aborts. The server does not receive the exception.

- CoCreateInstance() returns REGDB_E_CLASSNOTREG when the object create fails, instead of the return code from DllGetClassObject() from the inproc server DLL.

- OLE objects inherit only "system" environment variables, not "user" environment variables.

- Users can log on and start an ole app before the OLE service is autostarted.

- Passing CoMarshalInterface() a NULL pUnk (pointer to IUnknown) causes an access violation.

- CoGetClassObject() returns E_OUTOFMEMORY when DllGetClassObject() fails.

- If an OLE Server which dies after registering its class, the container will stall waiting for CoGetClassObject() to succeed.

- Class cache never shrinks - all classes are treated as InUse.

- Default handler causes RPC_E_FAULT exceptions during OnClose.

- ::SetColorScheme() does not validate lpLogPal.

- Messages for server ownerdraw menuitems on menu bar go to the container.

- The server is started if you "Insert" a "Link from File" choosing icon format.

- IsLinkUpToDate() returns S_FALSE after object creation. This can cause containers to run the server twice during creation of the object.

- Setting the link source with IOL::SetSourceMoniker() does not update the presentation cache, even though it does run the server app.

- AddRef() does not marshall count back properly for return value.

- BindToStorage() on non-existent file returns MK_E_INVALIDEXTEN.

- IOleCache::Cache() fails for ICON aspect unless metafile format is used.

- IOleCache2::DiscardCache() does not persist uncaches.

- If an enhanced metafile node exists and IOleCache::Cache is used to cache multiple NULL FORMATETCs, a node is added.

- IViewObject::GetColorSet() after flushing cache should return OLE_E_BLANK, but signals Win32 error 0x8007000e.

- OleDuplicateMedium() does not GlobalUnlock() source METAFILEPICT in the error case.

- Iconic aspect has incorrect colors when played in metafile.

- OLE Cache has a limit of 100 nodes, but accepts more and never returns from the call.

- GetColorSet() for WMF should return LOGPALLETE, which is the union of colors used in the contained CreatePalette() calls. GetColorSet() only returns the first colorset found in this case.

Additional reference words: 3.50
KBCategory: kbprg kbbuglist
KBSubcategory: LeTwoMisc

BUG: Win32 SDK Version 3.5 Bug List - WinDbg Debugger

PSS ID Number: Q122681
Authored 09-Nov-1994 Last modified 29-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 3.5

SUMMARY

The following is a list of bugs in the WinDbg debugger that were known at the time of the release of the Win32 SDK version 3.5.

MORE INFORMATION

- Enter rgbGlobal, s in watch window (note: rgbGlobal is defined as a character array). The variable is displayed as a character string and an array that can be expanded. When expanding the array, the value of each element is "CAN0026: Error: bad format string."

- REP and REPE are the same prefix codes. REPE is to be used for string comparisons and REP for all other instructions. WinDbg always uses REP.

- Type information defined in a DLL is not available when the current context is another DLL or an EXE.

- Breakpoint message classes do not match class list in SPY.

- C++ expresssion evaluator doesn't handle default function arguments. This is because the compiler does not include them in the debug information.

- Locals window updates on radix change.

- Evaluation of a function with breakpoints returns an incomplete result.

- The Memory Window can't be scrolled up before the starting address.

- Locals window collapses expanded structures on change of scope, such as stepping into a block (not a new function).

- The value of array members cannot be changed.

- The expression evaluator does not handle casting from a class to a primitive data type.

- Remove Last in Quickwatch only works once when multiple items are added to the watch list in a single quickwatch session.

- The return value type is not reported for ?<FuncName>.

- Function evaluation reports "Error: function requires implicit conversion" for a function taking a structure (not a pointer to a structure).

- Watch window shift-key selection is not consistent: sometimes all characters from the beginning of the expression to the caret position are selected, sometimes 2 characters are selected.

- User DLLs dialog silently discards edits after picking a DLL and changing the radio button from suppress to load.

- Information windows don't maintain color after structure expansion.

- WinDbg disassembles F2 66 F0 F0 AF as "repne lock lock sca" not "repne lock lock scasw".

- Breakpoints may not work correctly in multithreaded apps in areas not protected by critical sections.

- Choosing Stop Debugging and Restart causes memory leak (100K per iteration).

- Combo box in dialog for browsing symbol files is too narrow to show the *.dbg.

- Debug.Watch does not set default watch expression to the selection made in the source window.

- If you set a conditional breakpoint, you step over it with an F10, and the condition is not currently satisfied, the program will run to completion, rather than stepping.

- Long expression (?arg00+arg01+...+arg31) causes debuggee to run to termination.

- ?<function returning near pointer> displays segment.

- Windbg hangs if exited during aedebug start.

- A vararg function evaluation fails on Mips and Alpha.

- Private members may not be evaluatable.

- First Command Window prompt after connecting to target machine for kernel debugging is ">", not "KDx86>", "KDMIPS>", or "KDALPHA>".

- Context expression evaluation of item up the callstack cannot be evaluated and causes CXX0036: Error: bad context {...} specification.

- Alpha: Disassembly of RS, RC, RPCC, FETCH, and FETCH_M instructions displays no operand.

- Help file says "u" command is for unfreezing a thread. The "u" command is for unassemble; it is the "z" command that is used for unfreezing a thread.

- Automatic forward searching not done by breakpoint dialog. Otherwise, when setting a breakpoint on a line that does not contain executable code, the breakpoint is set on the next executable line.

- Automatic forward searching not done when modules are loaded. Otherwise, when setting a breakpoint on a line that does not contain executable code, the breakpoint is set on the next executable line.

- OK button not always active on Set Process dialog.

- Alpha: Large enumerated value not displayed correctly (16-bits instead of 32-bits).

- The Delete button in User DLLs dialog is always active.

- ?Spinlock::Spinlock should display the prototype for the function, but it causes CXX0046: Error: argument list required for member function.

- Flat callstack displayed debugging 16-bit Windows-based application.

- File menu Save_All is not enabled consistently on all platforms.

- Page up/down goes farther than scroll thumb in the Memory Window.

- Page up/down doesn't move scroll thumb in Memory Window.

- Disassembler option "Display Symbols" ignored on Alpha.

- Ppcodes always displayed in lower case in MIPS disasembly, even if "Uppercase symbols and opcodes" is checked.

- Create several workspaces for a single program, choose Delete from the Program menu, and select several of the workspaces. WinDbg locks up when you select OK.

- Deleting the last debugger DLL causes an access violation.

- Bad caret movement when editing Memory Window with ASCII format.

- Calls window not updated if the current thread is changed with the Set Thread dialog. The Calls window is updated if the Command window is used to set the current thread.

- Thread-specific translations of segment registers is not done. The segment register is translated using thread 0's descriptor table.

- When stepping over a function which contains a breakpoint, execution halts, but there is no message indicating that a breakpoint was hit.

- Value of "this" pointer is incorrect in a virtual function in a derived class.

- Based pointers in flat segments are displayed as a 16-bit value, not a 32-bit value. In addition, nothing happens when you click the expansion

button.

- WINDBG won't set a breakpoint on code placed in memory and then executed.

- Windbg does not know about all exceptions that can occur while debugging 16-bit code.

- Alpha: CVTxx instructions disassembled with 3 operands, instead of only 2 operands. The first operand is wrong, the second operand would be the correct first operand, and the third operand would be the correct second operand.

- !help <str> reports that there is no help available.

- Set a breakpoint on a function call which spans multiple source lines, but don't set the breakpoint on the last line. Save the information and leave the debugger. When you restart WinDbg with the saved information, WinDbg cannot resolve the breakpoint.

- Alpha: Cannot step through call through a function pointer.

- Commands sxeld and sxdld cause the debugger to stop when a DLL is loaded.

- If there are no symbols loaded, double-clicking a symbol in the call stack produces a disassembly window with a starting address of 0.

- The following context operators cause "CXX0036: Error: bad context {...} specification":

```
?{,functest.c,functest.exe}count
?{,functest.c,}count
```

  The following context operators cause "CXX0017: Error: symbol not found":

```
?{,,functest.exe}count
?{,,}count
```

- When the current instruction is "cmp dword ptr [esp+18],01", the register window shows a calculation based on [esp], rather than [esp+18].

- WinDbg displays only the first letter of a 'const WCHAR *const' variable. Casting the variable to a WCHAR * in the Watch window works around the problem.

- Run windbg -g cmd.exe and invoke a batch file that repeatedly invokes another command; WinDbg will leak memory.

- x86: f2a6 is disassembled as "repnee cmpsb", not "repne cmpsb",
  f2a7 is disassembled as "repnee cmpsb", not "repne cmpsd",
  f2ae is disassembled as "repnee scasb", not "repne scasb",
  f2af is disassembled as "repnee scasd", not "repne scasd",
  f0a6 is disassembled as "locke cmpsb", not "lock cmpsb",

f0af is disassembled as "locke scasd", not "lock scasd",
f32ea6 is disassembled as "rep cmpsb", not "repe cmpsb",
f326a7 is disassembled as "rep cmpsd", not "repe cmpsd",
f32ea7 is disassembled as "rep cmpsd", not "repe cmpsd",
f366a7 is disassembled as "rep cmpsw", not "repe cmpsw",
f36665a7 is disassembled as "rep cmpsw", not "repe cmpsw",
f326ae is disassembled as "rep scasb", not "repe scasb",
f365af is disassembled as "rep scasd", not "repe scasd",
f33eaf is disassembled as "rep scasd", not "repe scasd",
f3f0af is disassembled as "rep locke scasw", not "repe lock scasw",
f366af is disassembled as "rep scasw", not "repe scasw",
f36636af is disassembled as "rep scasw", not "repe scasw".

- dc doesn't accept the '&' prefix for an address specifier.

- CXX0004: Error: syntax error on reference to float array. For example, the error is produced by "g .115;?Pf[8], where Pf is declared float Pf[11].

- If you have a DLL built with multiple files with the same name (that live in different source directories), you cannot set a break point in 2nd file with same name.

- Error "CXX0034: Error: types incompatable with operator" accessing members, member functions, and overloaded operators of base classes and virtual base classes or a derived class.

- Alpha: WinDbg doesn't display floating part of a float constant.

- Crash dumps fail because of bad symbol lookup. This breaks !process when kernel debugging as well.

Additional reference words: 3.50
KBCategory: kbtool kbbuglist
KBSubcategory: TlsWindbg

BUG: Win32 Ver 3.5 SDK Bug List at Release - Subsystems & WOW

PSS ID Number: Q122048
Authored 25-Oct-1994 Last modified 29-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 3.5

SUMMARY

The following is a list of the bugs in the Win32 version 3.5 SDK that were known when Win32 version 3.5 was released. The list is divided into three sections: OS/2 Subsystem, POSIX Subsystem, and Windows on Win32 (WOW).

MORE INFORMATION

OS/2 Subsystem

- Extended characters in editors don't work with right-hand Alt key.

- DosRmDir() returns ERROR_PATH_NOT_FOUND, not ERROR_FILE_NOT_FOUND, if passed the name of a directory that does not exist.

- CTRL-X doesn't show with OS/2 Epsilon, although it is received by the application.

- DosSelectDisk() returns error if disk not ready, but not under OS/2.

- DosFindFirst() doesn't find config.sys in the registry.

- Ctrl-C doesn't centre text in Word 5.5 for OS/2.

- Ctrl-C will not cancel Fortran 5.1 setup.

- DosQApptype() returns ERROR_INVALID_EXE_SIGNATURE (191) if pszFileName does not use the correct .EXE format, but OS/2 returns ERROR_BAD_EXE_FORMAT (193).

- DosMove() returns error 80 if the target file exists, but OS/2 returns error 5.

- DosSetFileMode() returns error 0x02 if the file does not exist, but OS/2 returns error 0xCE.

- OS/2 subsystem doesn't load REXXINIT.DLL on startup, as OS/2 does, so OS/2 applications using REXX will fail on the first attempt.

- Flags3 field not preserved between DosDevIOCtl/ASYNC_SETDCB and DosDevIOCtl/ASYNC_GETDCB. Flags3 is always returned with a value of 3.

- Many vio functions return the wrong error code.

- Epsilon not jumping back after matching braces.

- NetHandleGetInfo() returns error 87 at level 2 or level 3.

- DosCreateThread() corrupts 11 words of temporary (under OS/2, only 2 words are corrupted).

- UUPC OS/2 1.x application tries to open and write to the com port. CTS and DTR come up, but RTS never does.

- SQL 4.2 setup fails to configure sort order.

- DosKillProcess() of ancestor with DKP_PROCESSTREE succeeds, but OS/2 returns ERROR_NOT_DESCENDANT.

- DosCWait() with DCWA_PROCESSTREE returns when the process terminates, not when the tree terminates.

- OS/2 CMD.EXE hangs after running Epsilon or PWB.

- Signals not held while resizing data segments with DosReallocSeg().

POSIX Subsystem

- printf() doesn't check to see if writing fails on the first character; it tries to write the second character instead of returning an error.

Windows on Win32 (WOW)

- GlobalSize() of handle returned by GetMetaFile() may not match.

- Invalid TEMP environment variable causes problems, whereas Windows 3.1 will use the root if the TEMP environment variable is invalid.

- GetExitCodeProcess() does not return exit codes for WOW applications.

- Floating point exceptions are not reported to the debugger.

- Segment Load Failure in KRNL386 after using WINDISK.EXE.

- Locking at negative offset fails on WOW with EACCES, but works under Windows.

- WIN16 emulator incorrectly handles FBSTP instruction on MIPS, Alpha, and x86 with no math coprocessor.

- WOW passes wrong size buffer to spooler when 16-bit application passes wrong size structure to WOW.

- WOW uses Windows 3.0 devmode structure, not 3.1 devmode structure, which has two extra fields.

- Fast Alt+Tab between WOW apps selects the File menu from one of the applications.

- WM_MENUSELECT wParam and lParam are lost in PostMessage().

- EndDeferWindowPos() returns 0 for success, but returns nonzero for success under Windows.

Additional reference words: 3.50 buglist3.50
KBCategory: kbprg kbbuglist
KBSubcategory: SubSys

BUG: Win32s 1.2 Bug List (at the Time of its Release)

PSS ID Number: Q121906
Authored 23-Oct-1994 Last modified 15-Jun-1995

The information in this article applies to:

- Microsoft Win32s version 1.2

The following is a list of the known bugs in Win32s version 1.2 at the time of its release.

- EM_GETWORDBREAKPROC return code is incorrect.

- Int 3 cannot be trapped via Structured Exception Handling (SEH) on Win32s.

- GlobalAlloc(GMEM_FIXED) from 32-bit .EXE locks memory pages. It is more efficient to use GlobalAlloc(GMEM_MOVEABLE) and call GlobalFix() if necessary.

- WinHlp32.exe will not run.

- C Run-time functions getdcwd()/getcwd() do not work.

- PlayMetaFileRecord()/EnumMetaFile() contain incorrect lpHTable.

- Print setup common dialog does not work as expected (the dialog does not appear in some situations).

- Size of memory mapped files is rounded to a whole number of pages, meaning that the size is a multiple of 4096 bytes.

- Functions chdrive() and SetCurrentDirectory() fail on PCNFS network drives.

- Calling CreateWindowEx() with a 32-bit menu handle causes int 3.

- GetExitCodeProcess() does not return exit codes for Win16 applications.

- FindFirstFile()/FindNextFile() return local time, not Universal Coordinated Time (UTC) time. This behavior matches Windows 3.1, but not Windows NT.

- Memory passed to Netbios() must be allocated with GlobalAlloc().

- biSizeImage field of BITMAPINFOHEADER is zero

- CreateFile() on certain invalid long filenames closes Windows.

- Only the first CBT hook gets messages.

- GlobalUnlock() sets an error of ERROR_INVALID_PARAMETER.

- CreateFile() with share options of 0 does not lock file.

- lstrcmp()/lstrcmpi() do not use the collate table correctly.

- FreeLibrary() in DLL_PROCESS_DETACH crashes system.

- LockResource() does not return NULL if hResource is invalid.

- sndPlaySound() may cause a crash or may work poorly.

- GetVolumeInformation() fails for Universal Naming Convention (UNC) root path.

- Stubbed API GetFileAttributesW() does not return -1 on error.

- Code page CP_MACCP not supported.

- Invalid LCIDs are not recognized.

- CreateFile() fails to open an existing file in share mode.

- CreateDirectory()/RemoveDirectory() handle errors differently on Windows NT and Win32s.

- SetCurrentDirctory() returns different error codes than on Windows NT.

- FindText() leaks memory.

- FindText() may cause GP fault.

- Not all 32-bit DLLs have correct version numbers.

- WINMM16.DLL has no version information.

- RegEnumValue() and other Registry functions return ERROR_SUCCESS even though they are not implemented. Win32s implements only the registry functions supported by Windows.

- COMPOBJ.DLL calls FreeLibrary() on w32sys.dll, leaving the FP exception vector invalid. Causes crash, often out to MS-DOS. A binary patch for COMPBOJ.DLL is available in the Microsoft OLE 2.02 appnote WW1116.

- CreatePolyPolygonRgn() is not closing the polygons.

- SetDIBits() with DIB_PAL_INDICES is not supported (this behavior matches Windows 3.1, but not Windows NT).

- Progman gets restored when debugger app exits.

- Cannot do ReadProcessMemory (RPM) on memory that has a hardware break point set on it.

- Pointer to common dialog structures (lParam) becomes invalid.

- ResumeThread() while debugging writes to debuggee stack.

- Fault in initialization if app has more than 128K bytes of local data. This is because the application stack is limited to 128K.

- Using CVW debugger, exception will terminate the app being debugged.

- RealizePalette() error on Windows NT is -1, but is not defined under Win32s.

- Using StartDoc() does not produce document from printer.

- Can't open file using full path with different drive in common dialog.

- GetFileInformationByHandle() doesn't return correct file attribute.

- GlobalReAlloc(x,y,GMEM_MOVEABLE) returns wrong handle type.

- Thread Local Storage (TLS) data not initialized to zero in TlsAlloc().

- spawnl() does not pass parameters to an MS-DOS application.

- Incorrect context at EXIT_PROCESS_DEBUG_EVENT.

- Win32s doesn't support language files other than default (l_intl.nls).

- SetLocaleInfoW()/SetLocaleInfoA() not implemented.

- GetScrollPos() fails if scroll pos is 0.

- SetScrollPos() fails if last scroll pos is 0.

- FreeLibrary() may crash when using universal thunks.

- Win32s does not support forwarded exports.

- GetCurrentDirectory() returns wrong directory after calling GetOpenFileName(). The workaround is to call SetCurrentDirectory(".") right after returning from the call to GetOpenFileName().

- Changing system locale in Win32s will not have an effect until Win32s is loaded again, unlike on Windows NT.

- Module management APIs missing ANSI to OEM translation.

- LoadString() leaks memory if the string is a null string.

Additional reference words: 1.20
KBCategory: kbprg kbbuglist
KBSubcategory: W32s

BUG: Win32s 1.25a Bug List

PSS ID Number: Q130138
Authored 11-May-1995 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Win32s, version 1.25a

The following is a list of the known bugs in Win32s version 1.25 at the time of its release.

- Incorrect context at EXIT_PROCESS_DEBUG_EVENT.

- Progman gets restored when debugger app exits.

- Using StartDoc() does not produce document from printer.

- EM_GETWORDBREAKPROC return code is incorrect.

- Int 3 cannot be trapped via Structured Exception Handling (SEH) on Win32s.

- Win32s does not open all files in RAW mode, as Windows NT does.

- Cannot do ReadProcessMemory (RPM) on memory that has a hardware breakpoint set on it.

- C run-time functions getdcwd()/getcwd() do not work.

- GetFullPathName() returns the root directory for any drive that is not the current drive.

- PlayMetaFileRecord()/EnumMetaFile() contains incorrect lpHTable.

- Size of memory mapped files is rounded to a whole number of pages, meaning that the size is a multiple of 4096 bytes.

- Functions chdrive() and SetCurrentDirectory() fail on PCNFS network drives.

- GetExitCodeProcess() does not return exit codes for 16-bit Windows-based applications.

- Memory passed to Netbios() must be allocated with GlobalAlloc().

- biSizeImage field of BITMAPINFOHEADER is zero.

- CreateFile() on certain invalid long filesnames closes Windows.

- Only the first CBT hook gets messages.

- Most registry functions return the Windows 3.1 return codes, not the Windows NT return codes.

- GlobalReAlloc(x,y,GMEM_MOVEABLE) returns wrong handle type.

- GetVolumeInformation() fails for Universal Naming Convention (UNC) root path.

- ResumeThread while debugging writes to debuggee stack.

- GetShortPathName() doesn't fail with a bad path, as it does on Windows NT.

- CreateDirectory()/RemoveDirectory() handle errors differently than on Windows NT.

- SetCurrentDirctory() returns different error codes than on Windows NT.

- FindText() leaks memory.

- Win32s doesn't support language files other than default (l_intl.nls).

- spawnl does not pass parameters to an MS-DOS-based application.

- Win32s does not support forwarded exports.

- GetDlgItemInt() only translates numbers <= 32767 (a 16-bit integer).

- Changing system locale in Win32s will not have an effect until Win32s is loaded again, unlike on Windows NT.

- Module Management APIs missing ANSI to OEM translation.

- When WS_TABSTOP is passed to CreateWindow(), this forces a WS_MAXIMIZEBOX.

- Stubbed API FindFirstFileW() does not return -1 to indicate failure.

- SearchPath() and OpenFile() don't work properly with OEM chars in the filename.

- GetSystemInfo() doesn't set correct ProcessorType for the Pentium.

- FormatMessage() doesn't set last error.

- FormatMessage() fails with LANG_NEUTRAL | SUBLANG_DEFAULT, but works with LANG_ENGLISH | SUBLANG_ENGLISH_US.

- After calling CreateFile() on a write-protected floppy GetLastError() returns 2, instead of 19, as it should.

- VirtualProtect() with anything other than PAGE_NOACCESS, PAGE_READ, OR PAGE_READWRITE yields unpredictable page protections.

- COMPAREITEMSTRUCT, DELETEITEMSTRUCT, DRAWITEMSTRUCT, AND MEASUREITEMSTRUCT incorrectly sign-extend fields.

- GetWindowTextLength() & GetWindowText() incorrectly sign-extend the return value.

- MoveFile() fails on Windows for Workgroups when the source is remote and the destination is local.

Additional reference words: 1.25 1.25a
KBCategory: kbprg kbbuglist
KBSubcategory: W32s

BUG: WNetGetUniversalName Fails Under Windows 95

PSS ID Number: Q131416
Authored 11-Jun-1995 Last modified 12-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SYMPTOMS

The WNetGetUniversalName function takes a drive-based path for a network resource and obtains a data structure that contains a more universal form of the name. This function always fails with error 1200 when called from a 32-bit application running under Windows 95.

WORKAROUND

The functionality provided by WNetGetUniversalName can be implemented using the Win32 network enumeration functions WNetOpenEnum and WNetEnumResource. Here is an example of how to use these functions to implement similiar functionality:

```
#include <windows.h>
#include <stdio.h>

// Function Name:  GetUniversalName
//
// Parameters:     szUniv  - contains the UNC equivalent of szDrive
//                           upon completion
//
//                 szDrive - contains a drive based path
//
// Return value:   TRUE if successful, otherwise FALSE
//
// Comments:       This function assumes that szDrive contains a
//                 valid drive based path.
//
//                 For simplicity, this code assumes szUniv points
//                 to a buffer large enough to accomodate the UNC
//                 equivalent of szDrive.

BOOL GetUniversalName( char szUniv[], char szDrive[] )
{
   // get the local drive letter
   char chLocal = toupper( szDrive[0] );

   // cursory validation
   if ( chLocal < 'A' || chLocal > 'Z' )
      return FALSE;

   if ( szDrive[1] != ':' || szDrive[2] != '\\' )
```

```
      return FALSE;

   HANDLE hEnum;
   DWORD dwResult = WNetOpenEnum( RESOURCE_CONNECTED, RESOURCETYPE_DISK,
                                  0, NULL, &hEnum );

   if ( dwResult != NO_ERROR )
      return FALSE;

   // request all available entries
   const int    c_cEntries   = 0xFFFFFFFF;
   // start with a reasonable buffer size
   DWORD        cbBuffer     = 50 * sizeof( NETRESOURCE );
   NETRESOURCE *pNetResource = (NETRESOURCE*) malloc( cbBuffer );

   BOOL fResult = FALSE;

   while ( TRUE )
   {
      DWORD dwSize   = cbBuffer,
            cEntries = c_cEntries;

      dwResult = WNetEnumResource( hEnum, &cEntries, pNetResource,
                                   &dwSize );

      if ( dwResult == ERROR_MORE_DATA )
      {
         // the buffer was too small, enlarge
         cbBuffer = dwSize;
         pNetResource = (NETRESOURCE*) realloc( pNetResource, cbBuffer );
         continue;
      }

      if ( dwResult != NO_ERROR )
         goto done;

      // search for the specified drive letter
      for ( int i = 0; i < (int) cEntries; i++ )
         if ( pNetResource[i].lpLocalName &&
              chLocal == toupper(pNetResource[i].lpLocalName[0]) )
         {
            // match
            fResult = TRUE;

            // build a UNC name
            strcpy( szUniv, pNetResource[i].lpRemoteName );
            strcat( szUniv, szDrive + 2 );
            _strupr( szUniv );
            goto done;
         }
   }

done:
   // cleanup
   WNetCloseEnum( hEnum );
```

```
   free( pNetResource );

   return fResult;

}
```

STATUS

Microsoft has confirmed this to be a bug in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: 4.00 Win95
KBCategory: kbnetwork kbbuglist kbcode
KBSubcategory: NtwkWinnet

Button and Static Control Styles Are Not Inclusive

PSS ID Number: Q74297
Authored 15-Jul-1991 Last modified 16-May-1995

-
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

-

The class-specific window styles for button controls (BS_*) are mutually exclusive, as are the class-specific window styles for static controls (SS_*). In other words, they cannot be OR'd together as can most styles for edit controls, list boxes, and combo boxes.

For example, the following style is invalid:

```
BS_OWNERDRAW | BS_AUTORADIOBUTTON
```

A button control is either owner-draw or it is not. In the same manner, the following style is also invalid:

```
SS_LEFT | SS_GRAYFRAME
```

WINDOWS.H defines button and static styles sequentially (1, 2, 3...), instead of as individual bits (1, 2, 4...) as other control's styles are defined. If sequential styles are OR'd together, the resulting style may be completely different from that intended.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Calculating String Length in Registry

PSS ID Number: Q94920
Authored 25-Jan-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

When writing a string to the registry, you must specify the length of the string, including the terminating null character (\0). A common error is to use strlen() to determine the length of the string, but to forget that strlen() returns only the number of characters in the string, not including the null terminator.

Therefore, the length of the string should be calculated as:

```
strlen( string ) + 1
```

Note that a REG_MULTI_SZ string, which contains multiple null-terminated strings, ends with two (2) null characters, which must be factored into the length of the string. For example, a REG_MULTI_SZ string might resemble the following in memory:

```
string1\0string2\0string3\0laststring\0\0
```

When calculating the length of a REG_MULTI_SZ string, add the length of each of the component strings, as above, and add one for the final terminating null.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseMisc

Calculating Text Extents of Bold and Italic Text

PSS ID Number: Q74298
Authored 15-Jul-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

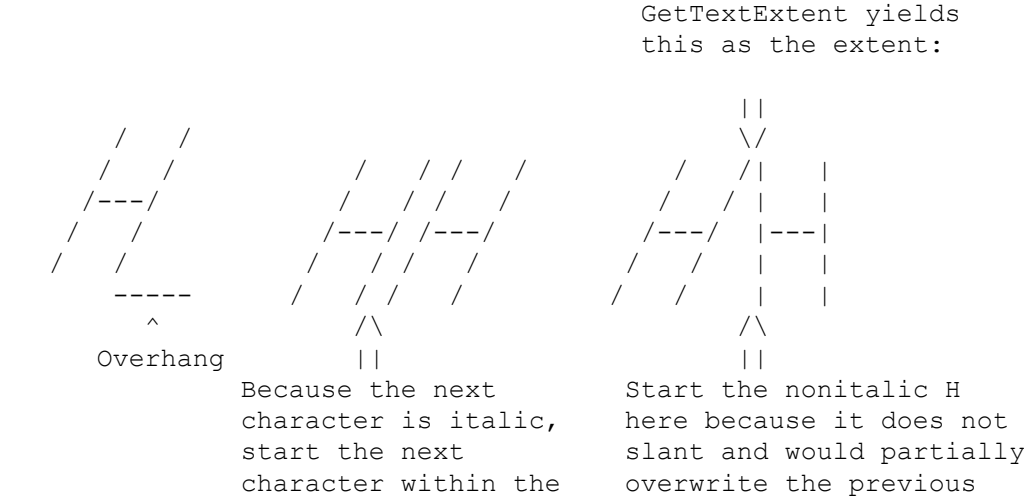
GetTextExtent() can be used to calculate the extent of a string. The value returned may need to be adjusted, depending upon the style of the font. When an italic or bold font is requested and none are available, the graphics device interface (GDI) may simulate those styles using an existing raster or vector font.

MORE INFORMATION

GDI-simulated bold and italic fonts both include overhangs. The overhang is specified in a TEXTMETRIC structure obtained by calling the GetTextMetrics function. The proper method for calculating the extent of a line of italic or bold text is shown below:

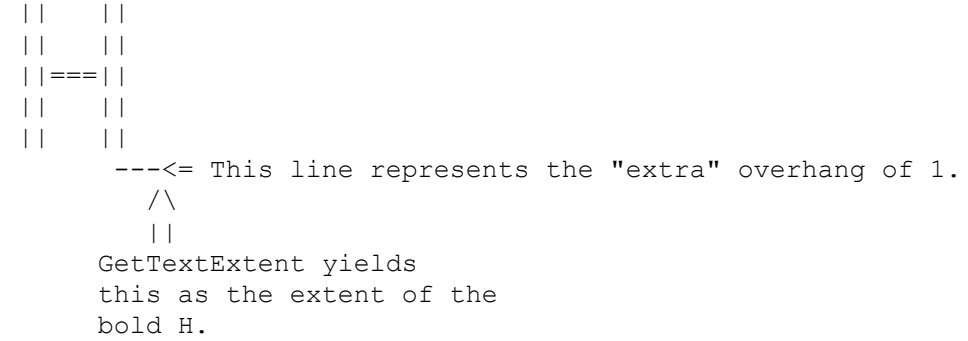
```
dwExtent = GetTextExtent(hDC, lpString, nCount);
GetTextMetrics(hDC, &tm);
xExtent = LOWORD(dwExtent) - tm.tmOverhang;
```

Listed below are examples of italic text alignment. If the next character is not italic, the overhang should not be subtracted from the extent returned from the GetTextExtent function. The overhang needs to be subtracted only when the next character has the same style.



```
               overhang of the         italic character.
               current character
```

The overhang for bold characters synthesized by GDI is generally 1 because GDI synthesizes bold fonts by outputting the text twice, offsetting the second output by one pixel, effectively increasing the width of each character by one pixel. Calculating the extent of the bold text is similar to the method for italic text. The GetTextExtent function always returns the extent of the text plus 1 for bold text. Thus by subtracting the tmOverhang(1), the proper extent is achieved.



Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiFnt

Calculating The Logical Height and Point Size of a Font

PSS ID Number: Q74299
Authored 15-Jul-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

To create a font in the Microsoft Windows graphical environment given only the required point size, an application must calculate the logical height of the font because the CreateFont() and CreateFontIndirect() functions use logical units to specify height.

To describe a font to the user, an application can calculate a font's point size, given its height. This article provides the formulas required to perform these calculations for the MM_TEXT mapping mode. You will have to derive a new equation to calculate the font size in another mapping mode.

MORE INFORMATION

To calculate the logical height, use the following formula:

$$\text{height} = \text{Internal Leading} + \frac{\text{Point Size} * \text{LOGPIXELSY}}{72}$$

LOGPIXELSY is the number of pixels contained in a logical inch on the device. This value is obtained by calling the GetDeviceCaps() function with the LOGPIXELSY index. The value 72 is significant because one inch contains 72 points.

The problem with this calculation is that there is no method to determine the internal leading for the font because it has not yet been created. To work around this difficulty, use the following variation of the formula:

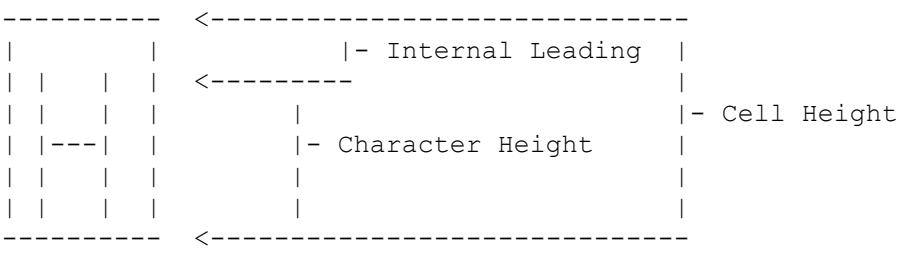
$$\text{height} = \frac{-(\text{Point Size} * \text{LOGPIXELSY})}{72}$$

This formula may also be written as follows:

```
plf->lfHeight = -MulDiv (nPtSize, GetDeviceCaps (hdc, LOGPIXELSY), 72);
```

When an application calls the CreateFont() or CreateFontIndirect() functions and specifies a negative value for the height parameter, the font mapper provides the closest match for the character height rather than the cell height. The difference between the cell height and the character

height is the internal leading, as demonstrated by the following diagram:



The following formula computes the point size of a font:

$$\text{Point Size} = \frac{(\text{Height} - \text{Internal Leading}) * 72}{\text{LOGPIXELSY}}$$

The Height and Internal Leading values are obtained from the TEXTMETRIC data structure. The LOGPIXELSY value is obtained from the GetDeviceCaps function as outlined above.

Round the calculated point size to the nearest integer. The Windows MulDiv() function rounds its result and is an excellent choice to perform the above calculation.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiFnt

Calculating the Point Size of a Font

PSS ID Number: Q74300
Authored 15-Jul-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The generic formula listed below can be used to compute the point size of a font in the MM_TEXT mapping mode. Any other mapping mode will require a different equation, because the Height will be in a different unit.

MORE INFORMATION

$$\text{Point Size} = \frac{(\text{Height} - \text{Internal Leading}) * 72}{\text{LOGPIXELSY}}$$

Height - Cell height obtained from the TEXTMETRIC structure.

Internal Leading - Internal leading obtained from TEXTMETRIC structure.

72 - One point is 1/72 of an inch.

LOGPIXELSY - Number of pixels contained in a logical inch on the device. This value can be obtained by calling the GetDeviceCaps() function and specifying the LOGPIXELSY index.

The value returned from this calculation should be rounded to the nearest integer. The Windows MulDiv() function rounds its result and is an excellent choice for performing the above calculation.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiFnt

Calculating the TrueType Checksum

PSS ID Number: Q102354
Authored 03-Aug-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

To calculate a TrueType checksum:

1. Sum all the ULONGS in the .TTF file, except the checkSumAdjust field (which contains the calculated checksum). Note that TrueType files are big-endian, while Windows and Windows NT are little-endian, so the bytes must be swapped before they are summed.

2. Subtract the result from the magic number 0xb1b0afba.

MORE INFORMATION

Example
---

1. Open the SYMBOL.TTF distributed with Windows NT. It is 64492 bytes long.

2. Step through the 16123 ULONGS, summing each one, except for the checkSumAdjust field for the file (which in this case is 0xa7a81151).

3. Subtract the result from 0xb1b0afba. The result is 0xa7a81151.

The TrueType font file specification is available from several sources, including the Microsoft Software Library (query on the word TTSPEC1).

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiFnt

Call the Windows Help Search Dialog Box from Application

PSS ID Number: Q86268
Authored 01-Jul-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

In the Microsoft Windows environment, an application can invoke the Search dialog box of the Windows Help application independent of the main help window. For example, many applications have an item like "Search for Help on" in their Help menus.

An application can invoke the Search dialog box via the WinHelp function by specifying HELP_PARTIALKEY as the value for the fuCommand parameter and by specifying a pointer to an empty string for the dwData parameter. The following code demonstrates how to call the Windows Help Search dialog box from an application:

```
LPSTR lpszDummy,
      lpszHelpFile;

// Allocate memory for strings
lpszDummy = malloc(5);
lpszHelpFile = malloc(MAX_PATH);

// Initialize an empty string
lstrcpy(lpszDummy, "");

// Initialize the help filename
lstrcpy(lpszHelpFile, "c:\\windows\\myhelp.hlp");

// Call WinHelp function
WinHelp(hWnd, lpszHelpFile, HELP_PARTIALKEY, (DWORD)lpszDummy);
```

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDlgs

Calling 16-bit Code from Win32-based Apps in Windows 95

PSS ID Number: Q125715
Authored 02-Feb-1995 Last modified 01-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

As a developer, you may need to access the functionality provided by a 16-bit DLL from your Win32-based applications. This is true particularly when you do not have the source code for the DLL so that you can port it to Win32. This article discusses the mechanism by which Win32-based DLLs can call Windows-based DLLs. The mechanism is called a thunk and the method implemented under Windows 95 is called a flat thunk.

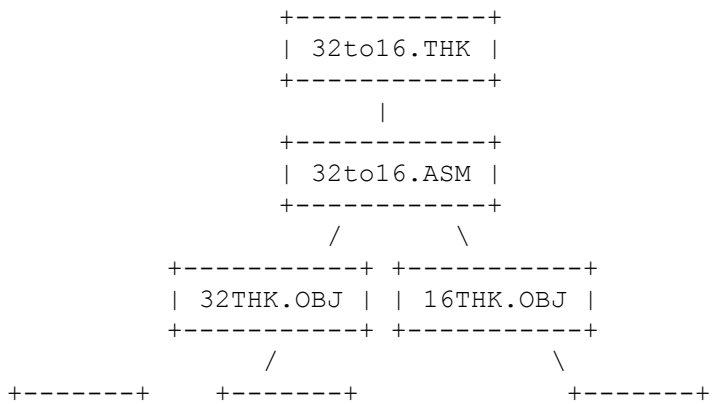
The three major steps in writing the thunk code are:

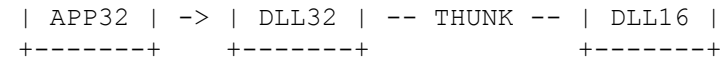
1. Creating the Thunk Script
2. Building the Win32-based DLL
3. Building the Windows-based DLL

MORE INFORMATION

The recommended way to design a thunk call is to isolate all thunk specific code in DLLs (a 16-bit DLL and a 32-bit DLL, to provide both sides of the thunk). That way, you can install certain DLLs on one platform and replace them on another platform, for portability.

Designing a new flat thunk involves creating a thunk script (.THK file). This script is compiled with the Thunk Compiler into an assembly file. This file is assembled using two different flags: -DIS_32 and -DIS_16. This allows you to create both the 16-bit and 32-bit object modules. These object modules are linked in the Windows-based and Win32-based DLLs, respectively. The following diagram summarizes the files involves in building the DLLs.





Creating the Thunk Script

You need to create a script that will be used by the thunk compiler to create a thunk. A thunk script contains the function prototype and a specification for the input and output values. You need to include the following statement to create a 32-bit to 16-bit thunk call:

```
enablemapdirect3216 = true
```

The following is an example of a simple thunk script for a function that has no input and output:

```
enablemapdirect3216 = true

void MyThunk16()
{
}
```

The following is an example of script that takes two parameters and returns a value. The second parameter is an output parameter and contains a pointer that is passed back to the Win32-based DLL.

```
enablemapdirect3216 = true

typedef int   BOOL;
typedef char *LPSTR;

BOOL MyThunk32(LPSTR lpstrInput, LPSTR lpstrOutput)
{
   lpstrOutput = output;
}
```

The statement "lpstrOutput = output" tells the script compiler that the 16-bit code will return an address that needs to be converted from a selector:offset pointer to a flat memory pointer.

The following thunk script uses more complex parameter types, such as structures. This example also shows how to specify input and output parameters.

```
enablemapdirect3216 = true

typedef int BOOL;
typedef unsigned int UINT;
typedef char *LPSTR;

typedef struct tagPOINT {
   INT  x;
   INT  y;
} POINT;
```

```
typedef POINT *LPPOINT;

typedef struct tagCIRCLE {
   POINT center;
   INT   radius;
} CIRCLE;
typedef CIRCLE *LPCIRCLE

void MyThunk32( LPCIRCLE lpCircleInOut)
{
   lpCircleInOut = InOut;
}
```

The statement "lpCircleInOut = InOut" tells the script compiler that this pointer is going to be used for input and output. This means that conversion from a flat memory pointer to a 16-bit selector:offset and vice-versa needs to be accomplished.

The thunk compiler usage is as follows:

```
thunk.exe /options <inputfile> -o <outputfile>
```

The following command line shows how to create a 16-bit thunk code.

```
thunk -t thk 32to16.thk -o 32to16.asm
```

The "-t thk" option tells the thunk compiler to prefix the thunk functions in the assembly language file with "thk_." This will create an assembly language file.

Building the Win32-based DLL (DLL32)

1. In the DllEntryPoint function (DllMain if you're using the Microsoft C Run-time libraries) in your Win32-based DLL, you must make a call to the imported function thk_ThunkConnect32, as shown here:

```
BOOL WINAPI DllMain(HINSTANCE hDLLInst,
                    DWORD  fdwReason,
                    LPVOID lpvReserved)
{
   if (!thk_ThunkConnect32("DLL16.DLL",
                           "DLL32.DLL",
                           hDLLInst,
                           fdwReason))
   {
      return FALSE;
   }
   switch (fdwReason)
   {
      case DLL_PROCESS_ATTACH:
         break;

      case DLL_PROCESS_DETACH:
         break;
```

```
            case DLL_THREAD_ATTACH:
               break;

            case DLL_THREAD_DETACH:
               break;
         }
         return TRUE;
      }
```

2. Include the following lines in the EXPORT section of the module definition (DEF) file for DLL32.

```
   thk_ThunkData32
```

3. Export the function that you are thunking to.

4. Assemble the assembly language file produced by the thunk compiler as a 32-bit object module. The following line shows an example:

```
   ml /DIS_32 /c /W3 /nologo /coff /Fo thk32.obj 32to16.asm
```

5. Link this object module as part of the Win32-based DLL (DLL32).

Building the Windows-based DLL (DLL16)

1. The Windows-based DLL must export a function named "DllEntryPoint". This function must make a call to an imported function thk__ThunkConnect16.

```
   BOOL FAR PASCAL __export DllEntryPoint(DWORD dwReason,
                                          WORD  hInst,
                                          WORD  wDS,
                                          WORD  wHeapSize,
                                          DWORD dwReserved1,
                                          WORD  wReserved 2)
   {
      if (!thk_ThunkConnect16("DLL16.DLL",
                              "DLL32.DLL",
                              hInst,
                              dwReason))
      {
         return FALSE;
      }
      return TRUE;
   }
```

2. Include the following lines in the IMPORTS section of the module definition (DEF) file for DLL16:

```
   C16ThkSL01      = KERNEL.631
   ThunkConnect16  = KERNEL.651
```

3. Include the following lines in the EXPORTS section of the module definition (DEF) file for DLL16. The THK_THUNKDATA16 is defined in the

object file that is assembled from the output of the thunk compiler.

```
THK_THUNKDATA16 @1  RESIDENTNAME
DllEntryPoint   @2  RESIDENTNAME
```

4. Once you have done that you need to assemble the assembly language file produced by the thunk compiler as a 16-bit object module. The following line shows an example:

   ```
   ml /DIS_16 /c /W3 /nologo /Fo thk16.obj 16to32.asm
   ```

5. Link this object module as part of the 16-bit DLL (DLL16) object file.

6. Mark the Windows-based DLL as version 4.0. To do this you can use the resource compiler (RC.EXE). The following line shows the syntax:

   ```
   rc -40 <DLL file>
   ```

   This -40 option is available in the resource compiler that is provided with the Windows 95 SDK and later SDKs.

Additional reference words: 4.00 95 flat thunk win16
KBCategory: kbprg
KBSubcategory: SubSys BseMisc

Calling 32-bit Code from 16-bit Apps in Windows 95

PSS ID Number: Q125718
Authored 02-Feb-1995 Last modified 01-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

It is often desirable to port 16-bit Windows-based applications to Win32 a little at a time, rather than all at once. For example, you may want to port Windows-based DLLs to Win32, but still be able to call them from 16-bit code. This article discusses the mechanism by which Windows-based DLLs can call Win32-based DLLs. The mechanism is called a thunk and the method implemented under Windows 95 is called a flat thunk.

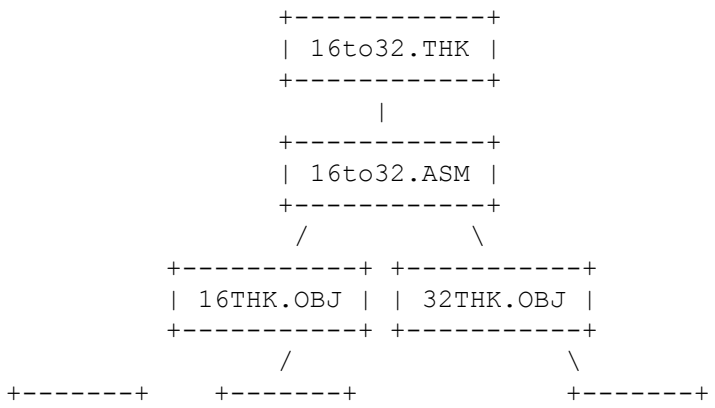
The three major steps in writing the thunk code are:

1. Creating the Thunk Script
2. Building the Windows-based DLL
3. Building the Win32-based DLL

MORE INFORMATION

The recommended way to design a thunk call is to isolate all thunk specific code in DLLs (a 16-bit DLL and a 32-bit DLL, to provide both sides of the thunk). That way, you can install certain DLLs on one platform and replace them on another platform, for portability.

Designing a new flat thunk involves creating a thunk script (.THK file). This script is compiled with the Thunk Compiler into an assembly file. This file is assembled using two different flags: -DIS_32 and -DIS_16. This allows you to create both the 16-bit and 32-bit object modules. These object modules are linked in the Windows-based and Win32-based DLLs, respectively. The following diagram summarizes the files involves in building the DLLs.



```
| APP16 | -> | DLL16 | -- THUNK -- | DLL32 |
+-------+    +-------+             +-------+
```

Creating the Thunk Script

You need to create a script that will be used by the thunk compiler to create a thunk. A thunk script contains the function prototype and a specification for the input and output values. You need to include the following statement to create a 16-bit to 32-bit thunk call:

```
enablemapdirect1632 = true
```

By default, the Win32-based DLL is loaded only on the first encounter of a 16->32 thunk. Because late binding is used, 16-bit code must not depend on any action taken by the initialization of the Win32-based DLL. Also a loading failure of the Win32-based DLL will not be detected until the first 16->32 thunk has been called. To disable late binding of the Win32-based DLL add the following line in your thunk script:

```
preload32=true;
```

The following is an example of a simple thunk script for a function that has no input and output:

```
enablemapdirect1632 = true

void MyThunk32()
{
}
```

The following is an example of script that takes two parameters and returns a value. The second parameter is an output parameter and contains a pointer that is passed back to the Windows-based DLL.

```
enablemapdirect1632 = true

typedef int   BOOL;
typedef char *LPSTR;

BOOL MyThunk32(LPSTR lpstrInput, LPSTR lpstrOutput)
{
   lpstrOutput = output;
}
```

The statement "lpstrOutput = output" tells the script compiler that the 32-bit code will return an address that needs to be converted from flat memory pointer to a selector:offset pointer.

The following thunk script uses more complex parameter types, such as structures. This example also shows how to specify input and output parameters.

```
enablemapdirect1632 = true
```

```
typedef int BOOL;
typedef unsigned int UINT;
typedef char *LPSTR;

typedef struct tagPOINT {
   INT x;
   INT y;
} POINT;
typedef POINT *LPPOINT;

typedef struct tagCIRCLE {
   POINT center;
   INT   radius;
} CIRCLE;
typedef CIRCLE *LPCIRCLE

void MyThunk32( LPCIRCLE lpCircleInOut)
{
   lpCircleInOut = InOut;
}
```

The statement "lpCircleInOut = InOut" tells the script compiler that this pointer is going to be used for input and output. This means that conversion from a 16-bit selector:offset to a flat memory pointer and vice-versa needs to be accomplished.

The thunk compiler usage is as follows:

```
thunk.exe /options <inputfile> -o <outputfile>
```

The following line shows how to create a 16-bit thunk code.

```
thunk -t thk 16to32.thk -o 16to32.asm
```

The "-t thk" option tells the thunk compiler to prefix the thunk functions in the assembly language file with "thk_". This will create an assembly language file.

Building the Windows-based DLL (DLL16)

1. The Windows-based DLL must export a function named "DllEntryPoint". This function must make a call to an imported function thk__ThunkConnect16.

```
BOOL FAR PASCAL __export DllEntryPoint(DWORD dwReason,
                                       WORD  hInst,
                                       WORD  wDS,
                                       WORD  wHeapSize,
                                       DWORD dwReserved1,
                                       WORD  wReserved 2)
{
   if (!thk_ThunkConnect16("DLL16.DLL",
                           "DLL32.DLL",
                           hInst,
                           dwReason))
```

```
        {
           return FALSE;
        }
        return TRUE;
     }
```

2. Include the following lines in the IMPORTS section of the module definition (DEF) file for DLL16.

```
   C16ThkSL01      = KERNEL.631
   ThunkConnect16  = KERNEL.651
```

3. Include the following lines in the EXPORTS section of the module definition (DEF) file for DLL16. The THK_THUNKDATA16 is defined in the object file that is assembled from the output of the thunk compiler.

```
   THK_THUNKDATA16 @1  RESIDENTNAME
   DllEntryPoint   @2  RESIDENTNAME
```

4. Once you have done that you need to assemble the assembly language file produced by the thunk compiler as a 16-bit object module. The following line shows an example:

```
   ml /DIS_16 /c /W3 /nologo /Fo thk16.obj 16to32.asm
```

5. Link this object module as part of the 16-bit DLL (DLL16) object file.

6. Mark the Windows-based DLL as version 4.0. To do this you can use the resource compiler (RC.EXE). The following line shows the syntax:

```
   rc -40 <DLL file>
```

   This -40 option is available in the resource compiler that is provided with the Windows 95 SDK and later SDKs.

Building the Win32-based DLL (DLL32)

1. In the DllEntryPoint function (DllMain if you're using the Microsoft C Run-time libraries) in your Win32-based DLL, you must make a call to the imported function thk_ThunkConnect32, as shown here:

```
   BOOL WINAPI DllMain(HINSTANCE hDLLInst,
                       DWORD fdwReason,
                       LPVOID lpvReserved)
   {
      if (!thk_ThunkConnect32("DLL16.DLL",
                              "DLL32.DLL",
                              hDLLInst,
                              fdwReason))
      {
         return FALSE;
      }
      switch (fdwReason)
      {
```

```
            case DLL_PROCESS_ATTACH:
               break;

            case DLL_PROCESS_DETACH:
               break;

            case DLL_THREAD_ATTACH:
               break;

            case DLL_THREAD_DETACH:
               break;
         }
         return TRUE;
      }
```

2. Include the following lines into the EXPORT section of the module definition (DEF) file for DLL32:

   ```
   thk_ThunkData32
   ```

3. Export the function that you are thunking to.

4. Assemble the assembly language file produced by the thunk compiler as a 32-bit object module. The following line shows an example:

   ```
   ml /DIS_32 /c /W3 /nologo /coff /Fo thk32.obj 16to32.asm
   ```

5. Link this object module as part of the Win32-based DLL (Win32).

Additional reference words: 4.00 95 flat thunk win16
KBCategory: kbprg
KBSubcategory: SubSys BseMisc

Calling a New 32-bit API from a 16-bit Application

PSS ID Number: Q125674
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

Windows 95 supports a new set of APIs for 32-bit applications. These APIs are exported by USER32, GDI32, KERNEL32, and so on. In Windows 95, some of these new APIs are also exported by the 16-bit counterpart DLLs in the system such as USER16, GDI16, and so on. But 16-bit applications running on Windows 95 should not call these new APIs from the 16-bit system DLLs because these calls are not unsupported and might be removed from the 16-bit system DLLs in the future.

MORE INFORMATION

APIs such as WindowFromDC(), SetWindowRgn(), SetForeGroundWindow(), and so on for 16-bit USER window management and PolyBezier(), PolyBezierTo(), and so on for 16-bit GDI graphics management are exported from the 16-bit system DLLs.

Even though these APIs are intended for 32-bit applications, the 16-bit system DLLs export some of them. 16-bit Windows 95 Applications should not call them. They are not supported and the APIs will not work as intended.

If Windows 95 Applications need to use these APIs, port the 16-bit application to 32-bit. This is the best solution and is the one that Microsoft recommends. One addtional solution is to write a 32-bit DLL that actually calls the 32-bit API; then the 16-bit application can thunk into this 32-bit DLL. However, Microsoft strongly discourages developers from having applications thunk into sytem DLLs (16- or 32-bit).

Additional reference words: 4.00
KBCategory: kbprg
KBSubcategory: UsrMisc

Calling a Win32 DLL from a Win16 App on Win32s

PSS ID Number: Q97785
Authored 21-Apr-1993 Last modified 22-Mar-1995

The information in this article applies to:

- Microsoft Win32s versions 1.1 and 1.2

SUMMARY

A Windows version 3.1 application can call a Win32 dynamic-link library (DLL) under Win32s using Universal Thunks.

The following are required components (in addition to the Windows 3.1 application and the Win32 DLL):

- A 16-bit DLL that provides the same entry points as the Win32 DLL. This serves as the 16-bit end for the Universal Thunk. The programmer must also include code that will detect whether the 32-bit side is loaded.

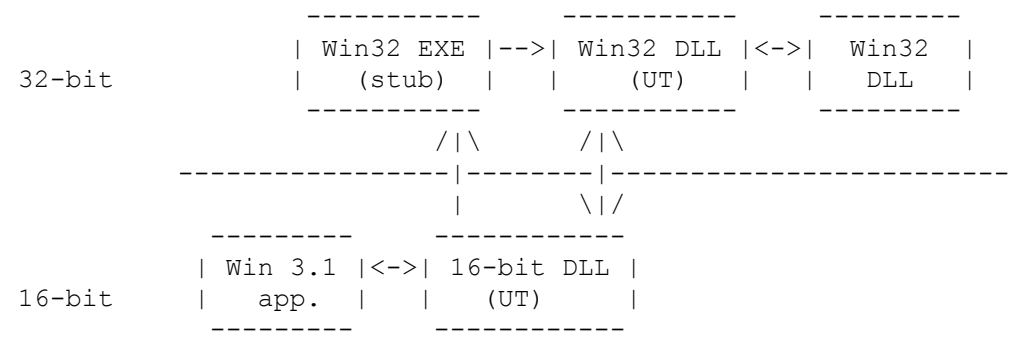
- A Win32 DLL that sets up the Universal Thunk. This serves as the 32-bit end of the Universal Thunk. This DLL is supported only under Win32s.

- A Win32 EXE that loads the 32-bit DLL described above.

NOTE: Universal Thunks were designed to work with a Win32-based application calling a 16-bit DLL. The method described here has limitations. Because the application is 16-bit, no 32-bit context is created, so certain calls will not work from the Win32 DLL. For example, the first time you call malloc() or new() from a DLL entrypoint called by the 16-bit application, the system will hang. This is because MSVCRT20.DLL is using TLS to store C Run-time state information and there is no TLS set up.

MORE INFORMATION

The following diagram illustrates how the pieces fit together:



The load order is as follows: The Windows 3.1 application loads the 16-bit DLL. The 16-bit DLL checks to see whether the 32-bit side has been initialized. If it has not been initialized, then the DLL spawns the 32-bit EXE (stub), which then loads the 32-bit DLL that sets up the Universal Thunks with the 16-bit DLL. Once all of the components are loaded and initialized, when the Windows 3.x application calls an entry point in the 16-bit DLL, the 16-bit DLL uses the 32-bit Universal Thunk callback to pass the data over to the 32-bit side. Once the call has been received on the 32-bit side, the proper Win32 DLL entry point can be called.

Note that the components labeled Win32 DLL (UT) and Win32 DLL in the diagram above can be contained in the same Win32 DLL. Remember that the code in the Win32 DLL (UT) portion isn't supported under Windows NT, so this code must be special-cased if the DLLs are combined.

For more information, please see the "Win32s Programmer's Reference."

For a sample program, see the following file on CompuServe, in the MSWIN32 forum:

```
   LIB 14 !dir /des /long /lib:all rev*.*
   [76150,2543]    Lib:14
   REVUT./Bin      Bytes:   8848, Count:  205, 10-Jan-94   Last:29-Jul-94

      Title   : Revut
      Keywords: UT UNIVERSAL THUNK WIN32S 32-BIT DLL 16-BIT APP

      16-bit app calling a 32 bit dll under win32s (reverse UT)
```

Additional reference words: 1.10 1.20 reverse universal thunk
KBCategory: kbprg
KBSubcategory: W32s

Calling a Win32 DLL from a Win16 Application Under WOW

PSS ID Number: Q104009
Authored 02-Sep-1993 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

Under Windows NT, it is possible to call routines in a Win32 dynamic-link library (DLL) from a 16-bit Windows application using an interface called Windows on Win32 (WOW) Generic Thunking. This is not to be confused with Win32s Universal Thunks, which provides this functionality under Windows 3.1.

NOTE: Windows 95 also supports Generic Thunks.

WOW presents a few new 16-bit application programming interfaces (APIs) that allow you to load the Win32 DLL, get the address of the DLL routine, call the routine (passing it up to thirty-two 32-bit arguments), convert 16:16 (WOW) addresses to 0:32 addresses (useful if you need to build up a 32-bit structure that contains pointers and pass a pointer to it), and free the Win32 DLL.

The Generic Thunks specification is included on the Win32 SDK CD in \DOC\SDK\MISC\GENTHUNK.TXT. However, the 3.1 version of this document does not include correct information for all of the function prototypes. The 3.5 version of the document has been corrected and it includes information on new APIs introduced in 3.5 for thunking.

The following prototypes should be used:

```
DWORD FAR PASCAL LoadLibraryEx32W( LPCSTR, DWORD, DWORD );
DWORD FAR PASCAL GetProcAddress32W( DWORD, LPCSTR );
DWORD FAR PASCAL CallProc32W( DWORD, ..., LPVOID, DWORD, DWORD );
DWORD FAR PASCAL GetVDMPointer32W( LPVOID, UINT );
BOOL FAR PASCAL FreeLibrary32W( DWORD );
```

Note that although these functions are called in 16-bit code, they need to be provided with 32-bit handles, and they return 32-bit handles.

In addition, be sure that your DLL routines are declared with the _stdcall convention; otherwise, you will get an access violation.

CallProc32W() is a Pascal function which was designed to take a variable number of arguments, a Proc address, a mask, and the number of parameters. The mask is used to specific which arguments should be treated as being passed by value and which parameters should be translated from 16:16 pointers to flat pointers. Note that the low-order bit of the mask represents the last parameter, the next lowest bit represents the next to the last parameter, and so forth.

NOTE: It is a good idea to test the Win32 DLL by calling it from a Win32-based application before attemting to call it from a 16-bit Windows-based application, because the debugging support is superior in the 32-bit environment.

MORE INFORMATION

Sample

The following code fragments can be used as a basis for Generic Thunks. Assume that the 16-bit Windows-based application is named app16, that the Win32 DLL is named dll32, and that the following are declared:

```
typedef void (FAR PASCAL *MYPROC)(LPSTR);

DWORD ghLib;
MYPROC hProc;
char FAR *TestString = "Hello there";
```

The DLL routine is defined in dll32.c as follows:

```
void WINAPI MyPrint( LPTSTR lpString, HANDLE hWnd )
{
   ...
}
```

Attempt to load the library in the app16 WinMain():

```
if( NULL == (ghLib = LoadLibraryEx32W( "dll32.dll", NULL, 0 )) ) {
   MessageBox( NULL, "Cannot load DLL32", "App16", MB_OK );
   return 0;
}
```

Attempt to get the address of MyPrint():

```
if( NULL == (hProc = (MYPROC)GetProcAddress32W( ghLib, "MyPrint" ))) {
   MessageBox( hWnd, "Cannot call DLL function", "App16", MB_OK );
   ...
}
```

Call MyPrint() and pass it TestString and hWnd as arguments:

```
CallProc32W( (DWORD) TestString, (DWORD) hWnd|0xffff0000, hProc, 2, 2 );
```

The hWnd is OR'd with 0xffff0000, because this is the way to convert a 16-bit window handle to a 32-bit window handle in Windows NT. If you wanted to convert a 32-bit window handle to a 16-bit window handle, you would simply truncate the upper word. Note that this only works for windows handles, not for other types of handles. In Windows NT 3.5, you should use the following functions exported by WOW32.DLL: WOWHandle32() and WOWHandle16(), rather than relying on this relationship. These functions (and many others) are discussed in \DOC\SDK\MISC\GENTHUNK.TXT.

A mask of 2 (0x10) is given because we want to pass TestString by reference (WOW translates the pointer) and we want to pass the handle by value.

Free the library right before exiting WinMain():

```
FreeLibrary32W( ghLib );
```

NOTE: When linking the Windows-based application, you need to put the following statements in the .DEF file, indicating that the functions will be imported from the WOW kernel:

```
IMPORTS
   kernel.LoadLibraryEx32W
   kernel.FreeLibrary32W
   kernel.GetProcAddress32W
   kernel.GetVDMPointer32W
   kernel.CallProc32W
```

The complete sample is also available in the Microsoft Software Library.

Download GTHUNKS.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for GTHUNKS.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download GTHUNKS.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \softlib\mslfiles directory
  Get GTHUNKS.EXE

Faults

On MIPS systems, an alignment fault will occur when a Win32-based application deferences a pointer to unaligned data that was passed by a 16-bit Windows application. As a workaround, declare the parameter with the UNALIGNED keyword. For example,

```
void func( DWORD *var );
```

becomes

```
void func( DWORD unaligned *var);
```

An application can use SetErrorMode() to specify SEM_NOALIGMENTFAULTEXCEPT flag. If this is done, the system will automatically fix up alignment

faults and make them invisible to the application.

The default value of this error mode is OFF for MIPS, and ON for ALPHA. So on MIPS platforms, an app MUST call SetErrorMode() and specify SEM_NOALIGMENTFAULTEXCEPT if it wants the system to automatically fix alignment faults. This call does not have to be made on ALPHA platforms. This flag has no effect on x86 systems. Note that the fix above is preferable.

Additional reference words: 3.10 3.50 4.00 softlib GTHUNKS.EXE
KBCategory: kbprg kbfile
KBSubcategory: SubSys

Calling a Win32 DLL from an OS/2 Application

PSS ID Number: Q119217
Authored 10-Aug-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, version 3.5

SUMMARY

The OS/2 subsystem provides a general mechanism that allows 16-bit OS/2 and PM applications to load and call a Win32 DLL. This feature is useful if you would like to call one of the Win32 APIs without having to spawn a Win32 application to do so; it is also useful if you are porting your OS/2 application to Win32 in stages. A small set of APIs (described in the "MORE INFORMATION" section, below) provides this functionality to your application.

To take full advantage of this feature, write a small Win32 DLL to provide the thunk. The OS/2 application calls the thunk DLL, which in turn calls the real Win32 API, using the parameters that were passed from the OS/2 application. You could call the Win32 DLL directly, but the OS/2 subsystem thunking mechanism has only one generic pointer parameter. Most Win32 APIs require different numbers or types of parameters, so the thunking layer is required to retrieve the parameters from the parameter pointer. The parameter pointer typically points to a structure that contains the actual parameters.

You will also need to modify your OS/2 application to call the thunking APIs described below.

MORE INFORMATION

The following are the header file, descriptions, and import statements for the 16-bit APIs that are used by the OS/2 application to load and call the thunk DLL. The APIs are defined in the same manner as the OS/2 APIs.

Header File

```
extern USHORT pascal far
Dos32LoadModule(PSZ DllName, PULONG pDllHandle);

extern USHORT pascal far
Dos32GetProcAddr(ULONG Handle, PSZ pszProcName, PULONG pWin32Thunk);

extern USHORT pascal far
Dos32Dispatch(ULONG Win32Thunk, PVOID pArguments, PULONG pRetCode);

extern USHORT pascal far
```

```
   Dos32FreeModule(ULONG DllHandle);

   extern USHORT pascal far
   FarPtr2FlatPtr(ULONG FarPtr, PULONG pFlarPtr);

   extern USHORT pascal far
   FlatPtr2FarPtr(ULONG FlatPtr, PULONG pFarPtr);
```

API Descriptions

Dos32LoadModule

```
   USHORT pascal far Dos32LoadModule (
      PSZ    DLLName,
      PULONG pDllHandle );
```

Purpose:

Load the Win32 thunk DLL that will intermediate between the OS/2 application and the Win32 APIs.

Parameters:

DLLName - Name of the thunk DLL.
pDllHandle - Receives the module handle of the DLL. Used as an argument to Dos32GetProcAddr.

Return:

If NO_ERROR is returned, the value pointed to by pDllHandle is used for other WIN32 thunk APIs as described below. It is invalid for usage with regular OS/2 APIs.

If ERROR_MOD_NOT_FOUND is returned, the value pointed to by pDLLHandle is undefined.

Dos32GetProcAddr

```
   USHORT pascal far Dos32GetProcAddr (
      ULONG  DllHandle,
      PSZ    pszProcName,
      PULONG pWin32Thunk );
```

Purpose:

Get a flat pointer to a routine in the Win32 thunk DLL previously opened by Dos32LoadModule.

Parameters:

DllHandle - Handle obtained through Dos32LoadModule.
pszProcName - Name of the API to be called.

pWin32Thunk - Receives pointer to the thunk routine.

Return:

If NO_ERROR is returned, then pszProcName is exported by the thunk DLL.

If ERROR_PROC_NOT_FOUND or ERROR_INVALID_HANDLE is returned, then the value pointed to by pWin32Thunk is undefined.

Dos32Dispatch

```
USHORT pascal far Dos32Dispatch (
   ULONG  Win32Thunk,
   PVOID  pArguments,
   PULONG pRetCode );
```

Purpose:

Call the thunk routine through the pointer obtained through Dos32GetProcAddr.

Parameters:

Win32Thunk - Thunk routine obtained through Dos32GetProcAddr.
pArguments - Argument for thunk routine.
pRetCode   - Error code returned from the thunk routine.

The structure pointed to by pArguments is application specific and the value is not modified to the OS/2 subsystem.

On the Win32 side (in the thunk DLL), the thunk looks like:

```
   ULONG MyFunc (
      PVOID pFlatArg );
```

The OS/2 subsystem does translate pArguments from a 16:16 pointer to a flat pointer (pFlatArg). To translate pointers inside the structure, use FarPtr2FlatPtr.

Return:

If NO_ERROR is returned, then pFlatArg is a valid pointer.

Dos32FreeModule

```
USHORT pascal far Dos32FreeModule (
   ULONG DllHandle );
```

Purpose:

Unload the Win32 thunk DLL that is intermediating between the OS/2 application and the Win32 APIs.

Parameter:

DllHandle - Handle obtained through Dos32LoadModule.

Return:

If NO_ERROR is returned, then DllHandle corresponds to a Win32 DLL previously loaded by Dos32LoadModule. After the call, DllHandle is no longer valid. Otherwise, ERROR_INVALID_HANDLE is returned.

FarPtr2FlatPtr

```
USHORT pascal far FarPtr2FlatPtr (
   ULONG  FarPtr,
   PULONG pFlatPtr );
```

Purpose:

Translates a segmented far pointer to a flat pointer.

Parameters:

FarPtr   - Segmented far pointer.
pFlatPtr - Points to flat pointer.

Return:

If NO_ERROR is returned, FarPtr is a valid 16:16 pointer and pFlatPtr contains a valid flat pointer to be used by Win32 code. Otherwise, ERROR_INVALID_PARAMETER is returned and pFlatPtr is undefined.

FlatPtr2FarPtr

```
USHORT pascal far FlatPtr2FarPtr (
   ULONG  FlatPtr,
   PULONG pFarPtr );
```

Purpose:

Translates a flat pointer to a segmented far pointer.

Parameters:

FlatPtr - Flat pointer.
pFarPtr - Points to a segmented far pointer.

Return:

If NO_ERROR is returned, the flat pointer maps to a valid segmented pointer in the 16-bit application's context and pFarPtr contains a valid segmented pointer to be used by 16-bit OS/2 code. Otherwise, ERROR_INVALID_PARAMETER is returned and pFarPtr is undefined.

IMPORTS Section for the Module Definition File

```
   IMPORTS
      DOSCALLS.DOS32LOADMODULE
      DOSCALLS.DOS32GETPROCADDR
      DOSCALLS.DOS32DISPATCH
      DOSCALLS.DOS32FREEMODULE
      DOSCALLS.FARPTR2FLATPTR
      DOSCALLS.FLATPTR2FARPTR
```

Additional reference words: 3.50 OS2
KBCategory: kbprg
KBSubcategory: SubSys

Calling CRT Output Routines from a GUI Application

PSS ID Number: Q105305
Authored 17-Oct-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

To use C Run-time output routines, such as printf(), from a GUI application, it is necessary to create a console. The Win32 application programming interface (API) AllocConsole() creates the console. The CRT routine setvbuf() removes buffering so that output is visible immediately.

This method works if the GUI application is run from the command line or from File Manager. However, this method does not work if the application is started from the Program Manager or via the "start" command. The following code shows how to work around this problem:

```
int hCrt;
FILE *hf;

AllocConsole();
hCrt = _open_osfhandle(
          (long) GetStdHandle(STD_OUTPUT_HANDLE),
          _O_TEXT
       );
hf = _fdopen( hCrt, "w" );
*stdout = *hf;
i = setvbuf( stdout, NULL, _IONBF, 0 );
```

This code opens up a new low-level CRT handle to the correct console output handle, associates a new stream with that low-level handle, and replaces stdout with that new stream. This process takes care of functions that use stdout, such as printf(), puts(), and so forth. Use the same procedure for stdin and stderr.

Note that this code does not correct problems with handles 0, 1, and 2. In fact, due to other complications, it is not possible to correct this, and therefore it is necessary to use stream I/O instead of low-level I/O.

MORE INFORMATION

When a GUI application is started with the "start" command, the three standard OS handles STD_INPUT_HANDLE, STD_OUTPUT_HANDLE, and STD_ERROR_HANDLE are all "zeroed out" by the console initialization routines. These three handles are replaced by valid values when the GUI application calls AllocConsole(). Therefore, once this is done, calling GetStdHandle() will always return valid handle values. The problem is that

the CRT has already completed initialization before your application gets a chance to call AllocConsole(); the three low I/O handles 0, 1, and 2 have already been set up to use the original zeroed out OS handles, so all CRT I/O is sent to invalid OS handles and CRT output does not appear in the console. Use the workaround described above to eliminate this problem.

In the case of starting the GUI application from the command line without the "start" command, the standard OS handles are NOT correctly zeroed out, but are incorrectly inherited from CMD.EXE. When the application's CRT initializes, the three low I/O handles 0, 1, and 2 are initialized to use the three handle numbers that the application inherits from CMD.EXE. When the application calls AllocConsole(), the console initialization routines attempt to replace what the console initialization believes to be invalid standard OS handle values with valid handle values from the new console. By coincidence, because the console initialization routines tend to give out the same three values for the standard OS handles, the console initilization will replace the standard OS handle values with the same values that were there before--the ones inherited from CMD.EXE. Therefore, CRT I/O works in this case.

It is important to realize that the ability to use CRT routines from a GUI application run from the command line was not by design so this may not work in future versions of Windows NT or Windows. In a future version, you may need the workaround not just for applications started on the command line with "start <application name>", but also for applications started on the command line with "application name".

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseMisc

Calling DdePostAdvise() from XTYP_ADVREQ

PSS ID Number: Q102571
Authored 04-Aug-1993 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

The documentation for DdePostAdvise() in the Windows 3.1 Software Development Kit "Programmer's Reference, Volume 2: Functions" states the following in the Comments section:

> If a server calls DdePostAdvise() with a topic/item/format name set that includes the set currently being handled in an XTYP_ADVREQ callback, a stack overflow may result.

MORE INFORMATION

This is merely a warning against calling DdePostAdvise() from within a DDE callback function's XTYP_ADVREQ transaction, because it may result in a stack overflow.

Like window procedures, DDE callbacks must be coded with care to avoid infinite recursion (eventually resulting in a stack overflow). Because DdePostAdvise() causes DDEML to send an XTYP_ADVREQ transaction to the calling application's DDE callback function, calling DdePostAdvise() on the same topic/item/format name set as the one currently being handled results in an infinite loop.

An analogous piece of code that has become a classic problem in Windows programming involves calling UpdateWindow() in a WM_PAINT case:

```
case WM_PAINT:
    InvalidateRect (hWnd, NULL, TRUE);
    UpdateWindow (hWnd);
```

Calling UpdateWindow() as in the code above causes a WM_PAINT message to be sent to a window procedure, and thus results in the same type of infinite recursion that occurs when calling DdePostAdvise() from an XTYP_ADVREQ transaction.

An example of a situation that would lend itself to this scenario would be one where data needs to be updated as a result of a previous data change. There are two ways to work around the stack overflow problem in this case:

- Post a user-defined message and handle the data change

asynchronously. For example,

```
// in DdeCallback:
case XTYP_ADVREQ:
          if ((!DdeCmpStringHandles (hsz1, ghszTopic)) &&
              (!DdeCmpStringHandles (hsz2, ghszItem)) &&
              (fmt == CF_SOMEFORMAT))
            {
                  HDDEDATA  hData;

                  hData = DdeCreateDataHandle ();
                  PostMessage (hWnd, WM_DATACHANGED,hData,);
                  return (hData);
            }
            break;

 // in MainWndProc():
 case WM_DATACHANGED:
         DdePostAdvise (idInst, ghszTopic, ghszItem);
           :
```

- Return CBR_BLOCK from the XTYP_ADVREQ and let DDEML suspend further transactions on that conversation, while the server prepares data asynchronously.

More information on how returning CBR_BLOCK allows an application to process data "asynchronously" may be derived from Section 5.8.6 of the Windows 3.1 Software Development Kit (SDK) "Programmer's Reference, Volume 1: Overview," or by querying on the following words in the Microsoft Knowledge Base:

```
DDEML and CBR_BLOCK
```

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDde

Calling LoadLibrary() on a 16-bit DLL

PSS ID Number: Q123731
Authored 07-Dec-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.15a and 1.2

SUMMARY

In Win32-based applications, LoadLibrary() returns an HINSTANCE and GetLastError() is used to determine the error. If HINSTANCE is NULL, the DLL was not successfully loaded. If the HINSTANCE is not null, the DLL was loaded and the usage count was incremented; however, you may still see that the last error was set if the DLL is a 16-bit DLL.

NOTE: At this point, the DLL is loaded and the usage count is incremented. Call FreeLibrary() to unload the DLL.

MORE INFORMATION

In order to see all possible error returns, you'll need to call SetLastError(0) before calling LoadLibrary(). If HINSTANCE is not NULL and GetLastError() is ERROR_BAD_EXE_FORMAT, the DLL is a 16-bit DLL. You can access the DLL resources and/or printer APIs from your Win32-based application.

To call routines in the 16-bit DLL, you should load and call the DLL via the Universal Thunk. This increments the usage count again. Later, you can use FreeLibrary() to free the DLL from the 16-bit code, but this won't unload the DLL from memory unless you already called FreeLibrary() from the 32-bit code. This is because the usage count is not zero. We recommend you call FreeLibrary() from the 32-bit code after the DLL is loaded by the 16-bit code, so the DLL isn't unloaded and then reloaded.

REFERENCES

For more information on how to get resources from a 16-bit DLL, please see the following article in the Microsoft Knowledge Base:

```
ARTICLE-ID: Q105761
TITLE     : Getting Resources from 16-bit DLLs Under Win32s
```

For more information on Universal Thunk, please see Chapter 4 of the Win32s Programmer's Reference.

Additional reference words: 1.20
KBCategory: kbprg
KBSubcategory: W32s

Cancelling Overlapped I/O

PSS ID Number: Q90368
Authored 13-Oct-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

There is no routine in the Win32 API to cancel an asynchronous request once it has been issued. When a thread does an overlapped I/O (that is, a write), the system starts up another thread to do the I/O and leaves your thread free to do other work. Once it is started, there is no way to stop it.

If it necessary to interrupt the I/O, you can either

- Split the writes into batches and check for interruptions. For example, you could break a 20 megabyte (MB) write into 20, 1 MB writes.

  -or-

- Create another thread yourself to handle the I/O. Terminating the thread will cancel the I/O. You should have a thread in the process explicitly close the handle to the device.

  -or-

- Close the handle to the device with the pending I/O. The close has the net effect of cancelling the I/O.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseFileio

Cancelling WaitCommEvent() with SetCommMask()

PSS ID Number: Q105302
Authored 17-Oct-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

If a serial port is in nonoverlapped mode (without FILE_FLAG_OVERLAPPED) and SetCommMask() is called, the call does not return until any pending WaitCommEvent() calls return. This apparently contradicts the following statement from the SetCommMask() Help

> If SetCommMask() is called for a communications resource while a wait is pending for that resource, WaitCommEvent() returns an error.

and the following statement from the WaitCommEvent() Help:

> If a process attempts to change the device handle's event mask by using the SetCommMask() function while a WaitCommEvent() operation is in progress, WaitCommEvent() returns immediately.

However, this is the expected behavior. If you open a serial port in the nonoverlapped mode, then you can do only one thing at a time with the serial port. SetCommMask() must block while the WaitCommEvent() call is blocking.

If the serial port was opened with FILE_FLAG_OVERLAPPED, WaitCommEvent() will return after SetCommEvent() has been called.

Additional reference words: 3.10 com1 com2
KBCategory: kbprg
KBSubcategory: BseCommapi

Cannot Load <exe> Because NTVDM Is Already Running

PSS ID Number: Q103863
Authored 01-Sep-1993 Last modified 23-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

WinDbg can debug 16-bit Windows-based applications running on Windows NT, under the Win16 VDM (virtual MS-DOS machine), NTVDM. By default, each 16-bit Windows-based application run as a thread in NTVDM. On Windows NT 3.5, each application can be run in a separate address space.

If NTVDM is running when the debugger tries to start the application, you get the following error message:

```
Cannot load <exe> because NTVDM is already running
```

Under Windows NT 3.5, to work around this problem, go to the WinDbg Options menu, choose Debug, and check Separate WOW VDM, to allow the debuggee to be run in a different address space.

Alternatively, you could terminate NTVDM. On Windows NT 3.1, your only choice is to terminate NTVDM, because separate address spaces for 16-bit Windows-based application are not supported.

To terminate NTVDM, run PView, select NTVDM, and choose "Kill Process." Note that there may be two NTVDM processes. The one that you want to terminate has one thread for each Windows-based application (plus a few more).

The Windows NT WinLogon is set up to automatically start WoWExec, which automatically starts the Win16 VDM. This behavior can be changed by removing WoWExec from:

```
HKEY_LOCAL_MACHINE\
          Software\
         Microsoft\
        Windows NT\
    CurrentVersion\
          Winlogon\
             Shell
```

Additional reference words: 3.10 3.50
KBCategory: kbtool
KBSubcategory: TlsWindbg

Captions for Dialog List Boxes

PSS ID Number: Q24646
Authored 16-Dec-1987 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

To place text into the caption bar specified for a list box, use the SetWindowText() function. First, use GetDlgItem() to get the handle of the list box, then call SetWindowText() to set the list box caption.

MORE INFORMATION

The following code fragment illustrates the necessary steps. Note: The list box includes the WS_CAPTION window style.

```
   ...
   BOOL FAR PASCAL TemplateDlg(hWndDlg, message, wParam, lParam)
   ...
       switch (message)
        {
       case WM_INITDIALOG:
           ...
           /* The following line sets the Listbox caption */
           SetWindowText( GetDlgItem(hWndDlg, IDDLISTBOX),
                          (LPSTR)"Caption");
           for (i = 0; i < CSTR; i++) {
               LoadString(hInstTemplate, IDSSTR1+i, (LPSTR)szWaters, 12);
               SendDlgItemMessage(hWndDlg, IDDLISTBOX, LB_ADDSTRING, 0,
                   (LONG)(LPSTR)szWaters);
               }
           SendDlgItemMessage(hWndDlg, IDDLISTBOX, LB_SETCURSEL, iSel, 0L);
           ...
           return TRUE;

       case WM_COMMAND:
   ...
```

Additional reference words: 3.00 3.10 3.50 4.00 95 listbox
KBCategory: kbprg
KBSubcategory: UsrCtl

Caret Position & Line Numbers in Multiline Edit Controls

PSS ID Number: Q68572
Authored 22-Jan-1991 Last modified 27-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

This article explains how to determine the position (row and column) of the caret and the line number of the first visible line of text in a multiline edit control.

MORE INFORMATION

Edit controls process several messages that return information relevant to the position of the caret within the control. These messages help an application determine the line number of the caret relative to the number of lines of text in the control.

Once the line number is known, the application can compute the caret's character position within that line and the line number of the first visible line of text in the control.

An edit control must be subclassed in order to track the caret position because the position changes with mouse clicks and keystrokes. The subclass procedure must process the WM_KEYDOWN and WM_LBUTTONDOWN messages, and compute the caret position upon receipt of each message.

The remainder of this article describes three procedures:

- Finding the line number of the caret position
- Finding the column number of the caret position
- Finding the line number of the first visible line

Note that you may replace any mention of the SendMessage API in this article with the SendDlgItemMessage function. Also note that the term return value refers to the value returned by the SendMessage or the SendDlgItemMessage function.

Finding the Line Number of the Caret Position

Perform the following two steps:

1. Send the EM_GETSEL message to the edit control. The high-order word of the return value is the character position of the caret relative to the first character in the control.

2. Send the EM_LINEFROMCHAR message to the edit control and specify the value returned from step 1 as wParam. Add 1 to the return value to get the line number of the caret position because Windows numbers the lines starting at zero.

Finding the Column Number of the Caret Position

Perform the following three steps:

1. Send the EM_GETSEL message to the edit control. The high-order word of the return value is the character position of the caret relative to the first character in the control.

2. Send the EM_LINEINDEX message with wParam set to -1. The value returned is the count of characters that precede the first character in the line containing the caret.

3. Subtract the value returned in step 2 from the value in step 1 and add 1 because Windows numbers the columns starting at zero. This result is the column number of the caret position.

Finding the Line Number of the First Visible Line

Windows 3.1 and later define the EM_GETFIRSTVISIBLELINE message, which an application can send to a single line or a multiline edit control. For single line edit controls, this value returned for the message is the offset of the first visible character. For multiline edit controls, the value returned is the number of the first visible line.

Under Windows 95, it would be more efficient to use a combination of GetCaretPos() and EM_CHARFROMPOS.

If an application must be compatible with Windows 3.0, it can perform the following 10-step procedure:

1. Follow steps 1 and 2 of "Finding the Line Number of the Caret Position," presented above, and save the line number.

2. Call the GetCaretPos function to fill a POINT structure with the caret's coordinates relative to the client area of the edit control. (The client area is inside the border.)

3. Call the GetDC function using the handle to the edit control to retrieve a handle to a device context for the edit control. Store this handle in a variable named hDC.

4. Send the WM_GETFONT message to the edit control. The return value is a handle to the font used by the edit control. If the value returned is NULL, proceed to step 6 because the control is using the system default font.

5. Call the SelectObject function to select the font used by the edit control into hDC. Do not call the SelectObject function if

WM_GETFONT returned NULL in step 4. Save the value returned by SelectObject in the hOldFont variable.

6. Call the GetTextMetrics function with hDC to fill a TEXTMETRIC data structure with information about the font used by the edit control (the font which is selected into hDC). The field of interest is tmHeight.

7. While the vertical coordinate of the caret is greater than the value of tmHeight, subtract tmHeight from the vertical coordinate and subtract 1 from the line number of the caret from step 1.

8. Repeat step 7 until the vertical coordinate of the caret is less than or equal to tmHeight.

9. Call SelectObject to select hOldFont back into hDC. Then call ReleaseDC to return the display context to the system.

10. The value remaining in the line number variable is the line number of the first visible line in the edit control.

Additional reference words: 3.00 3.10 3.50 4.00 95 caretpos
KBCategory: kbprg
KBSubcategory: UsrCrt

Case Sensitivity in Atoms

PSS ID Number: Q38901
Authored 07-Dec-1988 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

If the same string is added to an atom table twice, but a different case is used, the string is only stored once; only the first one is present.

The "Microsoft Windows 2.0 Software Development Kit Update" for versions 2.03 and 2.1 states that atoms are case insensitive.

This means that when the AddAtom() function is used, the case is ignored when atoms are compared. Therefore, if AddAtom("DIR") is called, and then AddAtom("dir"), the single atom "DIR" (with a reference count of 2) will result. If AddAtom("dir") is called first, the single atom "dir" will result.

Similarly, the other atom-handling functions are case insensitive. For example, calling FindAtom("dIr") will find the atom "dir", "DIR", or "Dir", and so on.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDde

Cases Where "Normal" Window Position, Size Not Available

PSS ID Number: Q68583
Authored 22-Jan-1991 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

In Windows version 3.0, the "normal" size and position of a window is not available when that window is maximized (zoomed) or minimized (represented as an icon). In Windows version 3.1, two new functions named GeWindowPlacement and SetWindowPlacement have been added which provide access to normal position information.

MORE INFORMATION

The remainder of this article provides two possible ways to work around this limitation in Windows version 3.0:

1. If the normal size is needed only as the application is shut down, restore the window and retrieve its position before closing the application. The following call can be used to restore the window whose window handle is hWnd:

   ```
   SendMessage(hWnd, WM_SYSCOMMAND, SC_RESTORE, 0L);
   ```

   The GetWindowRect or GetClientRect functions can then be used to obtain the window's position.

2. If the normal size is needed at all times, keep track of the position every time the window receives a WM_MOVE message. If the IsIconic and IsZoomed functions both return FALSE, assume the window is normal and update the position values. Otherwise, do not change the saved position information.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrWndw

Centering a Dialog Box on the Screen

PSS ID Number: Q74798
Authored 30-Jul-1991 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK, version 3.5 and 3.51

When an application developed for the Microsoft Windows graphical environment displays a dialog box, centering the dialog box on the screen is sometimes desirable. However, on systems with high-resolution displays, the application displaying the dialog box may be nowhere near the center of the screen. In these cases, it is preferable to place the dialog near the application requesting input.

To center a dialog box on the screen before it is visible, add the following lines to the processing of the WM_INITDIALOG message:

```
    {
    RECT rc;

    GetWindowRect(hDlg, &rc);

    SetWindowPos(hDlg, NULL,
       ((GetSystemMetrics(SM_CXSCREEN) - (rc.right - rc.left)) / 2),
       ((GetSystemMetrics(SM_CYSCREEN) - (rc.bottom - rc.top)) / 2),
       0, 0, SWP_NOSIZE | SWP_NOACTIVATE);
    }
```

This code centers the dialog horizontally and vertically.

Under Windows 95, you should use the new style DS_CENTER to get the same effect.

Additional reference words: 3.00 3.10 3.50
KBCategory: kbprg
KBSubcategory: UsrDlgs

Chaining Parent PSP Environment Variables

PSS ID Number: Q96209
Authored 10-Mar-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

Some MS-DOS-based applications change the environment variables of their parent application by chaining through the program segment prefix (PSP). With Windows NT, this functionality doesn't work if the parent is a Win32-based application.

MORE INFORMATION

When an MS-DOS-based application is started from a single command shell (SCS), the application inherits a new copy of the environment variables. Any attempts by the MS-DOS-based application to modify its parent's environment variables will not work. When the MS-DOS-based application exits, the SCS will be "restored" to its original state. If another MS-DOS-based application is started, the second application will receive the same environment that the first MS-DOS-based application received.

If an MS-DOS-based application (B) is spawned by another MS-DOS-based application (A), any modifications to application A's environment variables will be reflected when application B exits.

For more information on how environment variables are set, please see the following article(s) in the Microsoft Knowledge Base:

ARTICLE-ID: Q96268
TITLE : How Environment Variables Are Set in Windows NT

Additional reference words: 3.10
KBCategory: kbprg
KBSubcategory: BseMisc

Changes to the MSTest WFndWndC()

PSS ID Number: Q108227
Authored 07-Dec-1993 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Win32 SDK, versions 3.1 and 3.5

A change made to MS-TEST WFndWnd() and WFndWndC() may cause them not to work as they did previously. Specifically, treatment of the first parameter of WFndWnd() and WFndWndC() has changed.

Previously, "" and NULL resulted in the caption of the window being ignored. Now, "" means the window must have an empty caption and NULL means to ignore the caption altogether.

Additional reference words: 3.10 3.50
KBCategory: kbtool
KBSubcategory: TlsMisc

Changing a List Box from Single-Column to Multicolumn

PSS ID Number: Q68580
Authored 22-Jan-1991 Last modified 16-May-1995

-
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32, SDK versions 3.5, 3.51, and 4.0

-

SUMMARY

A list box cannot be changed from single column to multicolumn by altering the list box's style bits "on the fly."

The effect of switching a single-column list box to multicolumn can be achieved by creating one single column and one multicolumn list box. Initially, hide the multicolumn list box. To switch the list boxes, hide the single-column list box and show the multicolumn list box.

MORE INFORMATION

In general, programmatically changing the style bits of a window usually leads to unstable results. The method of switching between two windows (hiding one and showing the other) can safely switch window styles.

Additional reference words: 3.00 3.10 3.50 4.00 95 listbox
KBCategory: kbprg
KBSubcategory: UsrCtl

Changing How Pop-Up Menus Respond to Mouse Actions

PSS ID Number: Q65256
Authored 28-Aug-1990 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

The TrackPopupMenu function allows an application to receive input from a menu that is displayed anywhere within the application's client area. This article demonstrates how to change the menu's default behavior for mouse selections.

MORE INFORMATION

The default action for floating pop-up menus maintained with TrackPopupMenu is as follows:

1. If the menu is displayed in response to a keystroke, the pop-up menu is visible until the user selects a menu item or presses ESC.

2. If the menu is displayed in response to a WM_*BUTTONUP message, it acts as if it were displayed in response to a keystroke.

NOTE: In the context of this article, the * in the WM_*BUTTONUP and WM_*BUTTONDOWN messages can be L (left mouse button), M (middle mouse button), or R (right mouse button).

An application can change the behavior of a floating pop-up menu displayed in response to a WM_*BUTTONDOWN message to keep it visible after the mouse button is released. However, when an application uses the techniques described below, it changes the menu's user interface. Specifically, to change menu selections with the mouse, the user must first release the mouse button and then press it again. Dragging the mouse between items with the button down, without releasing the button at least once, will not change the selection.

To cause a floating pop-up menu to remain visible after it is displayed in response to a WM_*BUTTONDOWN message, follow these four steps:

1. In the application, allocate a 256 byte buffer to hold the key state.

2. Call the GetKeyboardState function with a far pointer to the buffer to retrieve the keyboard state.

3. Set the keyboard state for the VK_*BUTTON index in the keyboard

   state array to 0.

4. Call SetKeyboardState with a far pointer to the buffer to register
   the change with Windows.

The keyboard state array is 256 bytes. Each byte represents the state of a particular virtual key. The value 0 indicates that the key is up, and the value 1 indicates that the key is down. The array is indexed by the VK_ values listed in Appendix A of the "Microsoft Windows Software Development Kit Reference Volume 2" for version 3.0.

The code fragment below changes the state of the VK_LBUTTON to 0 (up) during the processing of a WM_LBUTTONDOWN message. This causes TrackPopupMenu to act as if the menu were displayed as the result of a WM_LBUTTONUP message or of a keystroke. Therefore, the menu remains visible even after the mouse button is released and the WM_LBUTTONUP message is received. Items on this menu can be selected with the mouse or the keyboard.

```
   switch (iMessage)
      {
   case WM_LBUTTONDOWN:
      static BYTE rgbKeyState[256];

      GetKeyboardState(rgbKeyState);
      rgbKeyState[VK_LBUTTON] = 0;      // 0==UP, 1==DOWN
      SetKeyboardState(rgbKeyState);

      // Create the pop-up menu and call TrackPopupMenu.
      break;
      }
```

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMen

Changing Hypertext Jump Color in Windows Help

PSS ID Number: Q75111
Authored 12-Aug-1991 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The author of a file to be used with Windows Help can alter the default color of hypertext jump strings. This specification overrides both the default behavior of showing these strings as underlined green text and any color preference listed in the WIN.INI file. This article explains the steps required to make this type of modification.

MORE INFORMATION

This feature must be used with care. There are situations when the user may require a jump color other than the default. The profile strings in the WIN.INI file provide an opportunity to specify a color value appropriate for the particular situation. For more information on these settings, query on the following words:

```
prod(winsdk) and jumpcolor
```

To specify the color of a context jump when the RTF Help text is created using Microsoft Word for Windows, follow the six steps listed below. This procedure is a modification of the steps listed on page 17-11 of the "Microsoft Windows Software Development Kit Tools" manual.

1. Place the cursor at the point in the text where the jump term will be entered.

2. Choose Character from the Format menu and specify the double-underline attribute and the desired text color. Word for Windows provides 6 colors, along with auto, black, and white.

3. Type the jump word or words.

4. Choose Character from the Format menu. Turn off the double-underline attribute and choose the default text color and hidden text.

5. Type a percent sign (%), followed by the context string assigned to the topic. For example, JumpText%ContextString, where JumpText is given the desired color and %ContextString is hidden text.

6. Choose Character from the Format menu. Turn off hidden text.

After all topics have been created, save the file as RTF (rich text format) and build the .HLP file. For more information on this process, refer to Chapters 15 through 19 of the "Windows Software Development Kit (SDK) Tools" manual.

It is also possible to modify the six color values that Word for Windows provides as defaults. This can be done by modifying the color table in the RTF header. To do this, load the RTF file into Word for Windows, however, do not convert the file from RTF. For more information on the RTF color table, refer to page 389 of the "Microsoft Word for Windows Technical Reference" (Microsoft Press).

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsHlp

Changing Print Settings Mid-Job

PSS ID Number: Q85679
Authored 17-Jun-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

In Windows 3.1 and later, print settings can be changed on a page-by-page basis through the ResetDC() API.

MORE INFORMATION

An application can pass a new DEVMODE structure (containing new print settings) to ResetDC() between pages to change the print settings. For example, this function makes it possible to change the paper bin or paper orientation for each page in a print job. Note that ResetDC() cannot be used to change the driver name, device name, or the output port.

Before calling ResetDC(), the application must ensure that all objects (other than stock objects) that were previously selected into the printer device context are selected out.

Additional reference words: 3.10 3.50 4.00 95 dmOrientation
dmDefaultSource
dmPaperSize hDC WM_DEVMODECHANGE ExtDeviceMode
KBCategory: kbprg
KBSubcategory: GdiPrn

Changing the Controls in a Common Dialog Box

PSS ID Number: Q82299
Authored 30-Mar-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, version 3.5

One reason to incorporate the common dialogs library routines into an application is the ability to use the basic functionality of one or more of the common dialogs and tailor it to the needs of a particular application.

All of the predefined controls must be present for the Common Dialogs DLL (COMMDLG.DLL) to properly interact with a dialog box. Each predefined control in the dialog box must retain its control ID value. For these reasons, an application cannot delete unnecessary controls from a dialog box.

To prevent the user from interacting with a given control, move the control off screen by specifying very large coordinate values [for example, (4000, 4000)]. The application must also disable the control to prevent it from receiving the focus when the user uses the TAB key to cycle through the controls. Failing to disable the control can create "mystery" tab stops where the input focus disappears.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: UsrCmnDlg

Changing the Font Used by Dialog Controls in Windows

PSS ID Number: Q74737
Authored 29-Jul-1991 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.1, 3.5, 3.51, and 4.0

In Windows 3.x, there are two ways to specify the font used by dialog controls:

1. The FONT statement can be used in the dialog template to specify the font used by ALL the controls in the dialog box.

   -or-

2. The WM_SETFONT message can be sent to one or more dialog controls during the processing of the WM_INITDIALOG message.

If a font is specified in the dialog template, the controls will use a bold version of that font. The following code demonstrates how to change the font used by dialog box controls to a nonbold font using WM_SETFONT. The font should be deleted with DeleteObject() when the dialog box is closed.

```
HWND hDlg;
HFONT hDlgFont;
LOGFONT lFont;

case WM_INITDIALOG:
    /* Get dialog font and create non-bold version */
    hDlgFont = NULL;
    if ((hDlgFont = (HFONT)SendMessage(hDlg, WM_GETFONT, 0, 0L))
             != NULL)
       {
       if (GetObject(hDlgFont, sizeof(LOGFONT), (LPSTR)&lFont)
                != NULL)
          {
          lFont.lfWeight = FW_NORMAL;
          if ((hDlgFont = CreateFontIndirect(&lFont)) != NULL)
             {
             SendDlgItemMessage(hDlg, CTR1, WM_SETFONT, hDlgFont, 0L);
             // Send WM_SETFONT message to desired controls
             }
          }
       }
    else  // user did not specify a font in the dialog template
       {  // must simulate system font
       lFont.lfHeight = 13;
       lFont.lfWidth = 0;
       lFont.lfEscapement = 0;
```

```
        lFont.lfOrientation = 0;
        lFont.lfWeight = 200; // non-bold font weight
        lFont.lfItalic = 0;
        lFont.lfUnderline = 0;
        lFont.lfStrikeOut = 0;
        lFont.lfCharSet = ANSI_CHARSET;
        lFont.lfOutPrecision = OUT_DEFAULT_PRECIS;
        lFont.lfClipPrecision = CLIP_DEFAULT_PRECIS;
        lFont.lfQuality = DEFAULT_QUALITY;
        lFont.lfPitchAndFamily = VARIABLE_PITCH | FF_SWISS;
        lFont.lfFaceName[0] = NULL;
        hDlgFont = CreateFontIndirect(&lFont);

        SendDlgItemMessage(hDlg, CTR1, WM_SETFONT, hDlgFont,
           (DWORD)TRUE);
        // Send WM_SETFONT message to desired controls
        }

   return TRUE;
   break;
```

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDlgs

Changing/Setting the Default Push Button in a Dialog Box

PSS ID Number: Q67655
Authored 08-Dec-1990 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The default push button in a dialog box is defined to be the button that is pressed when the user chooses the ENTER key, provided that the input focus is not on another button in the dialog box. The default push button is visually distinguished from other buttons by a thick dark border. This article describes how to change the default push button.

MORE INFORMATION

To change the default push button, perform the following three steps:

1. Send the BM_SETSTYLE message to the current default push button to change its border to that of a regular push button.

2. Send a DM_SETDEFID message to the dialog box to change the ID of the default push button.

3. Send the BM_SETSTYLE message to the new default push button to change its border to that of a default push button.

The following is sample code that performs the three steps:

Sample Code

```
   // Reset the current default push button to a regular button.
   SendDlgItemMessage(hDlg, <ID of current default push button>,
         BM_SETSTYLE, BS_PUSHBUTTON, (LONG)TRUE);

   // Update the default push button's control ID.
   SendMessage(hDlg, DM_SETDEFID, <ID of new default push button>,
         0L);

   // Set the new style.
   SendDlgItemMessage(hDlg, <ID of new default push button>,
         BM_SETSTYLE, BS_DEFPUSHBUTTON, (LONG)TRUE);
```

NOTE: For Win32, the (LONG) casts should be changed to (LPARAM).

Note, however, that ANY push button that has the input focus will have a dark border. A default push button will retain this dark border even when

the input focus is transferred to another control in the dialog box, provided the new control is not another push button.

For example, if the input focus is on an edit control, check box, radio button, or any control other than a push button, and the ENTER key is pressed, Windows sends a WM_COMMAND message to the dialog box procedure with the wParam set to the control ID of the default push button.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Character Sets Supported by Hangeul (Korean) Windows Versions

PSS ID Number: Q130055
Authored 10-May-1995 Last modified 16-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 4.0
- Microsoft Win32 Software Development Kit (SDK) version 3.5
- Microsoft Win32s version 1.2

SUMMARY

Hangeul (Korean) Windows version 3.1 supports the Wansung code set only. However Hangeul Windows 95 will support the XWansung code set.

MORE INFORMATION

Two different Hangeul character standards exist in Korea. One is Wansung, and the other is Johap. The Korean government issued a Korean standard code set (KSC5601-1987), which is Wansung code set. Hangeul Windows version 3.1 supports this character set only.

Later, the Korean government amended and added to the standard and issued a new one (KSC5601-1992), which contains both Wansung and Johap code sets.

Hangeul Windows version 3.1 and Hangeul Windows 95 will not support the KSC5601-1992 standard. Instead, Hangeul Windows 95 will support the XWansung (Microsoft Extended Wansung) standard, which is an extended version of the KSC5601-1987 standard.

The XWansung Code system contains some Johab characters (not all the Johap codes). Johab characters were added into the vacant Range of old Wansung Code of Hangeul Windows version 3.1. Here are the details on the XWansung code set:

Number of characters:

```
     Exisitng Wansung    = 8,836
     Additional Assigned = 8,822 (Johab)
     Additional Reserved = 4,770
     ------------------------------
     Total characters    = 22,428
```

```
Leading byte range:    0x81-0xFE
Trailing byte range:   0x41-5A,0x61-0x7A,0x81-0xFE
```

Additional reference words: 1.20 3.10 4.00 3.50 Hangul Chohap kbinf
KBCategory: kbother
KBSubcategory: wintldev

Checking for Administrators Group

PSS ID Number: Q111542
Authored 14-Feb-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

It is often useful to know if the account under which your application is running is a member of the "Administrators" group. The sample code below demonstrates a method of making this determination:

Sample Code

```
BOOL IsAdmin(void)
{
   HANDLE hProcess, hAccessToken;
   UCHAR InfoBuffer[1024];
   PTOKEN_GROUPS ptgGroups = (PTOKEN_GROUPS)InfoBuffer;
   DWORD dwInfoBufferSize;
   SID_NAME_USE snuInfo;
   UCHAR szAccountName[256], szDomainName[256];
   DWORD dwAccountNameSize, dwDomainNameSize;
   UINT x;

   hProcess = GetCurrentProcess();

   if(!OpenProcessToken(hProcess,TOKEN_READ,&hAccessToken))
      return(FALSE);

   if(!GetTokenInformation(hAccessToken,TokenGroups,InfoBuffer,
     1024, &dwInfoBufferSize)) return(FALSE);

   for(x=0;x<ptgGroups->GroupCount;x++)
   {
      dwAccountNameSize = 256;
      dwDomainNameSize = 256;

      LookupAccountSid(NULL, ptgGroups->Groups[x].Sid,
      szAccountName, &dwAccountNameSize,
      szDomainName, &dwDomainNameSize, &snuInfo);

      if(!strcmp(szAccountName,"Administrators") &&
      !strcmp(szDomainName,"BUILTIN"))
      return(TRUE);
  }
  return(FALSE);
}
```

MORE INFORMATION

The sample code above begins by obtaining a handle to the access token via the OpenProcessToken() API (application programming interface). The GetTokenInformation() API is then called to obtain the TOKEN_GROUPS structure for the access token. The sample then looks up the name of each group via the LookupAccountSid() API and compares it to Administrators. If a group with an account name Administrators and domain name "BUILTIN" is found, then the sample returns TRUE indicating the user is a member of the Administrators group.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

Choosing the Debugger That the System Will Spawn

PSS ID Number: Q103861
Authored 01-Sep-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

With Windows NT version 3.1, it is possible to have the system spawn a debugger whenever an application faults. The capability is controlled by the following Registry key:

```
HKEY_LOCAL_MACHINE\
          SOFTWARE\
         Microsoft\
        Windows NT\
    CurrentVersion\
           AeDebug
```

This key contains the following values:

```
Auto
Debugger
```

If the value of Auto is set to "0" (zero), then the system will generate a pop-up window, and if the user chooses Cancel, spawn the debugger that is specified in the Debugger value. If the value of Auto is set to "1", then the system will automatically spawn the debugger that is specified in the Debugger value.

After installing Windows NT 3.1, the Debugger value is set to DRWTSN32 -p %ld -e %ld -g and the Auto value is set to 1.

If the Win32 SDK is installed, then the Debugger value is changed to <MSTOOLS>\BIN\WINDBG -p %ld -e %ld and the Auto value is set to 0.

MORE INFORMATION

The DRWTSN32 debugger is a post-mortem debugger similar in functionality to the Windows 3.1 Dr. Watson program. DRWTSN32 generates a log file containing fault information about the offending application. The following data is generated in the DRWTSN32.LOG file:

- Exception information (exception number and name)
- System information (machine name, user name, OS version, and so forth
- Task list
- State dump for each thread (register dump, disassembly, stack walk, symbol table)

A record of each application error is recorded in the application event log. The application error data for each crash is stored in a log file named DRWTSN32.LOG, which by default is placed in your Windows directory.

Additional reference words: 3.10 3.50
KBCategory: kbtool
KBSubcategory: TlsMisc

Clarification of COMMPROP dwMax?xQueue Members

PSS ID Number: Q94950
Authored 26-Jan-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The entry for the COMMPROP structure in the Win32 Programmer's Reference states that for the dwMaxTxQueue and dwMaxRxQueue members, "a value of 0 means that this field is not used".

MORE INFORMATION

This statement means that the provider does not restrict you to maximum Rx and Tx queue values, and these members [returned by GetCommProperties()] should not be used to determine the size of your transmit and receive buffers when calling SetupComm().

Based on the memory present in the system, the Windows NT serial driver determines a default Rx queue size (currently 128 bytes on low memory systems and 4K on high memory systems). The current Rx and Tx queue sizes are located in the dwCurrentTxQueue and dwCurrentRxQueue members.

SetupComm() allows you to change these default queue sizes. However, you should not assume that the given serial driver will allocate any memory. The queue size allocated is stored in the dwCurrentRxQueue member of the COMMPROP structure. You may use this information to set the XonLim and XoffLim members of the device control block (DCB) structure.

The Microsoft-supplied serial driver attempts to allocate at least the amount requested for the RXQUEUE and, failing this, the request will also fail. The driver never attempts to allocate memory for the TXQUEUE.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseCommapi

Clarification of SearchPath() Return Value

PSS ID Number: Q115826
Authored 05-Jun-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, version 3.1

SUMMARY

The entry for "SearchPath()" in the "Win32 Programmer's Reference" says that the return value will be one of the following:

- The length (in characters) of the string copied to the buffer.

  -or-

- 0 if the function fails [see "GetLastError()" in the "Win32 Programmer's Reference" for more information].

If the specified file is not found, then the return value is 0, but the value retrieved by GetLastError() is not changed.

In order to distinguish the "file not found" condition from another error (out of memory, invalid parameter, and so forth), call SetLastError() with a value of NO_ERROR before calling SearchPath().

MORE INFORMATION

This behavior is by design. The recommended way to check the return status for SearchPath() is:

```
return value >  buffer length    buffer too small
return value =  0                file not found or another error
return value <= buffer length    file found
```

Handle the case where the return value is 0 as follows:

```
TCHAR  szFilename[] = "MyFile.Txt";
TCHAR  szPathname[MAX_PATH];
LPTSTR lpszFilename;

SetLastError( NO_ERROR );
if( !SearchPath( NULL, szFilename, NULL, MAX_PATH, szPathname,
                 &lpszFilename ) )
{
   if( GetLastError() == NO_ERROR )
      Display( "File not found." );
   else
      Display( "SearchPath failed!" );
```

```
   }
```

Additional reference words: 3.10
KBCategory: kbprg
KBSubcategory: BseFileio

Clarification of the "Country" Setting

PSS ID Number: Q102765
Authored 10-Aug-1993 Last modified 05-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

Under Windows NT, the "Country" choice affects currency, date/time and number format, and so forth. The "Language" choice affects sorting, names of the days of the weeks and months, and so forth. These settings allow the user to choose the appropriate language and country format. For example, if you are British and living in the U.S., you can pick a locale of English (British) at setup time, then use Control Panel later to change your country to U.S. so that currency is in dollars instead of pounds.

In Windows 95, there is not both a Country and a Language choice. You are asked for a single Regional Setting.

Additional reference words: 3.10 3.50 4.00
KBCategory: kbprg
KBSubcategory: UsrNls WintlDev

Clearing a Message Box

PSS ID Number: Q74444
Authored 18-Jul-1991 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

During the processing of the MessageBox() function, Windows creates a bitmap to save the part of the screen covered by the message box. Normally, before the MessageBox() function returns, Windows repaints the portion of the screen covered by the message box using the bitmap. In this scenario, when the user clicks on a button to dismiss the message box, the message box disappears immediately.

It is important to note that under low memory conditions, Windows will discard the bitmap. If the bitmap is discarded and a significant amount of processing takes place between the MessageBox() call and painting the application's window, the vestigial image of the message box will remain on the screen during the processing. If the user clicks on this image with the mouse, the underlying window will receive the mouse messages. This can cause unexpected (and possibly undesirable) effects.

To address this problem, call UpdateWindow() immediately after MessageBox(). The parameter to UpdateWindow() should be the parent window of the message box (or of the application's main window if the message box has no parent). This will cause the application to paint the affected window if the bitmap has been discarded. The message box will disappear immediately under all circumstances.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDlgs

Client Service For Novell Netware Doesn't Support Named Pipes

PSS ID Number: Q129317
Authored 24-Apr-1995 Last modified 04-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

The Windows NT Client Service for Novell NetWare does not support named pipes. Consequently, 16-bit Novell NetWare client applications running on Windows NT cannot connect to Novell NetWare named pipe server applications. There are currently no plans to add named pipe support to the Windows NT Client Service for Novell NetWare.

For more information on Novell NetWare named pipes, consult the following Novell documentation:

- NetWare Client for OS/2 User's Guide
- NetWare Client for MS-DOS and Microsoft Windows User's Guide

Additional reference words: 3.10 3.50
KBCategory: kbnetwork
KBSubcategory: NtwkMisc

Clipboard Memory Sharing in Windows

PSS ID Number: Q11654
Authored 26-Oct-1987 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

The following are questions and answers on the topic of Clipboard memory sharing:

Q. Does the Clipboard UNLOCK before freeing the handle when I tell it to SetClipboardData()?

A. Yes, the Clipboard UNLOCKs before freeing the handle when you SetClipboardData().

Q. Does the Clipboard actually copy my global storage to another block, or does it just retain the value of my handle for referencing my block?

A. The Clipboard is sharable; it retains the value of the handle.

Q. Does GetClipboardData() remove the data from the Clipboard, or does it allow me to reference the data without removing it from the Clipboard?

A. The data handle returned by GetClipboardData() is controlled by the Clipboard, not by the application. The application should copy the data immediately, instead of relying on the data handle for long-term use. The application should not free the data handle or leave it locked. To remove data from the Clipboard, call SetClipboardData().

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrClp

ClipCursor() Requires WINSTA_WRITEATTRIBUTES

PSS ID Number: Q106384
Authored 07-Nov-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SUMMARY

The documentation for ClipCursor() states that the calling process must have WINSTA_WRITEATTRIBUTES access to the window station. However, this permission does not have to be enabled programmatically because the system always grants the Local Login SID (the interactive user) WINSTA_WRITEATTRIBUTES access, regardless of the permission groups to which the user belongs.

MORE INFORMATION

The following code shows how to confine the cursor to the application window during WM_ACTIVATE processing. Note that the clip cursor region must be restored to its previous state each time the application deactivates.

Sample Code

```
   static RECT rcOldClip;
    .
    .
    .

      case WM_ACTIVATE:{
         short fActive = LOWORD( wParam );

         if( fActive ){
            RECT rcNewClip;

            /* Record the area in which the cursor can move. */
            GetClipCursor( &rcOldClip );

            /* Get the dimensions of the application's client area. */
            GetClientRect( hWnd, &rcNewClip);

            /* Convert to screen coordinates. */
            MapWindowPoints( hWnd, NULL, (LPPOINT)&rcNewClip, 2 );

            /* Confine the cursor to the application's window. */
            ClipCursor( &rcNewClip );
        }
        else{
```

```
            /* Restore the cursor to its previous area. */
            ClipCursor( &rcOldClip );
        }
        break;
    }
```

Additional reference words: 3.10
KBCategory: kbprg
KBSubcategory: GdiCurico

Combo Box w/Edit Control & Owner-Draw Style Incompatible

PSS ID Number: Q82078
Authored 25-Mar-1992 Last modified 16-May-1995

-
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

-

SUMMARY

The owner-draw combo box styles (CBS_OWNERDRAWFIXED and CBS_OWNERDRAWVARIABLE) are incompatible with the combo box styles that contain an edit control (combo box styles CBS_SIMPLE and CBS_DROPDOWN). A combo box with either the CBS_SIMPLE or CBS_DROPDOWN style displays the currently selected item in its associated edit control. When an owner-draw style is specified for the combo box style CBS_SIMPLE or CBS_DROPDOWN, the current selection may not be displayed. Using the SetWindowText function to display the current selection in response to a CBN_SELCHANGE message may not be effective.

MORE INFORMATION

An owner-draw combo box can contain bitmaps or other graphic elements in its list box. Therefore, to correctly display the current selection, it is necessary to display a bitmap or other graphic element in the edit control. Because edit controls are not designed to display graphics, there is no natural method to display the current selection in an owner-draw combo box with an edit control.

The combo box style CBS_DROPDOWNLIST, which has a static text area instead of an edit control, can display any item, including graphics. Use this style combo box with the owner-draw styles.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

COMCTL32 APIs Unsupported in the Win32 SDK

PSS ID Number: Q105300
Authored 17-Oct-1993 Last modified 05-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1 and 3.5
- Microsoft Win32s version 1.2

The library COMCTL32.LIB was included in the Win32 SDK because the PERFMON sample makes use of it. The library is part of the Windows for Workgroups (WFW) COMMCTRL.LIB. However, Microsoft does not officially support COMCTL32.LIB or recommend the use of these application programming interfaces (APIs), and therefore they have not been documented in the SDK.

Microsoft does not recommend using these APIs, because Microsoft is providing the new controls in Windows 95, Windows NT 3.51, and Win32s 1.3.

If you must absolutely use COMCTL32 with earlier versions, the documentation can be found in the WFW SDK. Be aware that these APIs are unsupported, and code that you write will not work on Windows 95 and Windows NT 3.51.

Additional reference words: 1.20 3.10 3.50
KBCategory: kbprg
KBSubcategory: UsrCtl W32s

Common Dialog Boxes and the WM_INITDIALOG Message

PSS ID Number: Q74610
Authored 24-Jul-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, version 3.5

SUMMARY

An application using the common dialog box library (COMMDLG.DLL) can override any information initialized in the DLL by handling the WM_INITDIALOG message in its dialog hook function. If the application is using a private dialog template, it should also initialize all private dialog items while handling this message.

After processing the WM_INITDIALOG message, the hook function should return FALSE if it has set the focus to a dialog control, and return TRUE if Windows should set the focus.

MORE INFORMATION

For example, consider an application that is using the Open File common dialog box (via GetOpenFileName()) but does not want the Drives combo box to appear in the dialog box. Since all dialog items in the standard dialog template must be included in the application's private dialog template, the application will need to include code to disable and hide the Drives combo box and the corresponding "Drives:" static text control. This code would be implemented in the WM_INITDIALOG case of the dialog hook function, as follows:

```
   case WM_INITDIALOG:
      hWnd = GetDlgItem( hDlg, cmb2 );  // Get Drives combo box handle
      EnableWindow( hWnd, FALSE );      // No longer receives input,
                                        // no longer a tabstop
      SetWindowPos( hWnd, NULL, 0, 0, 0, 0, SWP_HIDEWINDOW );

      hWnd = GetDlgItem( hDlg, stc4 );  // Get "Drives:" static control
      EnableWindow( hWnd, FALSE );      // no longer an accelerator
      SetWindowPos( hWnd, NULL, 0, 0, 0, 0, SWP_HIDEWINDOW);

   // Initialize private dialog items here...

      return( TRUE );                   // Let Windows set the focus
```

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: UsrCmnDlg

Common File Mapping Problems and Platform Differences

PSS ID Number: Q125713
Authored 02-Feb-1995 Last modified 27-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.5, 3.51, and 4.0
- Microsoft Win32s, version 1.3

SUMMARY

This article addresses some common problems encountered when using file mapping. It also points out some platform differences in the file mapping implementation.

This article does not describe the procedures for performing file mapping. For information on using file mapping, please see the File Mapping overview in the Microsoft Win32 Programmer's Reference. Also see the descriptions for CreateFileMapping(), OpenFileMapping(), MapViewOfFile(), MapViewOfFileEx(), UnmapViewOfFile(), and FlushViewOfFile().

MORE INFORMATION

Name Space Conflicts

The names of event, semaphore, mutex, and file-mapping objects share the same name space, so it is not possible to have two different object types with the same name. It is an error to attempt to create or open an object of one type using a name that is already being used by an object of another type.

CreateFileMapping() and OpenFileMapping() will fail if they specify an object name that is in use by an object of another type. In both cases, GetLastError() will return ERROR_INVALID_HANDLE (6).

To avoid conflicts between object types, one solution is to include the object type in the name. For example, use "EV_myapp_block_ready" for an event object name and "FM_myapp_missile_data" for a file mapping object name.

Necessity of Unmapping All Views of a Mapped File

Windows maintains an internal handle to a file mapping object for each view of that object, whether created by MapViewOfFile() or MapViewOfFileEx(). This internal handle is kept in addition to the handle returned by CreateFileMapping(). The internal handle is not closed until the view associated with the handle is unmapped by calling UnmapViewOfFile(). To completely close a file mapping object requires that

all handles for the object, including internal handles, be closed. Thus, to close a file mapping object, all views of that object must be unmapped, and the handle returned by CreateFileMapping() must be closed.

Extant unmapped views of a file mapping object will NOT cause a CloseHandle() on the object's handle to fail. In other words, when your handle to the object is closed successfully, it is not necessarily true that all views have been unmapped, so the file mapping object has not necessarily been freed.

Failure to properly unmap all views of the object and to close the handle to the object will cause leaks in the application's paged pool, nonpaged pool, virtual bytes, and also in the system wide committed bytes.

Restrictions on the Size of File Mapping Objects

The size of a file mapping object backed by the system paging file is limited to available system virtual memory (meaning the amount of memory that could be committed with a call to VirtualAlloc()).

On Windows NT, the size of a file mapping object backed by a named disk file is limited by available disk space. The size of a mapped view of an object is limited to the largest contiguous block of unreserved virtual memory in the process performing the mapping (at most, 2GB minus the virtual memory already reserved by the process).

On Win32s, the size of a file mapping object backed by a named disk file is limited to available system virtual memory, due to the virtual memory management implementation of Win32s. Win32s allocates regular virtual memory for the memory mapped section even though it does not need swap space, and the amount of VM set by Windows is too small to use for mapping large files. As with Windows NT, available disk space will also impose a limitation.

On Windows 95, the size of a file mapping object backed by a named disk file is limited to available disk space. The size of a mapped view of an object is limited to the largest contiguous block of unreserved virtual memory in the shared virtual arena. This block will be at most 1GB, minus any memory in use by other components of Windows 95 which use the shared virtual arena (such as 16-bit Windows-based applications). Each mapped view will use memory from this arena, so this limit applies to the total size of all non-overlapping mapped views for all applications running on the system.

Mapped File May Not be Automatically Grown

If the size specified for a file mapping object backed by a named disk file in a call to CreateFileMapping() is larger than the size of the file used to back the mapping, the file will normally be grown to the specified size by the CreateFileMapping() call.

On Windows NT only, if PAGE_WRITECOPY is specified for the fdwProtect parameter, the file will not automatically be grown. This will cause

CreateFileMapping() to fail, and GetLastError() will return ERROR_NOT_ENOUGH_MEMORY (8). To set the size of the file before calling CreateFileMapping(), use SetFilePointer() and SetEndOfFile().

MapViewOfFileEx() and Valid Range of lpvBase

On Windows NT, views of file mapping objects are mapped in the address range of 0-2 GB. Passing an address outside of this range as the lpvBase parameter to MapViewOfFileEx() will cause it to fail, and GetLastError() will return ERROR_INVALID_PARAMETER (87).

On Windows 95, views of file mapping objects are mapped in the address range of 2-3 GB (the shared virtual arena). Passing an address outside of this range will cause MapViewOfFileEx() to fail, and GetLastError() will return ERROR_INVALID_ADDRESS (487). Note that future updates to Windows 95 may change the mapping range to 0-2 GB, as on Windows NT.

MapViewOfFileEx() and Allocation Status of lpvBase

If an address is specified for the lpvBase parameter of MapViewOfFileEx(), and there is not a block of unreserved virtual address space at that address large enough to satisfy the number of bytes specified in the cbMap parameter, then MapViewOfFileEx() will fail, and GetLastError() will return ERROR_NOT_ENOUGH_MEMORY (8). This does not mean that the system is low on memory or that the process cannot allocate more memory. It simply means that the virtual address range requested has already been reserved in that process.

Prior to calling MapViewOfFileEx(), VirtualQuery() can be used to determine an appropriate range of unreserved virtual address space.

MapViewOfFileEx() and Granularity of lpvBase

For the lpvBase parameter specified in a call to MapViewOfFileEx(), you should use an integral multiple of the system's allocation granularity. On Windows NT, not specifying such a value will cause MapViewOfFileEx() to fail, and GetLastError() to return ERROR_MAPPED_ALIGNMENT (1132). On Windows 95, the address is rounded down to the nearest integral multiple of the system's allocation granularity.

To determine the system's allocation granularity, call GetSystemInfo().

Addresses of Mapped Views

When mapping a view of a file (or shared memory), it is possible to either let the operating system determine the address of the view, or to specify an address as the lpvBase parameter of the MapViewOfFileEx() function. If the file mapping is going to be shared among multiple processes, then the recommended method is to use MapViewOfFile() and let the operating system select the mapping address for you. There are good reasons for doing so:

- On Windows NT, views are mapped independently into each process's address space. While it may be convenient to try to map the view at the same address in each process, the specified virtual address range may not be free in all of the processes involved. Therefore, the mapping could fail in one (or more) of the processes trying to share the file mapping.

- On Windows 95, file mapping objects exist in the 2-3 GB address range (the shared virtual arena). Therefore, once the initial address for the view is determined, additional views of the mapping will be mapped to the same address in each process anyway, and there is no benefit in trying to force the initial mapping to a specific address. For the second and subsequent views of a mapping object, if the address specified for lpvBase does not match the actual address where Windows 95 has mapped the view, then MapViewOfFileEx() fails, and GetLastError() returns ERROR_INVALID_ADDRESS (487). Additionally, when attempting to map the first view at a pre-determined address, that address may already be in use by other components of Windows 95 which use the shared virtual arena. Note that future updates to Windows 95 may change the mapping range to 0-2 GB, as on Windows NT.

If it is absolutely necessary to create the mappings at the same address in multiple processes under Windows NT, here are two possible approaches:

1. Pick an appropriate address and manage the virtual address space so that this address is left available. This means basing your DLLs, allocating memory at specific locations, and using a tool such as Process Walker to observe the virtual address space pattern. As soon as possible in the execution of the application, either reserve the desired address space or perform the mapping. One good place to do this is in the PROCESS_ATTACH handling in a DLL, because it is called before the executable itself is started. NOTE: There is still no guarantee that some DLL will not have already loaded at the address in question. If not all involved processes can map at the predetermined address, they can either fail or try a new address.

   -or-

2. Have all processes involved negotiate an appropriate address. The processes can all use the VirtualQuery() function to scan their address spaces until a common address is found in each process that has a large enough unreserved block. This requires that all processes involved map the address at the same time. A process that starts after the address has been determined must map at that address, and fail if it cannot do so. Alternatively, the negotiation process could be repeated, with each process remapping at the new address. Then, all pointers into the mapping must be readjusted.

The second method is far more likely to succeed. It can also be combined with the first to make it more likely that an appropriate address will be found quickly.

When views are mapped to different addresses under Windows NT, the difficulty that arises is storing pointers to the mapping within the

mapping itself. This is because a pointer in one process does not point to the same location within the mapping in another process. To overcome this problem, store offsets rather than pointers in the mapping, and calculate actual addresses in each process by adding the base address of the mapping to the offset. It is also possible to used based pointers and thus perform the base + offset conversion implicitly. A short SDK sample called BPOINTER demonstrates this technique.

Additional Platform Differences

Additional limitations when performing file mapping under Windows 95:

1. The dwOffsetHigh parameters of MapViewOfFile() and MapViewOfFileEx() are not used, and should be zero. Windows 95 uses a 32-bit file system.

2. The dwMaximumSizeHigh parameter of CreateFileMapping() is not used, and should be zero. Again, this is due to the 32-bit file system.

3. The SEC_IMAGE and SEC_NOCACHE flags for the fdwProtect parameter of CreateFileMapping() are not supported.

4. If the FILE_MAP_COPY flag is used to map a view of a file mapping object, the object must have been created using PAGE_WRITECOPY protection. Additionally, the object must be backed by a named file rather than the system paging file (in other words, a valid file handle, not (HANDLE)0xFFFFFFFF, must be specified for the hFile parameter of CreateFileMapping()). Failure to do either of these causes MapViewOfFile() to fail, and GetLastError() to return ERROR_INVALID_PARAMETER (87).

5. If two or more processes map a PAGE_WRITECOPY view of the same file mapping object (by using a named object, for example), they are able to see changes made to the view by the other process(es). The actual disk file is not modified, however. Under Windows NT, if one process writes to the view, it receives its own copy of the modified pages and will not affect the pages in the other process(es) or the disk file.

Additional reference words: 1.30 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseMm

Complete Enumeration of System Fonts

PSS ID Number: Q99672
Authored 03-Jun-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

Complete enumeration of system fonts is a two-phase process. Applications should first call EnumFontfamilies with NULL as the family name to enumerate all the font face names in the system. Applications should then take each face name and call EnumFontFamilies again to obtain the style names (for TrueType fonts only) or the supported point sizes (for raster fonts only). The style names are not supported for the raster and vector fonts. Because TrueType and vector fonts are continuously scalable, their point sizes are not enumerated.

MORE INFORMATION

The following steps detail the enumeration:

1. Call EnumFontFamilies with NULL as the family name (lpszFamily) to list one font from each available font family.

2. In the EnumFontFamProc callback function, look at the nFontType parameter.

3. If nFontType has the TRUETYPE_FONTTYPE flag set, then call EnumFontFamilies with the family name set to the font's type face name (lfFaceName of the ENUMLOGFONT structure). The callback function is called once for each style name. This enumeration is useful if the application is interested in finding a TrueType font with a specific style name (such as "Outline"). Because a TrueType font is continuously scalable, it is not necessary to enumerate a given font for point sizes. An application may use any desired point size. If the application is listing the enumerated TrueType fonts, it can simply choose some representative point sizes in a given range. The point sizes recommended by "The Windows Interface: An Application Design Guide" (page 159, Section 8.4.1.4) are 8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, 26, 28, 36, 48, and 72. All TrueType fonts are available on both displays and printers, so an application can be sure that the font appears the same on the display and the printer.

4. If nFontType has the RASTER_FONTTYPE flag set, then call EnumFontFamilies with the family name set to the font's type face name. The callback function is called once for each available point size. Raster fonts can be scaled only in multiples of the available

point sizes. Because scaled raster fonts are usually not appealing to the user, applications may choose to limit themselves to the available sizes. Because Microsoft Windows version 3.1 does not define style names for raster fonts, there is no need to enumerate for style names.

If the nFontType also has the DEVICE_FONTTYPE flag set, then the current font is a raster font available to the printer driver for use with the printer. The printer may have these fonts in hardware or be capable of downloading them when necessary. Applications that use such fonts should be aware that similar raster fonts may not be available on the display device. The converse is also true. If the DEVICE_FONTTYPE flag is not set, then applications should be aware that a similar font may not be available on the printer. Fonts generated by font packages such as Adobe Type Manager (ATM) are listed as device fonts.

5. If nFontType has neither the TRUETYPE_FONTTYPE nor the RASTER_FONTTYPE flags set, then the enumerated font is a vector font. Vector fonts are also continuously scalable so they do not have to be enumerated for point sizes. Because Windows 3.1 does not support style names for vector fonts, there is no need to enumerate them for style names. Vector fonts are generally used by devices such as plotters that cannot support raster fonts. These fonts generally have a poor appearance on raster devices, so many applications avoid them.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiFnt

Computing the Size of a New ACL

PSS ID Number: Q102103
Authored 28-Jul-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

When adding an access-allowed access control entry (ACE) to a discretionary access control list (DACL), it is useful to know the exact size needed for the new DACL. This is particularly useful when creating a new DACL and copying over the existing ACEs. The below code computes the size needed for a DACL if an access-allowed ACE is added:

```
ACL_SIZE_INFORMATION AclInfo;

GetAclInformation(pACL,&AclInfo,sizeof(ACL_SIZE_INFORMATION),
                  AclSizeInformation))

dwNewACLSize = AclInfo.AclBytesInUse +
               sizeof(ACCESS_ALLOWED_ACE) +
               GetLengthSid(UserSID) - sizeof(DWORD);
```

MORE INFORMATION

The call to GetAclInformation() takes a pointer to an ACL. This point is supplied by your program and should point to the DACL you want to add an access-allowed ACE to. The GetAclInformation() call fills out a ACL_SIZE_INFORMATION structure, which provides size information on the ACL.

The second statement computes what the new size of the ACL will be if an access-allowed ACE is added. This is accomplished by adding the current bytes being used to the size of an ACCESS_ALLOWED_ACE. We then add the size of the security identifier (SID) (provided by your application) that is to used in the AddAccessAllowedAce() API call. Subtracting out the size of a DWORD is the final adjustment needed to obtain the exact size. This adjust is to compensate for a place holder member in the ACCESS_ALLOWED_ACE structure which is used in variable length ACEs.

When adding an ACE to an existing ACL, often there is not enough free space in the ACL to accommodate the additional ACE. In this situation, it is necessary to allocate a new ACL and copy over the existing ACEs and then add the access-allowed ACE. The above code can be used to determine the amount of memory to allocate for the new ACL.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

Conditionally Activating a Button in Windows Help

PSS ID Number: Q76534
Authored 26-Sep-1991 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

An application can add buttons to the Windows Help engine button bar. There may be times when the button should be activated or deactivated based on an external condition.

For example, if an application adds a tutorial button to the bar and the user has not chosen to install the tutorial, the button should be dimmed to indicate that the tutorial is not available.

MORE INFORMATION

The following code fragment demonstrates activating and deactivating buttons using the macro facility:

```
char szMacro[255];
    .
    .
    .
/* Bring up the Help engine with the HLP file   */
/* This code fragment assumes that a button has */
/* been defined with an ID of TUTORIAL_BUTTON   */
WinHelp (hWnd, lpHelpFile, HELP_CONTENTS, 0L)

if (fTutorial)
/* If the tutorial is installed, the macro should enable the button */
    lstrcpy(szMacro,"EnableButton('TUTORIAL_BUTTON')");
else
/* If the tutorial is not installed, the macro should disable the
   button */
    lstrcpy(szMacro,"DisableButton('TUTORIAL_BUTTON')");

/* Run the appropriate macro */
WinHelp (hWnd, lpHelpFile, HELP_COMMAND, (LONG)szMacro);
```

Additional reference words: 3.10 3.50 4.00 95 grayed out disabled unavailable
KBCategory: kbtool
KBSubcategory: TlsHlp

Considerations for CreateCursor() and CreateIcon()

PSS ID Number: Q73667
Authored 02-Jul-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

An application can use the CreateCursor() and CreateIcon() APIs to create icons and cursors on the fly. The application determines the shape at run time.

When the shape of the icons and the cursors is known in advance, an application should use LoadIcon() and LoadCursor().

An application that uses CreateIcon() must call DestroyIcon() to free the memory used by the icon when it is no longer needed. An application that uses CreateCursor() must call DestroyCursor() to release the memory used by the cursor when it is no longer needed.

An application can call DestroyIcon() and DestroyCursor() only when the icon or the cursor is not in use. For example, if the cursor created by CreateCursor() has been specified in a SetCursor call, it must not be destroyed until it has been released by another SetCursor() call.

An application can only use DestroyIcon() and DestroyCursor() to destroy icons and cursors created by CreateIcon() and CreateCursor(). It should not try to destroy icons and cursors loaded with LoadIcon() and LoadCursor().

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiCurico

Consoles Do Not Support ANSI Escape Sequences

PSS ID Number: Q84240
Authored 05-May-1992 Last modified 18-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

Windows NT does not support ANSI escape sequences. There is some functionality that this affects (for example, changing the color of the prompt). This also affects a very limited number TTY-type programs that rely on the console for escape support to be provided.

This feature is is under review and is being considered for future releases.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseCon

Controlling the Caret Color

PSS ID Number: Q84054
Authored 29-Apr-1992 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

When an application creates a custom caret using a bitmap, it is possible to specify white or black for the caret color. In the case of Windows running on a monochrome display, the application can cause the caret to be the color of the display (white, green, amber, and so forth, as appropriate). However, because the caret color is determined by Windows at run time based on the hardware installed, the application cannot guarantee what color will be used under all circumstances. This article provides information about using color in a custom caret.

MORE INFORMATION

To create a caret, first create a bitmap with the desired pattern. To display the caret, Windows exclusive-ORs (XORs) and takes the opposite of the result (the NOT of the result) of the bitmap with the background of the client window. Therefore, to create a white caret, create a bitmap that when XOR'd with the window background will have an opposite value that will create a white color. It is when the caret blinks that Windows uses the reverse of the bitmap XOR'd with the background to draw the caret; this creates the white blink seen on the screen.

The bitmap for the caret cannot use a color palette. Windows does not use the color values from the palette in its calculations but the indices into the palette. While it is possible to use palette indices successfully, perfect symmetry of the colors in the palette is required. This is unlikely. For each color in the palette, its exact opposite color must be in the palette, in the exactly opposite index position.

However, when the application creates bitmaps itself, it has complete control over the bits. Therefore, the application can create the perfect counterpart that corresponds to the window background color. If the application uses this information to create the caret bitmap, when Windows creates the caret, it can choose the closest color available in the system palette.

Therefore, to create a white caret (or a black one, if the screen has too many light elements), the task is straightforward. Windows always reserves a few colors in the system palette and makes them available

to all applications. On a color display, these colors include black and white. On a monochrome display, these colors are whatever the monochrome color elements are.

Because black and white (or the monochrome screen colors) are always available, the application simply creates a bitmap that, when XOR'd with the screen background color, produces black or white.
The technique involves one main principle: background XOR background = FALSE. Anything XOR'd with itself returns FALSE, which in bitmap terms maps to the color black.

The process of creating a caret from the background color involves the four steps discussed below:

1. Create a pattern brush the same as the window background

2. Select the pattern brush into a memory display context (DC)

3. Use the PATCOPY option of the PatBlt function to copy the brush pattern into the caret bitmap.

4. Specify the caret bitmap in a call to the CreateCaret function.

When this caret is XOR'd with the background, black will result. When the caret blinks, and is therefore displayed, Windows computes the opposite of the caret and XOR's this value into the background. This yields NOT(background XOR background) = NOT(FALSE) = TRUE which corresponds to WHITE. The first background represents the caret bitmap and the second is the current background color of the window.

Note that half the time the custom bitmap is displayed (when the caret "blinks") the other half of the time the background is displayed, (between "blinks").

If the background color is light gray or lighter [RGB values (128, 128, 128) through (255, 255, 255)], then a black caret is usually desired. The process of creating a black caret is just as straightforward. Modify step 1 of the process given above to substitute the inverse of the background for the background bitmap. When the caret blinks, it will show black. The equation that corresponds to this case is NOT(inverse of background XOR background) = NOT(TRUE) = FALSE which corresponds to BLACK.

To change the caret color to something other than black or white requires considerably more work, with much less reliable results because the application must solve the following equation:

```
NOT(caret XOR background) = desired_color on the
                            "blink" of the caret.
```

where the value for the caret color must be determined given the desired color. A series of raster operations is required to solve this type of equation. (For more information on raster operations, see chapter 11 of the "Microsoft Windows [3.0] Software Development Kit Reference, Volume 2" or pages 573-585 of the "Microsoft Windows [3.1]

Software Development Kit Programmer's Reference, Volume 3: Messages, Structures, and Macros.")

Even after solving this equation, the color actually displayed is controlled by Windows and the colors in the current system palette. With colors other than black or white, an exact match for the desired color may not be available. In that case, Windows will provide the closest match possible. Because the palette is a dynamic entity and can be modified at will, it is impossible to guarantee a particular result color in all cases. The colors black and white should be safe most of the time because it is quite unusual for an application to modify the reserved system colors. Even when an application does change the system palette, it most likely retains a true black and a true white.

As long as a black and white remain in the palette (which is usually the case), this algorithm will provide a white or black caret.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCrt

Converting a Linear Address to a Flat Offset on Win32s

PSS ID Number: Q115080
Authored 18-May-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s, versions 1.1, 1.15, and 1.2

SUMMARY

Win32s flat code and data selectors are not zero-based. Linear addresses retrieved through a VxD can be used in a Win32-based application running under Win32s, after one small change is made.

In addition, there are two Universal Thunk APIs that are used to convert segmented addresses to flat addresses and vice versa.

MORE INFORMATION

Linear Address to Flat Address

Win32s does not base linear addresses at 0, so that exceptions will be generated when null pointers are dereferenced. Therefore, an access violation occurs when:

1. a 16-bit DLL calls a VxD to retrieve a linear address (the VxD got the address by translating a physical address to a linear address) through DPMI function 0800h (map physical to linear).

2. the 16-bit DLL returns the address to a Win32-based application through the Universal Thunk.

3. the Win32-based application uses this linear address.

In order to convert a linear address (based at 0) to a flat offset, add the base to the linear address. To do this, get the offset through GetThreadSelectorEntry() with the DS or CS and then subtract that base from the linear address that was returned by the VxD.

Segmented Address to Flat Address

The following Win32s Universal Thunk APIs are used for address translation:

- UTSelectorOffsetToLinear (segmented address to flat address)

- UTLinearToSelectorOffset (flat address to segmented address)

NOTE: In the nested function call

```
   UTLinearToSelectorOffset( UTSelectorOffsetToLinear( x ) );
```

where x is a segmented address, you may not necessarily get the original value of x back. It is by design that the sel:off pair may be different. If the memory was allocated by a 16-bit application, Win32s does not have x in its LinearAddress->selector translation tables. Therefore, when UTLinearToSelectorOffset() is called, new selectors are created.

Additional reference words: 1.10 1.20 gpf gp-fault
KBCategory: kbprg
KBSubcategory: W32s

Converting Colors Between RGB and HLS (HBS)

PSS ID Number: Q29240
Authored 26-Apr-1988 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows, versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The code fragment below converts colors between RGB (Red, Green, Blue) and HLS/HBS (Hue, Lightness, Saturation/Hue, Brightness, Saturation).

MORE INFORMATION

```
/* Color Conversion Routines --

RGBtoHLS() takes a DWORD RGB value, translates it to HLS, and
stores the results in the global vars H, L, and S. HLStoRGB takes the
current values of H, L, and S and returns the equivalent value in an
RGB DWORD. The vars H, L, and S are only written to by:

   1. RGBtoHLS (initialization)
   2. The scroll bar handlers

A point of reference for the algorithms is Foley and Van Dam,
"Fundamentals of Interactive Computer Graphics," Pages 618-19. Their
algorithm is in floating point. CHART implements a less general
(hardwired ranges) integral algorithm.

There are potential round-off errors throughout this sample.
((0.5 + x)/y) without floating point is phrased ((x + (y/2))/y),
yielding a very small round-off error. This makes many of the
following divisions look strange.
*/

#define  HLSMAX   RANGE /* H,L, and S vary over 0-HLSMAX */
#define  RGBMAX   255   /* R,G, and B vary over 0-RGBMAX */
                        /* HLSMAX BEST IF DIVISIBLE BY 6 */
                        /* RGBMAX, HLSMAX must each fit in a byte. */

/* Hue is undefined if Saturation is 0 (grey-scale) */
/* This value determines where the Hue scrollbar is */
/* initially set for achromatic colors */
#define UNDEFINED (HLSMAX*2/3)

void  RGBtoHLS(lRGBColor)
DWORD lRGBColor;
```

```
{
   WORD R,G,B;          /* input RGB values */
   BYTE cMax,cMin;      /* max and min RGB values */
   WORD  Rdelta,Gdelta,Bdelta; /* intermediate value: % of spread from max
*/

   /* get R, G, and B out of DWORD */
   R = GetRValue(lRGBColor);
   G = GetGValue(lRGBColor);
   B = GetBValue(lRGBColor);

   /* calculate lightness */
   cMax = max( max(R,G), B);
   cMin = min( min(R,G), B);
   L = ( ((cMax+cMin)*HLSMAX) + RGBMAX )/(2*RGBMAX);

   if (cMax == cMin) {           /* r=g=b --> achromatic case */
      S = 0;                     /* saturation */
      H = UNDEFINED;             /* hue */
   }
   else {                        /* chromatic case */
      /* saturation */
      if (L <= (HLSMAX/2))
         S = ( ((cMax-cMin)*HLSMAX) + ((cMax+cMin)/2) ) / (cMax+cMin);
      else
         S = ( ((cMax-cMin)*HLSMAX) + ((2*RGBMAX-cMax-cMin)/2) )
            / (2*RGBMAX-cMax-cMin);

      /* hue */
      Rdelta = ( ((cMax-R)*(HLSMAX/6)) + ((cMax-cMin)/2) ) / (cMax-cMin);
      Gdelta = ( ((cMax-G)*(HLSMAX/6)) + ((cMax-cMin)/2) ) / (cMax-cMin);
      Bdelta = ( ((cMax-B)*(HLSMAX/6)) + ((cMax-cMin)/2) ) / (cMax-cMin);

      if (R == cMax)
         H = Bdelta - Gdelta;
      else if (G == cMax)
         H = (HLSMAX/3) + Rdelta - Bdelta;
      else /* B == cMax */
         H = ((2*HLSMAX)/3) + Gdelta - Rdelta;

      if (H < 0)
         H += HLSMAX;
      if (H > HLSMAX)
         H -= HLSMAX;
   }
}

/* utility routine for HLStoRGB */
WORD HueToRGB(n1,n2,hue)
WORD n1;
WORD n2;
WORD hue;
{

   /* range check: note values passed add/subtract thirds of range */
```

```
   if (hue < 0)
      hue += HLSMAX;

   if (hue > HLSMAX)
      hue -= HLSMAX;

   /* return r,g, or b value from this tridrant */
   if (hue < (HLSMAX/6))
      return ( n1 + (((n2-n1)*hue+(HLSMAX/12))/(HLSMAX/6)) );
   if (hue < (HLSMAX/2))
      return ( n2 );
   if (hue < ((HLSMAX*2)/3))
      return ( n1 + (((n2-n1)*(((HLSMAX*2)/3)-hue)+(HLSMAX/12))/(HLSMAX/6))
);
   else
      return ( n1 );
}


DWORD HLStoRGB(hue,lum,sat)
WORD hue;
WORD lum;
WORD sat;
{
   WORD R,G,B;                /* RGB component values */
   WORD  Magic1,Magic2;       /* calculated magic numbers (really!) */

   if (sat == 0) {            /* achromatic case */
      R=G=B=(lum*RGBMAX)/HLSMAX;
      if (hue != UNDEFINED) {
         /* ERROR */
      }
   }
   else  {                    /* chromatic case */
      /* set up magic numbers */
      if (lum <= (HLSMAX/2))
         Magic2 = (lum*(HLSMAX + sat) + (HLSMAX/2))/HLSMAX;
      else
         Magic2 = lum + sat - ((lum*sat) + (HLSMAX/2))/HLSMAX;
      Magic1 = 2*lum-Magic2;

      /* get RGB, change units from HLSMAX to RGBMAX */
      R = (HueToRGB(Magic1,Magic2,hue+(HLSMAX/3))*RGBMAX +
(HLSMAX/2))/HLSMAX;
      G = (HueToRGB(Magic1,Magic2,hue)*RGBMAX + (HLSMAX/2)) / HLSMAX;
      B = (HueToRGB(Magic1,Magic2,hue-(HLSMAX/3))*RGBMAX +
(HLSMAX/2))/HLSMAX;
   }

   return(RGB(R,G,B));
}
```

Additional reference words: 3.00 3.10 3.50 4.00 95 color RGB HLS HBS
KBCategory: kbprg
KBSubcategory: GdiPal

Copy on Write Page Protection for Windows NT

PSS ID Number: Q103858
Authored 01-Sep-1993 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1, 3.5, and 3.51

SUMMARY

The Windows NT's Copy on Write page protection is a concept that allows multiple applications to map their virtual address spaces to share the same physical pages, until an application needs to modify the page and have its own instance copy. This is part of a technique called Lazy Evaluation, which allows the system to not waste time by committing resources, time, or execution until/unless absolutely necessary. Copy on Write allows the virtual memory manager to save memory and execution time.

MORE INFORMATION

Copy on Write works as follows: In generic terms, an application can load something into its virtual memory (for example, a code section or DLL code). Virtual memory is mapped to physical memory. Another process may want to load the same thing into its virtual memory. As long as neither process writes to this memory, they can map to, and share, the same physical pages.

If either process needs to write to this memory, because the memory is marked as Copy on Write, the physical page frame will be copied somewhere else in physical memory. Fixups are made for the virtual memory mapping of the writing process. Both applications now have their own instance of the memory contents. In short, applications can share the same physical memory with Copy on Write, until one of the applications has to modify the contents. At that point, a new copy of the contents is made, and the writing process has its own copy.

It should be emphasized that this is not to say that applications are sharing memory in the sense that one application can write to it and another can read what the first one wrote; as long as applications are only going to read a piece of memory (for example, a code section), then the physical pages supporting that memory for the applications can be shared. Once the application needs to write to the memory (for example, in the form of a fixup), then that application must have a new physical page so that the modifications are not seen by other processes. The processes are no longer sharing the same physical pages.

Applications

When multiple instances of the same Windows-based application load, you may

notice that most, if not all, of their instance handles (hInstance) have the same value. In fact, almost all of the windows on the desktop have this value, which represents the base address where the application loaded in virtual memory.

Each of the instances of the same application running has its own protected virtual address space to run in. If each of these applications can load into its default base address, each will map to, and be able to share, the same physical pages in memory. Using Copy on Write, the system will allow these applications to share the same physical pages until one of the applications modifies a page. Then a copy is made in physical memory, and that process's virtual memory is fixed up to use the new physical page. If for some reason one of these instances cannot load in the desired base address, it will get its own physical pages. See the section on DLLs below for more explanation.

DLLs

Dynamic-link libraries (DLLs) are created with a default base address to load at. Assuming that multiple applications call the DLL, they will all try to load it within their own address space at that default virtual address. If they are all successful, they can all map that virtual address space to share the same physical pages.

However, if for some reason the DLL cannot be loaded within the process's address space at the default address, it will load the DLL elsewhere. The DLL must be copied into another physical address frame. The reason is that fixups for jump instructions in a DLL are written as specific locations within the DLL's pages. If the DLL can be loaded at the same base address for each process, the second to the nth process does not have to write that memory location for the jump. If a process cannot load the DLL at the specified base address, the locations written in the DLL's jumps will be different for this process. This forces the fixup to write a new location into the jump, and the Copy on Write will automatically force a new physical page.

Note that all references to data must be fixed up too. If this causes virtual memory of the code section to be updated, then the process will again go through the Copy on Write process. For example, if there are a great many places in the code section that make reference into data in a DLL, if the DLL cannot be loaded at its default location, the locations in the code section data in the DLL are referenced will have to be modified. If this is one of multiple instances of a process, these fixups must go through the copy on write process, and the virtual memory pages of the process's code section will not be able to map and share the same physical pages as the other instances. If there are a lot of references to the data in the DLL by the code section, this can essentially cause the entire code section to be copied to new physical pages.

POSIX

In POSIX, there is a fork() instruction that basically creates two copies of the same program. It is an expensive process for the system to copy the

address space of one process into another. Instead, under Windows NT, the system simply marks the parent's pages with Copy on Write. This way new physical frames are copied only if and when they are needed (have been modified). The system does not waste time or memory if all of the address space doesn't need to be copied. (For more information, see "Inside Windows NT" by Microsoft Press).

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseMm

Copying Compressed Files

PSS ID Number: Q130331
Authored 17-May-1995 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.5 and 3.51

SUMMARY

It is not possible to copy NTFS compressed files without uncompressing them. This functionality is not available in Windows NT versions 3.5 and 3.51; however, this feature may be included in a future version of Windows NT.

Compressed files are expanded via memory-mapped files. This minimizes the performance hit of expanding the file.

MORE INFORMATION

One of the new features found in Windows NT 3.51 is file compression. Files or directories can be compressed or decompressed by calling DeviceIoControl() with one of the following compression flags:

FSCTL_SET_COMPRESSION	Sets the compression state of a file or directory.
FSCTL_GET_COMPRESSION	Obtains the compression state of a file or directory.

Two additional FSCTL codes are documented in the Win32 SDK as "not implemented in this release." They are FSCTL_READ_COMPRESSION and FSCTL_WRITE_COMPRESSION. These additional FSCTL codes will be part of the functionality that will allow you to read and write files on an NTFS compressed drive without having to decompress them first. Again, this functionality may be included in a future release of Windows NT.

Additional reference words: 3.50 NTFS File Compression
KBCategory: kbprg kbusage
KBSubcategory: BseFileIo

Correct Use of Try/Finally

PSS ID Number: Q83670
Authored 19-Apr-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

Try/finally, used correctly, helps to provide a robust application. However, if used incorrectly it can cause unnecessary overhead. any flow of control out of the try body of try/finally is an abnormal termination that can cause hundreds of instructions to be executed on an x86 system, and thousands on a MIPS machine, even if control leaves the try body via a control statement on the very last statement of the try body. The language definition states that control must leave the try body sequentially for normal termination to occur (that is, execution falls through the bottom of the try body).

The following sample demonstrates an incorrect use of try/finally:

```
/* Incorrect use of try/finally */

VOID
function (
   DWORD ... P1,
      .
   DWORD ... Pn
   )

{

   try {
       if (...) {
            .
            .
           return;
       }
            .
            .
   } finally {
      .
      .
   }
   return;
}
```

The overhead can be avoided in the above example by moving the return AFTER the end of the finally clause. The following provides more detail on the correct use of try/finally.

MORE INFORMATION

Execution of a termination handler due to abnormal termination of a try body is expensive. Abnormal termination occurs when control leaves a try body in any way other than by falling through the bottom. Intentionally branching out of a try body is still an abnormal termination.

In the above example, abnormal termination of the try body occurs if the return in the middle of the try body is executed. If the predicate of the if is false, then extremely efficient execution of the finally clause occurs because this is not abnormal termination and the finally clause is called directly by inline code.

When abnormal termination occurs hundreds to thousands of instructions are executed because an unwind must be executed, which must search backward through frames to determine if any termination handlers should be called. On an x86 system, this executes the C run-time handler and examines the handler list. On a MIPS machine, this also causes the function table to be searched and the prologue of each intervening function to be executed backwards interpretively.

You should always avoid the execution of a termination handler as a result of the abnormal termination of a try body by a return, or other direct flow of control out of the try body. Abnormal termination occurs whenever control leaves the try body other than by falling through the bottom. This can occur because of a return, goto, continue, or break. It can also occur because of an exception, which presumably cannot be avoided.

In the above example, abnormal termination in the nonexception case can be eliminated easily as follows:

```
/* Correct use of try/finally */

VOID
function (
   DWORD ... P1,
      .
      .
   DWORD ... Pn
   )

{

   try {
       if (...) {
           .
           .
       } else {
           .
           .
       }
```

```
   } finally {
      .
      .
   }
   return;
}
```

Now both clauses of the if fall through to the termination handler in all but exceptional cases and execute the termination handler in the most efficient way. This also has the same logical execution as the previous sample.

In summary, the correct use of try/finally is a powerful method to help you write robust applications. Care should be taken to ensure the correct use of try/finally.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseExcept

CPU Quota Limits Not Enforced

PSS ID Number: Q100329
Authored 20-Jun-1993 Last modified 03-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1, 3.5, and 3.51

SUMMARY

On page 88 of "Inside Windows NT," Table 4-1 indicates that a process object contains a quota limit for the maximum amount of processor time that the process can use.

This limit is not enforced in Windows NT versions 3.1 or 3.5x.

MORE INFORMATION

The key to understanding Windows NT thread scheduling and resultant application behavior is knowing the central algorithm used. This algorithm is very simple, and is the same one a number of other operating systems use. It is "run the highest priority thread ready." A list of ready threads or processes exists; it is often called the "dispatch queue" or "eligible queue." The queue entries are in order based on their individual priority. A hardware-driven real-time clock or interval timer will periodically interrupt, passing control to a device driver that calls the process or thread scheduler. The thread scheduler will take the highest priority entry from the queue and dispatch it to run.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseProcThrd

CreateFile() Using CONOUT$ or CONIN$

PSS ID Number: Q90088
Authored 08-Oct-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

If you attempt to open a console input or output handle by calling the CreateFile() function with the special CONIN$ or CONOUT$ filenames, this call will return INVALID_HANDLE_VALUE if you do not use the proper sharing attributes for the fdwShareMode parameter in your CreateFile() call. Be sure to use FILE_SHARE_READ when opening "CONIN$" and FILE_SHARE_WRITE when opening "CONOUT$".

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseFileio

CreateFileMapping() SEC_* Flags

PSS ID Number: Q108231
Authored 07-Dec-1993 Last modified 05-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The definition of CreateFileMapping() is as follows:

```
HANDLE CreateFileMapping(hFile, lpsa, fdwProtect, dwMaximumSizeHigh,
   dwMaximumSizeLow, lpszMapName)

HANDLE hFile;
LPSECURITY_ATTRIBUTES lpsa;
DWORD fdwProtect;
DWORD dwMaximumSizeHigh;
DWORD dwMaximumSizeLow;
LPCTSTR lpszMapName;
```

The following flags are four possible values for the parameter fdwProtect:

SEC_COMMIT
   All pages of a section are to be set to the commit state.
SEC_IMAGE
   The file specified for a section's file mapping is an executable image file.
SEC_NOCACHE
   All pages of a section are to be set as noncacheable.
SEC_RESERVE
   All pages of a section are to be set to the reserved state.

If none of these flags are specified, SEC_COMMIT is the default. This behaves the same way as MEM_COMMIT in VirtualAlloc().

MORE INFORMATION

Windows NT

The SEC_RESERVE flag is intended for file mappings that are backed by the paging file, and therefore use SEC_RESERVE only when hFile is -1. The pages are reserved just as they are when the MEM_RESERVE flag is used in VirtualAlloc(). The pages can be committed by subsequently using the VirtualAlloc() application programming interface (API), specifying MEM_COMMIT. Once committed, these pages cannot be decommitted.

The SEC_NOCACHE flag is intended for architectures that require various

locking structures to be located in memory that is not ever fetched into the CPU cache. On x86 and MIPS machines, use of this flag just slows down the performance because the hardware keeps the cache coherent. Certain device drivers may require noncached data so that programs can write through to the physical memory. SEC_NOCACHE requires that either SEC_RESERVE or SEC_COMMIT be used in addition to SEC_NOCACHE.

The SEC_IMAGE flag indicates that the file handle points to an executable file, and it should be loaded as such. The mapping information and file protection are taken from the image file, and therefore no other options are allowed when SEC_IMAGE is used.

Windows 95

Under Windows NT, the Win32 loader simply sits on top of the memory mapped file subsystem, and so when the loader needs to load a PE executable, it simply calls down into the existing memory mapped file code. Therefore, it is extremely easy for to support SEC_IMAGE in CreateFileMapping() under Windows NT.

In Windows 95, the loader is more complex and the memory mapped files are simple and only support the bare minimum of functionality to make the existing MapViewOfFile() work. Therefore, Windows 95 does not support SEC_IMAGE. There is support for SEC_NOCACHE, SEC_RESERVE and SEC_COMMIT.

Additional reference words: 3.10 3.50 4.00
KBCategory: kbprg
KBSubcategory: BseMm

Creating a Font for Use with the Console

PSS ID Number: Q105299
Authored 17-Oct-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1 and 3.5

It is possible to use FontEdit to create a font that can be used by the console. The following must be true:

- The face name must be System, Terminal, or Courier
- The font size must be different from any of the other console fonts
- The font must be fixed pitch
- The font must not be italic

In addition, in the U.S. market, the font should support codepage 437.

Install the font from the Control Panel. After rebooting, the font will be available to the console.

An EnumFonts() call is made by the console during its initialization to determine what fonts are available. The console saves a set of one-to-one mappings between the font sizes listed and a set of LOGFONTs. The console never has direct knowledge of what file is used.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: GdiFnt

Creating a Hidden MDI Child Window

PSS ID Number: Q70080
Authored 09-Mar-1991 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

Whenever Windows creates a new multiple-document interface (MDI) child window in response to a WM_MDICREATE message, it makes that child window visible.

The information below describes how to create a hidden MDI child window without causing an unattractive "flash" on the screen as the window is created visible and then hidden.

MORE INFORMATION

A code fragment such as the following can be used to create an invisible MDI child:

```
MDICREATESTRUCT mcs;            // structure to pass with WM_MDICREATE
HWND            hWndMDIClient;  // the MDI client window
HWND            hwnd;           // temporary window handle

      ...

// assume that we have already filled out the MDICREATESTRUCT...

// turn off redrawing in the MDI client window
SendMessage(hwndMDIClient, WM_SETREDRAW, FALSE, 0L);

/*
 * Create the MDI child. It will be created visible, but will not
 * be seen because redrawing to the MDI client has been disabled
*/
hwnd = (WORD)SendMessage(hwndMDIClient,
                         WM_MDICREATE,
                         0,
                         (LONG)(LPMDICREATESTRUCT)&mcs);

// hide the child
ShowWindow(hwnd, SW_HIDE);

// turn redrawing in the MDI client back on,
// and force an immediate update
SendMessage(hwndMDIClient, WM_SETREDRAW, TRUE, 0L);
InvalidateRect( hwndMDIClient, NULL, TRUE );
```

```
UpdateWindow( hwndMDIClient );
      ...
```

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMdi

Creating a List Box That Does Not Sort

PSS ID Number: Q68116
Authored 08-Jan-1991 Last modified 16-May-1995

-
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

-

A Windows list box with the LBS_STANDARD style will sort the list of items into alphabetical order before displaying them in the control.

To create a list box that will not sort, you must remove the LBS_SORT bit from the window style. The following style specification removes this bit:

```
(LBS_STANDARD | LBS_HASSTRINGS) & ~LBS_SORT
```

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Creating a List Box with No Vertical Scroll Bar

PSS ID Number: Q68115
Authored 08-Jan-1991 Last modified 16-May-1995

--
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0
--

A Windows list box with the LBS_STANDARD style will display a vertical scroll bar if there are more items in the list than can be displayed in the client area of the list box.

To create a list box that will not use a vertical scroll bar, you must remove the WS_VSCROLL bit from the window style. The following style specification removes this bit:

```
(LBS_STANDARD | LBS_HASSTRINGS) & ~WS_VSCROLL
```

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Creating a List Box Without a Scroll Bar

PSS ID Number: Q11365
Authored 01-Dec-1987 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

When LBS_STANDARD is used as follows

```
   LBS_NOTIFY | LBS_SORT | WS_BORDER | LBS_STANDARD
```

the following results (as defined in WINDOWS.H):

```
   LBS_STANDARD = #00A00003;
                  /* LBS_NOTIFY | LBS_SORT | WS_VSCROLL | WS_BORDER */
```

To create a dialog box that contains a list box without the vertical scroll bar, use NOT WS_VSCROLL as the style for creating a list box control without a vertical scroll bar, as follows:

```
   (LBS_STANDARD & ~WS_VSCROLL)  // NOT WS_VSCROLL
```

Additional reference words: 3.00 3.10 3.50 4.00 95 listbox
KBCategory: kbprg
KBSubcategory: UsrCtl

Creating a Logical Font with a Nonzero lfOrientation

PSS ID Number: Q104010
Authored 02-Sep-1993 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, and 3.51

SUMMARY

To create a font that writes in a direction other than left to right, an application should specify a nonzero lfEscapement in the LOGFONT structure that is passed to CreateFontIndirect(). This method works under Windows NT regardless of the graphics mode of the device context.

To create a font where the characters themselves are rotated, the application should specify a nonzero lfOrientation in the LOGFONT structure that is passed to CreateFontIndirect(). However, this setting is ignored in Windows NT unless the graphics mode is set to GM_ADVANCED.

Therefore, to successfully create a logical font with a nonzero lfOrientation, use

```
SetGraphicsMode( hDC, GM_ADVANCED )
```

to set the graphics mode of the device context to GM_ADVANCED.

MORE INFORMATION

The TTFONTS sample program is a good way to quickly and easily see the effects of the lfEscapement and lfOrientation fields. However, TTFONTS does not set the graphics mode of its test window HDC to GM_ADVANCED. As a result, the lfOrientation field apparently is ignored. It is easy to modify the DISPLAY.C module of TTFONTS in order to set the graphics mode of the window HDC to GM_ADVANCED.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: GdiFnt

Creating a Multiple Line Message Box

PSS ID Number: Q67210
Authored 27-Nov-1990 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

Message boxes are used to provide information to the user of an application. Error messages and warnings are also provided through message boxes. This article provides details on using message boxes in applications.

MORE INFORMATION

Message boxes are modal windows. When an application displays an application modal message box, which is the default message box type, the user cannot interact with any part of that application until the message box has been dismissed. However, the user may use the mouse or keyboard to activate another application and interact with it while the message box is displayed. Certain critical errors that may affect all of Windows are displayed in system modal message boxes. Windows will not perform any operations until the error condition is acknowledged and the system modal message box is dismissed.

There are times where it is necessary to display a long message in a message box. Windows does this when you start an MS-DOS-based application that uses graphics from inside an MS-DOS window. To break a message into many lines, insert a newline character into the message text. Here is a sample MessageBox() call:

```
MessageBox(hWnd, "This is line 1.\nThis is line 2.", "App",
    MB_OK | MB_ICONQUESTION);
```

If the text of a message is too long for a single line, Windows will break the text into multiple lines.

System modal message boxes treat the newline character as any other. A newline character is displayed as a black block in the text. Because system modal message boxes are designed to work at all times, even under extremely low memory conditions, it does not provide the ability to display more than one line of text.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDlgs

Creating a Nonblinking Caret

PSS ID Number: Q74607
Authored 24-Jul-1991 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK)versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

The Microsoft Windows graphical environment is designed to provide a blinking caret. However, using a timer and the SetCaretBlinkTime() function, an application can "trick" the caret into not blinking.

MORE INFORMATION

Although Windows is designed to blink the caret at a specified interval, a timer function and SetCaretBlinkTime() can be used to prevent Windows from turning the caret off by following these three steps:

1. Call SetCaretBlinkTime(10000), which instructs Windows to blink the caret every 10,000 milliseconds (10 seconds). This results in a "round-trip" time of 20 seconds to go from OFF to ON and back to OFF (or vice versa).

2. Create a timer, using SetTimer(), specifying a timer procedure and a 5,000 millisecond interval between timer ticks.

3. In the timer procedure, call SetCaretBlinkTime(10000). This resets the timer in Windows that controls the caret blink.

When an application implements this procedure, Windows never removes the caret from the screen, and the caret does not blink.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCrt

Creating a World SID

PSS ID Number: Q111543
Authored 14-Feb-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

The sample code below demonstrates how to create a World Security Identifier (SID). The World SID (S-1-1-0) is a group that includes all users. To determine if a SID (perhaps in an Access Control Entry) is the World SID, you must first create a World SID to compare it to. Once you have created a World SID, you can use the EqualSid() API (application programming interface) to determine equality.

Sample Code

```
PSID psidWorldSid;
SID_IDENTIFIER_AUTHORITY siaWorldSidAuthority =
     SECURITY_WORLD_SID_AUTHORITY;

psidWorldSid = (PSID)LocalAlloc(LPTR,GetSidLengthRequired( 1 ));

InitializeSid( psidWorldSid, &siaWorldSidAuthority, 1);
*(GetSidSubAuthority( psidWorldSid, 0)) = SECURITY_WORLD_RID;
```

Additional reference words: 3.10 and 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

Creating Access Control Lists for Directories

PSS ID Number: Q115948
Authored 07-Jun-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

The discretionary access control list (DACL) for a directory usually differs from that of a file. When assigning security to a directory, you are often specifying both the security for the directory and the security for any contained files and directories.

A directory's ACL will normally contain at least two access control entries (ACE):

- An ACE for the directory itself and any subdirectories.

- An ACE for any files in the directory.

If an ACE is to apply to object in the directory (subdirectories and files), the ACE is marked as an OBJECT_INHERIT_ACE and/or a CONTAINER_INHERIT_ACE. (In this article, a container means a directory.)

For example, when you use File Manager to set the security on a directory to "Change (RWXD)(RWXD)," the directory's DACL contains the following two ACEs:

```
   ACE1 (applies to files in the directory)
      ACE flags:   INHERIT_ONLY_ACE | OBJECT_INHERIT_ACE
      Access Mask: DELETE | GENERIC_READ | GENERIC_WRITE |
                   GENERIC_EXECUTE

   ACE2 (applies to the directory and subdirectories)
      ACE flags:   CONTAINER_INHERIT_ACE
      Access Mask: DELETE | FILE_GENERIC_READ | FILE_GENERIC_WRITE |
                   FILE_GENERIC_EXECUTE
```

MORE INFORMATION

The ACE flags are part of the ACE header. The structure of an ACE header can be found in the online help by searching on "ACE_HEADER".

In the above example, ACE1 applies only to contained files through the INHERIT_ONLY_ACE flag. If INHERIT_ONLY_ACE is not specified in an ACE, the ACE applies only to the current container.

NOTE: Adding one of these ACEs to a directory does not change the security

for any contained files or directories. The ACEs are only copied to files and directories created after the ACEs have been added to the directory.

When adding your own security to files, it is easy to create a combination that File Manager does not recognize as a "standard" setting. This is shown in file manager as "special" security.

If you want to match the DACLs you create to those used by File Manager, you can set the security of a file or directory in File Manager and then check the DACLs and ACEs. A tool for this is provided as a sample called "Check_SD" in the Win32 SDK. Check_SD can be found in the Q_A\SAMPLES\CHECK_SD directory on the Win32 SDK CD.

REFERENCES

- "Microsoft Win32 Programmer's Reference," Microsoft Corporation.

- "Microsoft Win32 SDK API Reference help file," Microsoft Corporation.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

Creating and Using a Custom Caret

PSS ID Number: Q74514
Authored 22-Jul-1991 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

In the Microsoft Windows graphical environment, creating a custom caret is simple. Windows has a series of caret control, creation, and deletion functions specifically designed to make manipulating the caret easy.

MORE INFORMATION

The caret is a shared system resource. Unlike brushes, pens, device contexts and such, but like the cursor, only one caret is available under Windows. Also, like the cursor, an application can define a custom shape for the caret.

The CreateCaret() function creates a custom caret. Its syntax is as follows:

```
void CreateCaret(HWND hWnd, HBITMAP hBitmap,
                 int nWidth, int nHeight);
```

The caret shape can be a line, a block, or a bitmap specified as the hBitmap parameter. If the hBitmap parameter contains a valid handle [a bitmap handle returned from the CreateBitmap(), CreateDIBitmap(), or LoadBitmap() function], CreateCaret() ignores the values of its nWidth and nHeight parameters and uses the dimensions of the bitmap. If hBitmap is NULL, the caret is a solid block; if hBitmap is one, the caret is a gray block. The nWidth and nHeight parameters specify the caret size in logical units. If either nWidth or nHeight is zero, the caret width or height is set to the window-border width or height.

If an application uses a bitmap for the caret shape, the caret can be in color; unlike the cursor, the caret is not restricted to monochrome.

CreateCaret() automatically destroys the previous caret shape, if any, regardless of which window owns the caret. The new caret is initially hidden; call the ShowCaret() function to display the caret.

Because the caret is a shared resource, a window should create a caret only when it has the input focus or is active. It should destroy the caret before it loses the input focus or becomes inactive. Only the window that owns the caret should move it, show it, hide it, or modify

it in any way.

Other functions related to the caret are the following:

- SetCaretPos()
  This function moves the caret to the specified position (in logical coordinates).

- GetCaretPos()
  This function retrieves the caret's current position (in screen coordinates).

- ShowCaret()
  This function shows the caret on the display at the caret's current position. When shown, the caret flashes automatically. If the caret is not owned by the window specified in the call, the caret is not shown.

- HideCaret()
  This function hides the caret by removing it from the display screen. HideCaret() hides the caret only if the window handle specified in the call is the window that owns the caret. Hiding the caret does not destroy it.

  NOTE: Hiding the caret is cumulative; ShowCaret() must be called once for every call to HideCaret(). For example, if HideCaret() is called five times, ShowCaret() must be called five times for the caret to be shown.

- DestroyCaret()
  This function removes the caret from the screen, frees the caret from the current owner-window, and destroys the current shape of the caret. It destroys the caret only if the current task owns the caret. This call should be used in conjunction with CreateCaret(). DestroyCaret() does not free or destroy a bitmap used to define the caret shape.

- SetCaretBlinkTime()
  This function sets the caret blink rate. After the blink rate is set, it remains the same until the same window changes it again, another window changes it, another application changes it, or Windows is rebooted.

- GetCaretBlinkTime()
  This function returns the current caret blink rate.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCrt

Creating Autosized Tables with Windows Help

PSS ID Number: Q81233
Authored 01-Mar-1992 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, versions 3.5 and 3.51

SUMMARY

A Windows Help file can contain a table to present information in a consistent manner. Help 3.1 supports "autosized" tables where the relative widths of the columns remain the same, even as the absolute size of the table changes. The absolute size is determined by the size of the Windows Help window.

MORE INFORMATION

To create an autosized table in the RTF file, create a centered table. Create the text for the table using the minimum possible width for each column. If the Help window is larger, the columns will be wider and Help will compute where the text wraps based on the available width. If the user sizes the Help window smaller than the table's authored size, Help will display a horizontal scroll bar.

In Word for Windows version 1.1, use the Table command on the Format menu to create a centered table. In Word for Windows 2.0, create a table, then choose Row Height from the Table menu and choose the Center button in the Alignment group.

Additional reference words: 3.10 3.50
KBCategory: kbtool
KBSubcategory: TlsHlp

Creating Instance Data in a Win32s DLL

PSS ID Number: Q109620
Authored 05-Jan-1994 Last modified 24-Feb-1995

The information in this article applies to:

- Microsoft Win32s versions 1.0, 1.1, and 1.2

SUMMARY

The Win32 dynamic-link libraries (DLLs) that are running on Win32s use shared data by default. This means that any global data in the DLL is shared by all processes that use the DLL. Thread local storage (TLS) can be used to create instance data; that is, data in the DLL that is specific to each process.

The Win32s "Programmer's Reference" mentions that TLS can be used to create instance data, but provides no details. The sample code below shows the source for a DLL that uses instance data on both Win32 and Win32s. The sample code was built using Microsoft Visual C++, 32-bit edition.

If you use a development environment that does not have similar support for TLS, you should still be able to use the API (application programming interface) calls. The API calls for TLS are TlsAlloc, TlsGetValue, TlsSetValue, and TlsFree.

MORE INFORMATION

One reason for wanting to create instance data on Win32s is to create a DLL that behaves identically on Win32s and Win32 (although it introduces extra overhead on Windows NT and Windows 95). Another way to create a DLL that behaves identically on Win32s and Win32 is to share all of the data in the Win32-based DLL. For additional information, please see the following article(s) in the Microsoft Knowledge Base:

```
   ARTICLE-ID: Q109619
   TITLE     : Sharing All Data in a DLL
```

Sample Code

```
/* Compile options used: /LD /MD
*/

int __declspec(thread) nVar = 0;    // Variables should be initialized

int __declspec(dllexport) GetVar()
{
    return nVar;
}
```

```
void __declspec(dllexport) SetVar(int nNew)
{
    nVar = nNew;
}
```

Additional reference words: 1.00 1.10 1.20
KBCategory: kbprg
KBSubcategory: W32s

Creating Lines with a Nonstandard Pattern

PSS ID Number: Q34614
Authored 22-Aug-1988 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The Microsoft Windows graphical environment provides six predefined pens for drawing dotted, dashed, and solid lines. However, an application cannot draw fine gray lines, such as those on a Microsoft Excel spreadsheet, with these pens. This article describes how to create such lines.

MORE INFORMATION

An application can use the LineDDA function to produce any type of patterned line. Based on the endpoints of a line, LineDDA calculates each point on the line and calls an application-defined callback function for each point. The callback function is free to use the calculated points in any manner desired. An application can draw a gray line similar to those used in Excel by calling the the SetPixel function in the callback function to draw every other point.

For example, the following code calculates all points on the line from coordinates (30, 40) to (100, 100). Then it calls the function pointed to by the lpfnLineProc variable with the points and the handle to a device context (hDC) as parameters:

```
LineDDA(30, 40, 100, 100, lpfnLineProc, (LPSTR)hDC);
```

For more information on this function, see pages 4-272 and 4-273 of the "Microsoft Windows Software Development Kit Reference, Volume 1" for Windows 3.0 or pages 568 and 569 of the "Microsoft Windows Software Development Kit: Programmer's Reference, Volume 2: Functions" for Windows 3.1. Charles Petzold's book "Programming Windows 3" (Microsoft Press, 1990) demonstrates using the LineDDA function in a programming example on pages 593 through 598.

The following code fragment draws 50 random Excel-style lines. Note that the LineProc function must be listed as an EXPORT in the module definition (DEF) file:

```
case WM_PAINT:
   {
   HDC hDC;
   int nIndex;
```

```
      PAINTSTRUCT ps;

      hDC = BeginPaint(hWnd, &ps);

      for (nIndex = 0; nIndex < 50; nIndex++)
         LineDDA(rand() / 110, rand() / 110, rand() / 110,
                 rand() / 110, lpfnLineProc, (LPSTR)hDC);

      EndPaint(hWnd, &ps);
      break;
      }

void FAR PASCAL LineProc(x, y, lpData)
short x, y;
LPSTR lpData;
{
   static short nTemp = 0;

   if (nTemp == 1)
      SetPixel((HDC)lpData, x, y, 0L);

   nTemp = (nTemp + 1) % 2;
}
```

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiDrw

Creating Windows in a Multithreaded Application

PSS ID Number: Q90975
Authored 26-Oct-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

In a multithreaded application, any thread can call the CreateWindow() API to create a window. There are no restrictions on which thread(s) can create windows.

It is important to note that the message loop and window procedure for the window must be in the thread that created the window. If a different thread creates the window, the window won't get messages from DispatchMessage(), but will get messages from other sources. Therefore, the window will appear but won't show activation or repaint, cannot be moved, won't receive mouse messages, and so on.

MORE INFORMATION

Normally, windows created in different threads process input independently of each other. The windows have their own input states and the threads are not synchronized with eachother in regards to input processing.

In order to have threads to share input state, have one thread call AttachThreadInput() to have its input processing attached to another thread. What this means is that these two threads will use a Windows 3.1 style system queue. The threads will still have separate input, but they will take turns reading out of the same queue.

Creating a window can force an implicit AttachThreadInput(), when a parent window is created in one thread and the child window is being created in another thread. When windows are created (or set) in separate threads with a parent-child relationship, the input queues are attached.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrWndw

Creating/Managing User Accounts Programmatically

PSS ID Number: Q119671
Authored 20-Aug-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

Windows NT and the Windows NT Advanced Server use the same APIs that Microsoft LAN Manager uses to create and maintain user- and group-account information. For example, to create a new global group, use NetGroupAdd(). To create a new user, use NetUserAdd(). To add the user to the global group, use NetGroupAddUser(). Local groups are created by using NetLocalGroupAdd() and members are added to local groups by using NetLocalGroupAddMember().

MORE INFORMATION

The APIs NetGroupAdd(), NetUserAdd(), NetGroupAddUser(), NetLocalGroupAdd(), and NetLocalGroupAddMember() require access at the administrator or accounts-operator level to run successfully. Windows NT includes the following built-in groups:

- Administrators
- Power Users
- Users
- Guests

Members of the Administrators group can fully administer user accounts; only Administrators can assign user rights and access privileges for resources. Members of the Power Users group can create accounts only in the Power Users, Users, and Guests groups; they can also maintain and delete the accounts they create. However, a Power User can neither change nor delete an account in these groups if the account was created by someone else. A member of the Users group can create, maintain, and delete accounts in local groups that he or she has created. Guests can neither create nor delete accounts.

REFERENCES

In the Win32 SDK, version 3.1, the documentation for the ported LAN Manager APIs is available in the file LMAPI.HLP on the SDK CD. In the installed Win32 SDK, version 3.5, the ported LAN Manager APIs are documented in the Help file "Win32 API Reference".

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: NtwkLmapi

Critical Sections Versus Mutexes

PSS ID Number: Q105678
Authored 22-Oct-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

Critical sections and mutexes provide synchronization that is very similar, except that critical sections can be used only by the threads of a single process. There are two areas to consider when choosing which method to use within a single process:

1. Speed. The Synchronization overview says the following about critical sections:

   > ... critical section objects provide a slightly faster, more efficient mechanism for mutual-exclusion synchronization.

   Critical sections use a processor-specific test and set instruction to determine mutual exclusion.

2. Deadlock. The Synchronization overview says the following about mutexes:

   > If a thread terminates without releasing its ownership of a mutex object, the mutex is considered to be abandoned. A waiting thread can acquire ownership of an abandoned mutex, but the wait function's return value indicates that the mutex is abandoned.

   WaitForSingleObject() will return WAIT_ABANDONED for a mutex that has been abandoned. However, the resource that the mutex is protecting is left in an unknown state.

   There is no way to tell whether a critical section has been abandoned.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseSync

CS_SAVEBITS Class Style Bit

PSS ID Number: Q31073
Authored 01-Jun-1988 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

If the CS_SAVEBITS style is included when registering a pop-up window, a bitmap copy of the screen image that the window will obscure is saved in memory when the window is displayed.

The bitmap is redisplayed at its original location and no WM_PAINT messages are sent to the obscured windows if the following is true when the window is removed from the display:

- The memory used by the saved bitmap has not been discarded.

- Other screen actions have not invalidated the image that has been stored.

As a general rule, this bit should not be set if the window will cover more than half the screen; a lot of memory is required to store color bitmaps.

The window will take longer to be displayed because memory needs to be allocated. The bitmap also needs to be copied over each time the window is shown.

Use should be restricted to small windows that come up and are then removed before much other screen activity takes place. Any memory calls that will discard all discardable memory, and any actions that take place "under" the window, will invalidate the bitmap.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrPnt

CTRL+C Exception Handling Under WinDbg

PSS ID Number: Q97858
Authored 22-Apr-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

An exception is raised for CTRL+C only if the process is being debugged. The purpose is to make it convenient for the debugger to catch CTRL+C in console applications. For the purposes of this article, the debugger is assumed to be WinDbg.

MORE INFORMATION

When the console server detects a CTRL+C, it uses CreateRemoteThread() to create a thread in the client process to handle the event. This new thread then raises an exception IF AND ONLY IF the process is being debugged. At this point, the debugger either handles the exception or it continues the exception unhandled.

The "gh" command marks the exception as having been handled and continues the execution. The application does not notice the CTRL+C, with one exception: CTRL+C causes alertable waits to terminate. This is most noticable when executing:

```
while( (c = getchar()) != EOF )  - or -  while( gets(s) )
```

It is not possible to get the debugger to stop the wait from terminating.

The "gn" command marks an exception as unhandled and continues the execution. The handler list for the application is searched, as documented for SetConsoleCtrlHandler(). The handler is executed in the thread created by the console server.

After the exception is handled, the thread created to handle the event terminates. The debugger will not continue to execute the application if Go On Thread Termination is not enabled (from the Options menu, choose Debug, and select the Go On Thread Termination check box). The thread and process status indicate that the application is stopped at a debug event. As soon as the debugger is given a go command, the dead thread disappears and the application continues execution.

There are three cases where CTRL+C doesn't cause the program to stop executing (instead it causes a "page down"):

1. When CTRL+C is already being handled.

2. When the debugger is in the foreground and a source window has the focus (both must be true).

3. When the CTRL+C exception is disabled (through the Debugger Exceptions dialog box).

This follows the convention of the WordStar/Turbo C/Turbo Pascal editor commands.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsWindbg

CTYPE Macros Function Incorrectly

PSS ID Number: Q94323
Authored 04-Jan-1993 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

When an application that is linked to CRTDLL.LIB is compiled without defining _MT and _DLL, the CTYPE.H family of macros will not operate correctly.

To define _MT and _DLL on the CL command line, just add the following to the command line:

```
-D_MT -D_DLL
```

By adding these defines, the CTYPE macros will be properly initialized.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsMisc

Custom Controls Must Use CS_DBLCLKS with Dialog Editor

PSS ID Number: Q71223
Authored 10-Apr-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

For a custom control to function properly with the Dialog Editor, the custom control window class must include the CS_DBLCLKS style.

If the custom control does not have the CS_DBLCLKS style, double-clicking the control in the Dialog Editor does not cause the custom control function to display its style dialog box. However, the control's style dialog box is still accessible from the Styles command on the Edit menu.

MORE INFORMATION

The Dialog Editor subclasses each control it creates and processes WM_LBUTTONDBLCLK messages. In response to this message, the custom control is asked to display its style dialog box.

If the custom control window class does not have the CS_DBLCLKS style, Windows does not send any WM_LBUTTONDBLCLK messages to the control. As a result, the Dialog Editor does not call the style dialog box function for the custom control and no dialog box appears.

NOTE: This article does not apply to the resource editor included with the Visual C++ development environment.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Customizing a Pop-Up Menu

PSS ID Number: Q12118
Authored 23-Oct-1987 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

The following are three methods to customize a pop-up menu:

1. Use the word SEPARATOR in a pop-up menu, to produce a horizontal bar.

2. Use the word MENUBREAK, to start the menus on another column.

3. Place the vertical bar symbol in the menu string to display a vertical bar on the menu.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMen

Customizing the FileOpen Common Dialog in Windows 95

PSS ID Number: Q125706
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

In Windows version 3.x, customizing the FileOpen common dialog meant providing a special hook function, and in most cases, a custom dialog box template. This custom dialog box template is created by modifying the standard FileOpen dialog box template used by COMMDLG, which was actually made available as part of the Windows versopm 3.1 SDK.

In Windows 95, the new dialog box templates will no longer be made available for modification. Instead, an application should provide a dialog template that includes only the items to be added to the standard dialog box. COMMDLG will then create this dialog as a child of the standard dialog box. Because it is a child, it must have the WS_CHILD style set.

Note that the Windows version 3.1 method of customizing common dialogs by modifying the dialog templates will still work for 16-bit applications.

MORE INFORMATION

Follow these steps to customize the FileOpen common dialog in Windows 95:

1. Create a dialog box template that will have all the controls you want to add to the FileOpen common dialog. Be sure to specify the styles:

   ```
   WS_CHILD | WS_VISIBLE | DS_3DLOOK | DS_CONTROL | WS_CLIPSIBLINGS
   ```

   WS_CHILD is specified because without it, the call to GetOpenFileName() fails. COMMDLG creates the dialog specified as a child of the standard FileOpen common dialog box. As a result, the hDlg passed to the application's hook function will be the child of the standard FileOpen dialog box. To get a handle to the standard dialog box from the hook function, call GetParent (hDlg).

   WS_CLIPSIBLINGS is specified so that overlapping controls paint properly.

   DS_3DLOOK is a new style for Windows 95 that gives the dialog box a nonbold font, and gives all the controls the 3D look.

   DS_CONTROL is another new style that among other things allows the user to tab between the controls of a dialog box to the controls of a child dialog box. As mentioned above, the dialog template will

be created as a child of the standard FileOpen common dialog box, so specifying this style will allow tabbing from the application-defined controls to the standard controls.

2. Include a static control in your dialog template, specifying a control ID of stc32. This control will serve as a placeholder for the standard controls.

   If there is no stc32 control specified, COMMDLG places all the new controls defined in your dialog template below the standard controls and looks at the size of the static control to attempt to fit all the standard controls in it. If it is not big enough, COMMDLG resizes this stc32 control to make room for the standard controls, and then repositions the new controls with respect to the resized stc32 control.

   Be sure to use the #include directive to include DLGS.H in your .RC file, as stc32 is defined in <dlgs.h>.

3. Initialize the Flags member of the OPENFILENAME structure to include the following flags:

   ```
   OFN_EXPLORER | OFN_ENABLETEMPLATE | OFN_ENABLEHOOK
   ```

   OFN_ENABLETEMPLATEHANDLE may be used instead of OFN_ENABLETEMPLATE if you want to specify a handle to a memory block containing a preloaded dialog box template.

4. If the OFN_ENABLETEMPLATE flag is set, specify the name of your application-defined template in the lpTemplateName field of the OPENFILENAME structure, and specify your application's instance handle in the hInstance field.

   If the OFN_ENABLETEMPLATEHANDLE flag is set, specify the handle to the memory block containing your dialog box template in the hInstance field of the OPENFILENAME structure.

5. Specify the address of a dialog box procedure associated with your dialog box in the lpfnHook field of the OPENFILENAME structure.

6. Process appropriate notifications and messages as a result of adding new controls.

REFERENCES

Much of the information contained in this article is derived from the MSDN Technical Article entitled "Using the Common Dialogs Under Windows 95." Please refer to that article for more information.

Additional reference words: 4.00
KBCategory: kbprg
KBSubcategory: UsrCmnDlg

Dangers of Uninitialized Data Structures

PSS ID Number: Q74277
Authored 15-Jul-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

In general, all fields in structures passed to functions in the Microsoft Windows graphical environment should be initialized. If a field is not initialized, it may contain random data, which can cause unexpected behavior.

For example, before an application registers a window class, it must initialize the cbClsExtra and cbWndExtra fields of the WNDCLASS data structure. Windows allocates cbClsExtra bytes for the class, and cbWndExtra bytes for each window created using the class. If these fields contain large random values, the application may run out of memory quickly.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrWndw

DDE Error Message: Application Using DDE Did Not Respond

PSS ID Number: Q94955
Authored 26-Jan-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, version 4.0

SUMMARY

DDEML displays a dialog box with the following error message when a terminate deadlock occurs, often caused by a DDE application not terminating correctly:

Application using DDE did not respond to the System's Exit command

MORE INFORMATION

A terminate deadlock situation occurs when DDEML times out while waiting for a responding terminate.

This message box appears when a DDEML application calls DdeUninitialize() with conversations still active. DdeUninitialize() posts WM_DDE_TERMINATE messages for each open conversation, and waits for a corresponding WM_DDE_TERMINATE for a set period of time. This time is actually set in the [DDEML] section of the WIN.INI file

```
[DDEML]
ShutdownTimeout= ?
ShutdownRetryTimeout=?
```

where both are defined as integers defaulting to 30000 milliseconds. These WIN.INI entries were purposely not documented to discourage people from setting them to some other value.

If DDEML does not receive a response within the set period of time, it brings up the message box to allow the user to choose to either quit, wait longer, or wait indefinitely. This was done to work around a problem in Windows 3.0 where the system locks up if an application attempts to post a message to a non-existent window, and to allow the user to save his work.

Additional reference words: 3.10 3.00 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDde

DdeInitialize(), DdeNameService(), APPCMD_FILTERINITS

PSS ID Number: Q108925
Authored 20-Dec-1993 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

DdeInitialize() and DdeNameService() work as complimentary functions during initialization of a server application. When called for the first time to initialize a server application, DdeInitialize() will cause DDEML to append the APPCMD_FILTERINITS flag to the third parameter of the function call by default. Once this flag is set, client applications will not be able to connect to the server until it is reset. This flag is reset when the function DdeNameService() is called to register the service name with DDEML. Also, if the server wishes to remain anonymous, then DdeInitialize() must be called the second time to specifically turn off the APPCMD_FILTERINITS flag.

MORE INFORMATION

The DdeInitialize() function registers an application with DDEML. This function must be called by both the client and the server applications before calling any other DDEML function.

When the server application calls DdeInitialize() for the first time, DDEML appends the APPCMD_FILTERINITS flag to the third parameter of the function call, regardless of whether the application specifies this flag. This flag when used, will prevent DDEML from sending the XTYP_CONNECT and XTYP_WILDCONNECT transactions to the server application until the server has created its string handles and performed other application-specific initialization. The server application then calls DdeNameService() to register its service name with DDEML so that other client or server applications are notified of its existence. Calling DdeNameService() after the server has gone through the process of initialization turns off the APPCMD_FILTERINITS flag.

Some DDEML server applications might not want to register their names with DDEML because of various reasons (for example, the server application is a custom server application that wants to service particular clients, and thus wishes to remain anonymous to the rest of the system).

In special cases like this, an application may choose not to call the DdeNameService() function, because this function broadcasts the name of the server to all DDEML applications on the system. Not calling the DdeNameService() function, however, causes the APPCMD_FILTERINIT flag not to be reset properly, thus keeping the server from getting any XTYP_CONNECT or XTYP_WILDCONNECT transactions even from its clients.

One other way to reset this flag is to call DdeInitialize() a second time, without specifying the APPCMD_FILTERINITS flag.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDde

DDEML Application-Instance IDs Are Thread Local

PSS ID Number: Q94091
Authored 23-Dec-1992 Last modified 01-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1 and 3.5

When using the DDEML (Dynamic Data Exchange Management Library) libraries from a spawned thread, the application-instance ID that is returned in the lpidInst parameter of DdeInitialize is thread local.

Therefore, the application-instance ID cannot be used by any other thread that is spawned by the process, nor can it be inherited from the parent.

To use the DDEML libraries within a thread, it is necessary to make both the DdeInitialize call and to use the DdeUninitialize call from within the thread; otherwise, there is no way to terminate the DDEML session.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: UsrDde

Dealing w/ Lengthy Processing in Service Control Handler

PSS ID Number: Q120557
Authored 14-Sep-1994 Last modified 13-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, and 3.51

The service control handler function must return within 30 seconds. If it does not, the Service Control Manager will return the following error: "Error 2186 - The service is not responding to the control function".

If a service needs to do lengthy processing when the service is in the service control handler, it should create a secondary thread to perform the lengthy processing and then return. This prevents the service from tying up the service control handler thread.

If you are processing a SERVICE_CONTROL_STOP, you may wish to register a status of SERVICE_STOP_PENDING. The dwWaitHint should be at least 30 seconds. You can make the control panel applet wait for a long time if you send multiple SERVICE_STOP_PENDING states which update the dwCheckPoint and use a long dwWaitHint.

The system shutting down is another event that limits the service control handler. The dwCtrlCode parameter for the service control handler returns SERVICE_CONTROL_SHUTDOWN. A service then has approximately 20 seconds to perform cleanup tasks. If the tasks are not done, the system shuts down regardless if the service shutdown is complete. If the user has selected "restart", all processes will halt quickly. If instead the system is left in the "shutdown" state, the service processes continue to run.

If you need a longer time to shut down or earlier notification, consider using SetConsoleCtrlHandler() or SetProcessShutdownParameters() instead of using SERVICE_CONTROL_SHUTDOWN. This is the same mechanism that the Service Controller uses to get its notification.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseService

Debugging a Service with WinDbg

PSS ID Number: Q98890
Authored 18-May-1993 Last modified 27-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The following steps illustrate how to debug a service under Windows NT using WinDbg, which ships with the Win32 Software Development Kit (SDK). For illustration purposes, this procedure uses the SERVICE sample, which is built with debugging information by default. This sample is located in:

Win32 SDK 3.1:
MSTOOLS\SAMPLES\SERVICE

Win32 SDK 3.5:
MSTOOLS\SAMPLES\WIN32\SERVICE

Win32 SDK 3.51 and 4.0:
MSTOOLS\SAMPLES\WIN32\WINNT\SERVICE

MORE INFORMATION

1. Build the sample.

2. Install the Simple service with the following command:

   Win32 SDK 3.1:
   instsrv simple <drive>:\mstools\samples\service\simple.exe

   Win32 SDK 3.5 and later:
   simple - install

   Upon success, you will receive the message "CreateService Success".

3. Use the Control Panel's Services application to start the Simple service. With the Win32 SDK 3.5, you can also use

   sc start simpleservice

   The SC.EXE is located in MSTOOLS\BIN.

4. Use PView to get the process ID (PID) for the Simple service. For instance, if PView shows the process as simple(0xD5), then the PID is 0xD5.

5. If using SDK versions 3.1 or 3.5, convert the PID from hexadecimal to

decimal. For example, 0xD5 is 213 in decimal. Later versions of WinDbg use hexadecimal PIDs.

6. At a command prompt, go to the directory containing the sample and type

   ```
   start windbg
   ```

   to start WinDbg in its own command shell.

7. In WinDbg, choose Open from the File menu and open the source file (SIMPLE.C).

8. Set breakpoints at (for example) lines 326, 335, 337, 344, and 353. The lines will not change color at this point, but the breakpoints are successfully set.

9. Open a command window in WinDbg and type

   ```
   .attach <PID>
   ```

   Note that the lines where breakpoints are set will have changed colors.

10. Type "g" (a go command) in the WinDbg command window to restart after the thread that WinDbg uses to do the .attach terminates.

11. At the command prompt, start the client by typing

    ```
    Win32 SDK 3.1:
       client \\.\pipe\simple boo

    Win32 SDK 3.5 and later:
       client [-pipe <pipename>] [-string <string>]
    ```

12. Press F5 (a go command) to debug the service. The breakpoint hit will be on line 335. Press F5 again to go to the next breakpoint. Keep pressing F5 until line 326 waits again for a client to connect. Try connecting another client and repeat the same steps.

Exiting WinDbg will kill the service, which must be restarted manually with the Control Panel.

Services must log on as LocalSystem for this to work correctly; otherwise, you will not have permission to debug the service.

The manual translation of the PID from hexadecimal to decimal will not be needed in the future once the .attach command is modified so that it accepts hexadecimal.

NOTE: Under Windows NT 3.5 and later, because services do not have access to the user's desktop by default, one additional step must be taken to debug services with Windbg. In order to debug services, the user must check the startup option "Allow Service to interact with Desktop" in the Services control panel applet. You can do the same thing programmatically by setting HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\<DisplayName>\Type to

0x110.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsWindbg

Debugging a System-Wide Hook

PSS ID Number: Q102428
Authored 03-Aug-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

Debugging a system-wide hook such as a journal hook must be done with the extreme caution. When an application installs such a hook, it effectively takes control of user input. In effect, this disables the interface with the debugger. For example, after installing a journal record hook, you must unhook the record hook when you want to allow the debugger to regain control.

It is not possible to use an interactive debugger to debug an actively installed journal hook using a single machine. It is possible to use a remote debugger, because one interface can be blocked (or recording) while the other one does the debugging.

MORE INFORMATION

System-wide input hook procedures can be thought of as being in three possible states:

```
unhooked (not installed)
suspended
hooked (installed)
```

In the unhooked state, the procedure imposes no control over user input. In the hooked state, all user input specifically defined to be handled by this hook passes through this procedure. In the suspended state, all user input specifically defined to be handled by this hook is completely blocked.

In the case of a journal record hook, the suspended state can be achieved when a breakpoint is reached within the hook procedure. When this happens, all user input (system wide, that is) in the form of mouse and keyboard input is blocked, and thus you cannot interact with the debugger or any other application as you normally would. Fortunately, when the user presses the CTRL+ESC or the CTRL+ALT+DEL key combinations, all system-wide hooks are automatically unhooked, returning the system to the unhooked state.

Once this has occurred, it is likely that the application with the journal hook is now in a undefined state (because it had the hook pulled out from underneath it, so to speak). Fortunately, the system will send all applications the WM_CANCELJOURNAL message to indicate that it has removed the hook. A well behaved application can intercept this message and adjust its state accordingly.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrHks

Debugging Applications Under Win32s

PSS ID Number: Q102430
Authored 03-Aug-1993 Last modified 04-May-1995

The information in this article applies to:

- Microsoft Win32s versions 1.0, 1.1, and 1.2

SUMMARY

To start debugging a Win32-based application, make sure that it runs correctly under Windows NT. Use either WinDbg or NTSD to track down any problems.

Then, install the debugging libraries for Windows 3.1 and the debug version of Win32s. Hook the machine to another machine running a terminal emulator and watch for any warnings that are issued. Be sure to select the Win32sDebug flags carefully--selecting too many will generate more information than you may care to see; selecting too few may cause you to miss important information and warnings.

If you need to debug on Win32s, there are currently two options:

- Use wdeb386 (note that this method is very tricky).

- Use remote WinDbg (WDBG32S.EXE) if you are not familiar with using a kernel debugger. This method requires two machines: a Win32s machine to run the application and the remote debugger and a Windows NT machine to run WinDbg.

If you have Microsoft Visual C++ 32-bit Edition version 1.0, CodeView for Win32s is an additional option. CodeView for Win32s is a user-level debugger; remote debugging is not necessary, and therefore CodeView for Win32s does not require a second machine. CVW32S does not come with Visual C++ 2.0 and later. You can still use CVW32S.EXE with later versions of Visual C++, if you link with /PDB:none and /INCREMENTAL:no and if you do not use new features such as templates or C++ exception handling.

MORE INFORMATION

When performing remote debugging, make sure that the cable is set up exactly as specified in the Win32s Programmer's Reference. The remote WinDbg does not support software flow control, so it is very important that the hardware flow control is set up properly. If it is not set up correctly, you will have problems as the buffers overflow.

WinDbg supports XON/XOFF (software) flow control, which means that the standard 3-wire cable can now be used, although the default is still hardware handshaking (5-wire cable). To enable XON/XOFF, you must specify the XON flag in the serial transport parameters on both WinDbg and remote WinDbg.

To enable XON/XOFF in the remote WinDbg:

1. Select Options to bring up the Transport dynamic-link library (DLL) dialog box.

2. Select the serial transport and make any needed modifications to the communications port or baud rate parameters.

3. Place the XON flag at the end of the Parameters box. For example, "COM1:19200 XON". Note that the space is needed.

To enable XON/XOFF on WinDbg:

1. Select Options/Debug DLLs.

2. Select the proper serial transport layer.

3. Choose the Change button.

4. Add XON to the end of the Parameters line: "COM1:19200 XON".

It is very important that both sides of the debugger use the same protocol. If they do not, both debuggers will probably hang. Also, the remote debugging environment requires that binaries be located on the same drive/directory on both the development and target systems. For example, if WIN32APP.EXE is built from sources in a C:\DEV\WIN32APP directory, the binary should be located in this directory on both systems. If you build your source files by specifying fully qualified paths for the compiler, the compiler will place this information with the debug records which will allow WinDbg to automatically locate the appropriate source files.

For additional information on remote debugging, please see the "Win32s Programmer's Reference" which is included with the Win32 SDK.

Additional reference words: 1.00 1.10 1.20
KBCategory: kbprg
KBSubcategory: W32s

Debugging Console Apps Using Redirection

PSS ID Number: Q102351
Authored 03-Aug-1993 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

To redirect the standard input (STDIN) for a console application named APP.EXE from a file named INPUT.TXT, the following syntax is used:

```
app < input.txt
```

However, the following syntax will not work when attempting to debug this application using WinDbg with STDIN redirected:

```
windbg app < input.txt
```

To debug the application as desired, use

```
windbg cmd /c "app < input.txt"
```

MORE INFORMATION

This will allow WinDbg to debug whatever goes on in the cmd window. A dialog box will be displayed that says "No symbolic Info for Debuggee." This message refers to CMD.EXE; dismiss this dialog box. When the child process (APP.EXE) is started, the command window will read "Stopped at program entry point." To continue, type "g" at the command window. Note that APP.EXE will begin executing, then you can open the source file and set breakpoints.

This technique is also useful when debugging an application that behaves differently when run with a debugger than it does when it is run in the command window.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsWindbg

Debugging DLLs Using WinDbg

PSS ID Number: Q97908
Authored 25-Apr-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

This article describes the process of debugging dynamic-link libraries (DLLs) under WinDbg. As a further example, debugging File Manager extensions under Windows NT is discussed in the "More Information" section in this article.

MORE INFORMATION

The application and the DLL must be built with certain compiler and linker switches so that debugging information is included. These switches can be found in the $(cdebug) and $(ldebug) macros, respectively, which are defined in NTWIN32.MAK.

NOTE: It is important to disable optimization with -Od or locals will not be available in the locals window and line numbers may not match the source.

The application is loaded into WinDbg either by specifying "windbg <filename>" on the command prompt or by starting WinDbg from the program group and specifying <filename> in the Program Open dialog box (from the Program menu, choose Open). Note that <filename> is the name of the application, not the DLL. It is not necessary to specify the name of the DLL to be debugged.

The DLL is loaded either when execution of the application begins or dynamically through a call to LoadLibrary(). In the first case, simply press F8 to begin execution. All DLLs and symbolic information are loaded. To trace through the DLL code, breakpoints can be set in the DLL using a variety of methods:

- From the Debug menu, choose Breakpoints. The dialog box is Program Open.

  -or-

- Open the source file and use F9 or the "hand" button on the toolbar.

  -or-

- Go to the Command window and type:

```
bp[#] <Options>
```

<Options>:

addr	break at address
@line	break at line

In the case that the DLL is dynamically loaded, pressing F8 causes all other DLLs and symbolic information to load. The same methods described above can be used to set breakpoints; however, the user will get a dialog box indicating that the breakpoint was not instantiated. After the call to LoadLibrary() has been executed, all breakpoints are instantiated (it is possible to note the color change if the DLL source window is open) and will behave as expected.

To set a breakpoint in a DLL that is not loaded, specify the context when setting the breakpoint. The syntax for a context specifier is:

{proc, module, exe}addr

-or-

{proc, module, exe}@line

Example: {func, module.c, app.exe}0x50987. The first two parameters are optional, so {,,app.exe}0x50987 or {,,app.exe}func could be used instead.

For example, assume that we are trying to debug a File Manager extension under Windows NT that has been built with full debugging information. The procedure to debug the extension is as follows:

1. Open a Command window.
2. Start WinDbg WINFILE.
3. Set a breakpoint on FmExtensionProc().
4. At the Command window, type "g" and press ENTER. The debugger will continue executing the program form the point where it stopped (which could be from the beginning, at the breakpoint, and so on).

WinDbg will start WINFILE and when FmExtensionProc() is executed, WinDbg will break into the WINFILE process.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsWindbg

Debugging OLE 2.0 Applications Under Win32s

PSS ID Number: Q123812
Authored 11-Dec-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s version 1.2

SUMMARY

The following are available to help you debug your OLE 2.0 applications under Win32s:

- Debug versions of the OLE DLLs, included with Win32s. (See the Win32s Programmer's Reference for more information on the debug DLLs.)

- Failure/trace messages.

- The OLE SDK for Win32s, version 1.2, included on the Microsoft Developer Network (MSDN) CD.

For information on a utility that will convert OLE error codes into error message, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID: Q122957
TITLE : SAMPLE: DECODE32: OLE Error Code Decoder Tool

MORE INFORMATION

The debug version of OLE can send diagnostic information to the debug terminal. To enable this feature, include the following lines in the SYSTEM.INI:

```
[Win32sDbg]
ole20str=xxxxx
ole20str16=yyy
```

Use ole20str for 32-bit OLE and ole20str16 for 16-bit OLE. Set them to a combination of the following letters (case sensitive):

f - Failure message, kind of asserts.
v - Verbose. General purpose messages.
1 - Trace special translation activity for 32/16 interoperability.
i - Trace initialization of OLE.
t - Trace termination and cleanup of OLE.

The following tools are contained in the OLE SDK for Win32s, version 1.2:

- DFVIEW - Show the content of storage files.

- LRPCSPY - Monitor LRPC messages sent by 16-bit OLE applications (does not require Win32s).

- RPCSPY32 - Monitor LRPC messages from both 16-bit and 32-bit OLE applications.

- DOBJVIEW and DOBJVW32 - View objects placed on the clipboard as well as objects transferred by drag and drop.

- IROTVIEW and IROTVW32 - Display the contents of the OLE running object table (ROT).

- OLE2VIEW and OLE2VW32 - Identify objects, interfaces, inproc and local servers, registration database entries, and so on.

Additional reference words: 1.20
KBCategory: kbole kbprg
KBSubcategory: W32s

Debugging the Win32 Subsystem

PSS ID Number: Q105677
Authored 22-Oct-1993 Last modified 23-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

The instructions on page 1-18 of Part II of the Win32 "Programmer's Guide" included with the Win32 Device Driver Kit (DDK) says to use NTSD -d -p -1 to attach to the Win32 subsystem process and enable debugging of its user-mode drivers. This results in the error:

```
NTSD: cannot debug PID -1
error = 5
```

To enable this procedure to work properly, change the GlobalFlag value under:

```
HKEY_LOCAL_MACHINE\
            SYSTEM\
 CurrentControlSet\
          Control\
   Session Manager
```

Remove the flag 0x00080000 from 0x211a0000 to make it 0x21120000. The 0x00080000 flag disables the ability to debug CSRSS.EXE (the client server run time subsystem), which is specified by the "-p -1" parameter.

It is also possible to debug CSRSS using "WinDbgRm -c -p-1" instead of NTSD. Make sure that WinDbgRm defaults to debugging using TLPIPE.DLL as its transport layer, then run "windbgrm -c -p-1" on the debuggee.

On the debugger machine, make sure that CSRSS.EXE and any dynamic-link libraries (DLLs) that you are debugging in association with it are in the same directory, and run WinDbg. To set the transport DLL, choose Debug from the Options menu, choose Transport DLLs, and set the transport DLL to TLPIPE. Set the host name entries to be the machine name of the debuggee.

Additional reference words: 3.10 3.50
KBCategory: kbtool
KBSubcategory: TlsMisc

Debugging Universal Thunks

PSS ID Number: Q105756
Authored 24-Oct-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.0, 1.1, and 1.2

The general recommendation for an application targeted for Win32s is to debug it under Windows NT, then make sure that the application works under Win32s. However, Universal Thunks are not supported on Windows NT, so it is not possible to debug Win32-based applications that use the Universal Thunk in this manner.

To debug across the Universal Thunk, you can use WDEB386, which is available with the Windows 3.1 Software Development Kit (SDK). If you are not familiar with WDEB386, you may find it simpler to use other methods. In that case, be sure to install the debug version of Windows 3.1 and the debug version of Win32s and enable suitable notifications for Win32s (unimplemented functions and messages, verbose, and so forth). You may find OutputDebugString() useful for displaying extra information.

For more information on WDEB386, please see the Knowledge Base article "Tips On Installing WDEB386." For information on installing the debug version of Windows, please see your Windows SDK documentation.

Additional reference words: 1.00 1.10 1.20
KBCategory: kbprg
KBSubcategory: W32s

Default Attributes for Console Windows

PSS ID Number: Q90837
Authored 22-Oct-1992 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

Console attributes (screen fonts, screen colors, insert/overstrike, etc.) are stored by console title within each user's profile. When a profile for a new window is not found, the .DEFAULT configuration is used.

MORE INFORMATION

You can change the start font, screen colors, pop up colors, and insert/overstrike defaults using REGEDT32. Under the key HKEY_USERS, there is a .DEFAULT along with an entry for each user who has an account on that machine. Select DEFAULT\Console\Configuration. Use Edit.AddValue to add FontSize as a REG_DWORD to change the default font, Add ScreenColors and PopupColors as REG_DWORDs to change those defaults. To reset the console to be in insert mode rather than overstrike mode, add InsertMode as REG_SZ and set it to ON.

To get the right settings for the font size and colors you should first set your MS-DOS Prompt window font size and colors. Look up Console\MS-DOS Prompt\Configuration under your account and write down the values for the keys you need to add. Then go back to DEFAULT\Console\Configuration and add those values.

The DEFAULT configuration is read when the user chooses the Command prompt. However, if the user chooses to run a command shell via the File Manager (selecting CMD.EXE and choosing Run from the File menu), the DEFAULT configuration will not be read out of the registry.

WARNING:

RegEdit is a very powerful utility that facilitates directly changing the Registry Database. Using RegEdit incorrectly can cause serious problems, including hard disk corruption. It may be necessary to reinstall the software to correct some problems. Microsoft does not support changes made with RegEdit. Use this tool at your own risk.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseCon

Default Edit Control Entry Validation Done by Windows

PSS ID Number: Q74266
Authored 15-Jul-1991 Last modified 16-May-1995

-
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

-

Under the Microsoft Windows graphical environment, multiline and single-line edit controls do not accept characters with virtual key code values less than 0x20. The two exceptions are the TAB and ENTER keys; users can enter these characters only in a multiline edit control.

If an application creates an edit control with the ES_LOWERCASE or ES_UPPERCASE style, text entry is converted into the specified case.

If an application creates an edit control with the ES_OEMCONVERT style, the text is converted from the ANSI character set to the OEM character set and then back to ANSI for display in the control.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Default Stack in Win32-Based Applications

PSS ID Number: Q97786
Authored 21-Apr-1993 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1 and 3.5

SUMMARY

By default, space is reserved for applications in the following manner:

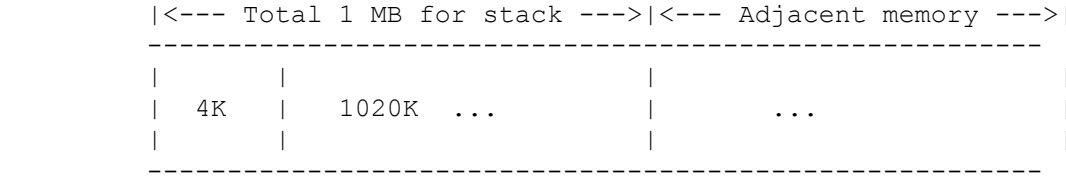
1 megabyte (MB) reserved (total virtual address space for the stack)

1 page committed (total physical memory allocated when stack is created)

Note: The -stack linker option can be used to modify both of these values.

The default stack size is taken from the process default reserve stack size.

The operating system will grow the stack as needed by committing 1 page blocks (4K on an x86 machine) out of the reserved stack memory. Once all of the reserved memory has been committed, Windows NT will attempt to continue to grow the stack into the memory adjacent to the memory reserved for the stack, as shown in the following example on an x86 machine:



However, once the stack grows to the point that the adjacent area is not free (and this may happen as soon as the reserved 1 MB has been committed), the stack cannot grow any farther. Therefore, it is very risky to rely on this memory being free. Applications should take care to reserve all the memory that will be needed by increasing the amount of memory reserved for the stack.

In other cases, it may be desirable to reduce the amount of memory reserved for the stack.

The /STACK option in the linker and the STACKSIZE statement in the DEF file can be used to change both the amount of reserved memory and the amount of committed memory. The syntax for each method is shown below:

/STACK:[reserve][,commit]

```
STACKSIZE [reserve][,commit]
```

MORE INFORMATION

Each new thread gets its own stack space of committed and reserved memory. CreateThread() has a stacksize parameter, which is the commit size. If a new size is not specified in the CreateThread() call, the new thread takes on the same stack size as the thread that created it, whether that be the default value, a value defined in the DEF file, or by the linker switch. If the commit size specified is larger than the default process stack size, the stack size is set to the commit size. When specifying a stack size of 0, the commit size is taken from the process default commit.

The system handles committing more reserved stack space when needed, but cannot reserve or commit more than the total amount initially reserved (or committed if no additional is reserved). Remember that the only resource consumed by reserving space is addresses in your process. No memory or pagefile space is allocated. When the memory is actually committed, both memory and pagefile resources are allocated. There is no harm in reserving a large area if it might be needed.

As always, automatic variables are placed on the stack. All other static data is located in the process address space. Because they are static, they do not need to be managed like heap memory.

Note that under Win32s 1.2 and earlier, stacks are limited to a maximum of 128K (this limit has been increased with Win32s 1.25a). The same stack is used on the 16-bit side of a Universal Thunk (UT). A 16:16 pointer is created and it points to the top of the 32-bit stack. The selector base is set in such a way that the 16-bit code is allocated at least an 8K stack.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseMm

Default/Private Dialog Classes, Procedures, DefDlgProc

PSS ID Number: Q68566
Authored 22-Jan-1991 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

The information below explains the differences between default and private dialog classes, their associated dialog procedures, and using the DefDlgProc() function.

This information is organized as a comparison between private and default dialog classes, covering class registration, dialog templates, dialog creation, and message processing.

Note that the source to the DefDlgProc() function is provided with the Windows Software Development Kit (SDK) version 3.0. The code is supplied on the Sample Source 2 disk in the DEFDLG.C file. By default, DEFDLG.C is placed into the \SAMPLES\DEFPROCS directory if the SDK installation program copies the sample source code.

There are many functions and macros used in DefDlgProc() that are internal to Windows and cannot be used by applications. No additional information is available on these functions and macros.

MORE INFORMATION

All dialog classes are window classes, just as all dialog boxes are windows. All dialog classes must declare at least DLGWINDOWEXTRA in the cbWndExtra field of the WNDCLASS structure before the dialog class is registered. The Windows Dialog Manager uses this area to store special information for dialog boxes.

The default dialog class is registered by Windows at startup. The window procedure for this class is known as DefDlgProc(), which is located in Windows's USER module. DefDlgProc() calls the application-provided dialog function, which returns TRUE if it processes a message completely or FALSE if DefDlgProc should process the message further.

If an application registers a private dialog class, it provides a window procedure for the dialog box. The window procedure is the same as that for any other application window and returns a LONG value. Messages that are not processed by this window function are passed to DefDlgProc().

Dialog Class Registration

Windows registers the default dialog class, which is represented by the value 0x8002. Windows uses this class when an application creates a dialog box using the DialogBox() or CreateDialog() functions, but specifies no class in the dialog resource template.

To use a private dialog class, the application must specify the fields of a WNDCLASS structure and call RegisterClass(). This is the same procedure that Windows uses to register the default dialog class.

In either case, the value in the cbWndExtra field of the WNDCLASS structure must contain a value of at least DLGWINDOWEXTRA. These bytes are used as storage space for dialog-box specific information, such as which control has the focus and which button is the default.

When a dialog class is registered, the lpfnWndProc field of the WNDCLASS structure must contain a function pointer. For the default dialog class, this field points to DefDlgProc(). For a private class, the field points to application-supplied procedure that returns a LONG (as does a normal window procedure) and passes all unprocessed messages to DefDlgProc().

Dialog Templates

Resource scripts are almost identical whether used with a default or a private dialog class. Dialog boxes using a private class must use the CLASS statement in the dialog template. The name given in the CLASS statement must match the name of class that exists (is registered) when the dialog box is created.

Dialog Creation and the lpfnDlgFunc Parameter

Applications create dialog boxes using the function DialogBox(), CreateDialog(), or one of the variant functions such as DialogBoxIndirect(). The complete list of functions is found on page 1-43 of the "Microsoft Windows Software Development Kit Reference Volume 1."

All dialog box creation calls take a parameter called lpfnDlgFunc, which can either be NULL or the procedure instance address of the dialog box function returned from MakeProcInstance(). When the application specifies a private dialog class and sets lpfnDlgFunc to a procedure instance address, the application processes each message for the dialog box twice. The message processing proceeds as follows:

1. Windows calls the dialog class procedure to process the message. To process a message in the default manner, this procedure calls DefDlgProc().

2. DefDlgProc() calls the procedure specified in the dialog box creation

call.

The procedure specified in lpfnDlgFunc must be designed very carefully. When it processes a message, it returns TRUE or FALSE and does not call DefDlgProc(). These requirements are the same as for any other dialog procedure.

Using a dialog procedure in conjunction with a private dialog class can be very useful. Processing for the private dialog class can be generic and apply to a number of dialog boxes. Code in the dialog procedure is specific to the particular instance of the private dialog class.

Dialog Message Processing

In dialog boxes with the default class, the application provides a callback dialog function that returns TRUE or FALSE, depending on whether or not the message was processed. As mentioned above, DefDlgProc(), which is the window procedure for the default dialog class, calls the application's dialog function and uses the return value to determine whether it should continue processing the message.

In dialog boxes of a private class, Windows sends all messages to the application-provided window procedure. The procedure either processes the message like any other window procedure or passes it to DefDlgProc(). DefDlgProc() processes dialog-specific functions and passes any other messages to DefWindowProc() for processing.

Some messages are sent only to the application-supplied procedure specified in the call to CreateDialog() or DialogBox(). Two examples of functions that Windows does not send to the private dialog class function are WM_INITDIALOG and WM_SETFONT.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDlgs

Defining Private Messages for Application Use

PSS ID Number: Q86835
Authored 19-Jul-1992 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

In the Microsoft Windows environment, an application can define a private message for its own use without calling the RegisterWindowMessage API. Message numbers between 0x8000 and 0xBFFF are reserved for this purpose.

For Windows NT and Windows 95, the system defines a new message WM_APP (value 0x8000). Applications can make use of the range WM_APP through 0xBFFF for private messages without conflict. The only requirement is that the .EXE file must be marked version 4.0 (use the linker switch /subsystem:windows,4.0). Windows NT 3.5 and 3.51 and Windows 95 will run applications marked version 4.0.

MORE INFORMATION

The documentation for the WM_USER message lists four ranges of message numbers as follows:

Message Number	Description
0 through WM_USER-1	Messages reserved for use by Windows.
WM_USER through 0x7FFF	Integer messages for use by private window classes.
0x8000 through 0xBFFF	Messages reserved for use by Windows.
0xC000 through 0xFFFF	String messages for use by applications.
Greater than 0xFFFF	Reserved by Windows for future use.

When an application subclasses a predefined Windows control or provides a special message in its dialog box procedure, it cannot use a WM_USER+x message to define a new message because the predefined controls use some WM_USER+x messages internally. It was necessary to use the RegisterWindowMessage function to retrieve a unique message number between 0xC000 and 0xFFFF.

To avoid this inconvenience, messages between 0x8000 and 0xBFFF were redefined to make them available to an application. Messages in this range do not conflict with any other messages in the system.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMsg

Definition of a Protected Server

PSS ID Number: Q102447
Authored 03-Aug-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

The Win32 application programming interface (API) reference briefly discusses creating a "protected server" that assigns security to private objects. This article explains the concept of a protected server" and its relationship to private objects.

MORE INFORMATION

A protected server is an application that provides services to clients. These services could be as simple as saving and retrieving information from a database while issuing security checks to verify that the client has proper access.

A private object is an application-defined data structure that both the client and server recognize. Private objects are not registered with nor recognized by the Windows NT operating system; they are entirely application-defined.

It is not uncommon for security to be assigned to private objects in a protected server's database. For example, when a client asks the server to create a new object in the database, the server could use the CreatePrivateObjectSecurity() Win32 API to create a security descriptor (SD) for the new private object. The server would then store the SD with the private object in the database. It is important to note that there is nothing in the SD that associates it with the private object. Instead, it is up to the protected server to maintain that association in the private object or in the database. It is likely that the private object and the associated SD would be stored together in a single database record.

A protected server application is responsible for checking a client's access before providing information. For example, when a client asks the server to retrieve some data, the server would go out and locate the record (which would contain the private object and SD) and bring a copy of the SD into memory. It would then call the AccessCheck() Win32 API passing the SD, the clients access token, and the desired access mask. AccessCheck() will check the client's access against the object's SD to determine if access is permitted. Depending on the result of AccessCheck(), the protected server would either provide the requested information or deny access.

In conclusion, a protected server is a application that performs operations on private objects that are entirely user defined. The protected server is

responsible for associating security descriptors to those objects and must take the steps necessary to verify a client's access.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

Detecting Closure of Command Window from a Console App

PSS ID Number: Q102429
Authored 03-Aug-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

Win32 console applications run in a command window. For the console application to detect when the console is closing, register a console control handler and look for the following values in your case statement:

CTRL_CLOSE_EVENT	User closes the console
CTRL_LOGOFF_EVENT	User logs off
CTRL_SHUTDOWN_EVENT	User shuts down the system

For an example, see the CONSOLE sample. For more information, see the entry for SetConsoleCtrlhandler() in the Win32 application programming interface (API) reference.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseCon

Detecting Data on the Communications Port

PSS ID Number: Q118625
Authored 25-Jul-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

To detect whether there is any data available on the communications (COM) port without calling ReadFile(), use the ClearCommError() API. The field COMSTAT.CbInQue contains the number of bytes not yet read by a call to ReadFile().

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseCommapi

Detecting Keystrokes While a Menu Is Pulled Down

PSS ID Number: Q35930
Authored 29-Sep-1988 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

In the Windows environment, there are two methods that an application can use to receive notification that a key is pressed while a menu is dropped down:

- The easier method is to process the WM_MENUCHAR message that is sent to an application when the user presses a key that does not correspond to any of the accelerator keys defined for the current menu.

- The other method is to use a message filter hook specified with the SetWindowsHook function. The hook function can process a message before it is dispatched to a dialog box, message box, or menu.

The hook functions and the WM_MENUCHAR message are documented in the Microsoft Windows SDK "Reference: Volume 1" for version 3.0 and in "Programmer's Reference, Volume 3: Messages, Structures, and Macros" for version 3.1.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrInp

Detecting Logoff from a Service

PSS ID Number: Q104122
Authored 07-Sep-1993 Last modified 21-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1 and 3.5

Sometimes it is handy for a service to know whether or not a user is logged on to the system. For example, suppose that there are circumstances under which the service displays a dialog box and waits for the user to respond. If this is done while the user is logged off, the service is blocked until the user logs on again.

Unfortunately, there is no direct way for a service to detect whether or not a user is logged on. There is, however, an indirect method. If you supply MB_DEFAULT_DESKTOP_ONLY as one of the flags in the fuStyle parameter of MessageBox(), the function will fail if no one is logged on or if a screen saver is running.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseService

Detecting the Presence of NetBIOS in Win32s

PSS ID Number: Q110844
Authored 31-Jan-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.1 and 1.2

The way to determine whether NetBIOS is present from a Win32 application running on Win32s is to issue an invalid NetBIOS command (such as 0xB2) and check that the return code is 0x3 (Illegal command).

Additional reference words: 1.10 1.20
KBCategory: kbprg
KBSubcategory: W32s

Detecting Windows NT from an MS-DOS-Based Application

PSS ID Number: Q100290
Authored 17-Jun-1993 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1 and 3.5

SUMMARY

There are calls that MS-DOS-based applications can make that are not supported under Windows NT. For example, calling Interrupt 25h to read the disk is not supported under Windows NT. Therefore, in some cases MS-DOS-based applications will need to know whether or not they are running under Windows NT.

Interrupt 21h, function 3306h can be used by MS-DOS-based applications to detect whether or not they are running under Windows NT. On return, registers BL and BH will contain the operating system major and minor numbers, respectively. If your application is running under Windows NT, the return will be:

```
BL = 5
BH = 50
```

MORE INFORMATION

Note that it is important to check both BL and BH, because MS-DOS 5.0 will also return a 5 in BL.

The following code demonstrates how to detect the operating system version from an MS-DOS-based application:

Sample Code

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>

void main()
{
   unsigned char cbh = 0;
   unsigned char cbl = 0;

   _asm {
      mov ax, 3306h
      int 21h
      mov cbh, bh
      mov cbl, bl
   }
```

```
   printf( "After int 21h\n" );
   printf( "%u, %u (bh, bl)\n", cbh, cbl );
}
```

Additional reference words: 3.10 3.50 determine
KBCategory: kbprg
KBSubcategory: SubSys

Detecting x86 Floating Point Coprocessor in Win32

PSS ID Number: Q124207
Authored 21-Dec-1994 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

In Windows NT (x86) and Windows 95, floating point is emulated by the operating system, in the event that a numeric coprocessor is not present. This allows Win32-based applications to be compiled with floating point instructions present, which will be trapped by the operating system at runtime in the event that a coprocessor is not present. This behavior is transparent to the application, so it is difficult to detect.

In some cases, it is useful to execute code based on the presence of a numeric coprocessor, so this article explains how to do it.

MORE INFORMATION

One approach you can use to detect whether a coprocessor is present is to read the CR0 (System Control Register). This is not possible from Ring 3 application code under Windows NT, so a different approach is outlined below.

To determine whether a coprocessor is present on a computer using the x86 platform running Windows NT, you need to determine if the registry entry HKEY_LOCAL_MACHINE\\HARDWARE\\DESCRIPTION\\System\\FloatingPointProcessor is present. If this key is present, a numeric coprocessor is present.

On the MIPS and Alpha platforms, this registry key is not present because floating point support is built-in. The following function indicates whether a numeric coprocessor is present on Windows NT. If the function returns TRUE, a coprocessor is present. If the function returns FALSE, and GetLastError() indicates ERROR_RESOURCE_DATA_NOT_FOUND, a coprocessor is not present. Otherwise, an error occured while attempting to detect for a coprocessor. Some error checking is omitted, for brevity.

```
BOOL IsCoProcessorPresent(void)
{
    #define MY_ERROR_WRONG_OS 0x20000000
    HKEY hKey;
    SYSTEM_INFO SystemInfo;

    // return FALSE if we are not running under Windows NT
    // this should be expanded to cover alternative Win32 platforms

    if(!(GetVersion() & 0x7FFFFFFF))
```

```
    {
        SetLastError(MY_ERROR_WRONG_OS);
        return(FALSE);
    }

    // we return TRUE if we're not running on x86
    // other CPUs have built in floating-point, with no registry entry

    GetSystemInfo(&SystemInfo);

    if((SystemInfo.dwProcessorType != PROCESSOR_INTEL_386) &&
       (SystemInfo.dwProcessorType != PROCESSOR_INTEL_486) &&
       (SystemInfo.dwProcessorType != PROCESSOR_INTEL_PENTIUM))
    {
        return(TRUE);
    }

    if(RegOpenKeyEx(HKEY_LOCAL_MACHINE,

"HARDWARE\\DESCRIPTION\\System\\FloatingPointProcessor",

                    0,
                    KEY_EXECUTE,
                    &hKey) != ERROR_SUCCESS)
    {
        // GetLastError() will indicate ERROR_RESOURCE_DATA_NOT_FOUND
        // if we can't find the key.  This indicates no coprocessor present
        return(FALSE);
    }

    RegCloseKey(hKey);
    return(TRUE);
}
```

Additional reference words: 3.10 3.50 4.00
KBCategory: kbprg kbcode
KBSubcategory: BseFltpt

Determining Available RGB Values of an Output Device

PSS ID Number: Q27225
Authored 03-Mar-1988 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The best way to determine the colors supported by a device is to enumerate the solid pens from a device context (DC) associated with that device. The EnumObjects function enumerates the pens supported by a specified device and calls a callback function for each pen. To do so, EnumObjects requires three parameters: a device context associated with the desired device as the hDC parameter, OBJ_PEN as the value of the nOjbectType parameter, and the procedure-instance address of a callback function as the lpObjectFunc parameter.

MORE INFORMATION

The first parameter for the callback function, lpLogObject, points to a LOGPEN data structure that describes each enumerated pen. When the lopnStyle field of the LOGPEN structure contains the PS_SOLID value, the application can retrieve and store the value of the lopnColor field.

NOTE: For true color devices (devices that support more than 8 bits-per-pixel of color resolution), Windows enumerates only a subset of the supported pens. These devices support almost any color.

The following code demonstrates the process described above:

```
int cMaxRGB, nCnt, nSolid; // Global variables for system RGB colors

void GetColorList(HWND hWnd)
{
   HDC hdc;
   FARPROC lpProcCallback;
   HANDLE hmem;
   BYTE FAR *lpmem;

   nCnt = nSolid = 0;
   hdc = GetDC(hWnd);
   cMaxRGB = GetDeviceCaps(hdc, NUMCOLORS);
   if (cMaxRGB >= 0x7FFF)
      return; // All colors available. Enumeration not required.

   lpProcCallback = MakeProcInstance(Callback, hInst);
```

```
   // Allocate space for color table
   hmem = GlobalAlloc(GHND, sizeof(COLORREF) * cMaxRGB);
   lpmem = GlobalLock(hmem);

   EnumObjects(hdc, OBJ_PEN, lpProcCallback, lpmem);

   FreeProcInstance(lpProcCallback);

   // Use the color table stored in the first nSolid entries of a
   // COLORREF array stored in lpmem.

   GlobalUnlock(hmem);
   GlobalFree(hmem);
   ReleaseDC(hWnd, hdc);
   return;
}
```

The source for the enumeration callback function is below. The callback function must be listed in the EXPORTS section of the module definition (DEF) file.

```
int FAR PASCAL Callback(LPLOGPEN lpLogPen, LPSTR lpData)
{
   nCnt++;
   if (lpLogPen->lopnStyle == PS_SOLID) // solid pen
      {
      COLORREF FAR *lpDest = (COLORREF FAR *)lpData + nSolid++;

      *lpDest = lpLogPen->lopnColor; // remember the solid color
      }

   if (nCnt >= cMaxRGB)
      return 0;
   return 1;
}
```

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiPal

Determining Selected Items in a Multiselection List Box

PSS ID Number: Q71759
Authored 01-May-1991 Last modified 16-May-1995

-
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

-

To obtain the indexes of all the selected items in a multiselection list box, the LB_GETSELITEMS message should be sent to the list box.

The message LB_GETCURSEL cannot be used for this purpose because it is designed for use in single-selection list boxes.

Another approach is to send one LB_GETSEL message for every item of the multiselection list box to get its selection state. If the item is selected, LB_GETSEL returns a positive number. The indexes can be built into an array of selected items.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Determining System Version from a Win32-based Application

PSS ID Number: Q92395
Authored 04-Nov-1992 Last modified 12-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0
- Microsoft Win32s, versions 1.2, 1.25a, and 1.3

SUMMARY

In order to create a Win32-based application that takes advantage of the features of each platform, it is necessary to determine the operating system on which the application is currently running.

You can use GetVersion() or GetVersionEx() to determine what operating system and version your application is running under. NOTE: GetVersion() is supported on Windows 3.1, but GetVersionEx() is new to the Win32 API. A Win32-based application might be running under MS-DOS/Windows using the Win32s extension, Windows NT Workstation, Windows NT Server, or Windows 95.

MORE INFORMATION

According to the documentation, the return value of GetVersion() is a DWORD that specifies the major and minor version numbers. GetVersionEx() uses members of the OSVERSIONINFO structure (dwMajorVersion and dwMinorVersion).

The following table shows the return values from GetVersion() under various environments:

Environment	LOWORD	HIWORD
Win32s on Windows 3.1	Windows version 3.1	RESERVED *
Windows NT 3.x	Windows version	RESERVED **
Windows 95	Windows version 4.0	RESERVED ***

  * The highest bit is 1. The remaining bits specify build number. Note that the version of MS-DOS cannot be determined as it can under Windows 3.x.
 ** The highest bit is 0. The remaining bits specify build number.
*** The highest bit is 1. The remaining bits are reserved.

The following sample code can be used to test the values returned by

GetVersion().

Sample Code 1

```
#include <windows.h>

void main()
{

   DWORD dwVersion;
   char szVersion[80];

   dwVersion = GetVersion();

   if (dwVersion < 0x80000000) {
   // Windows NT
       wsprintf (szVersion, "Microsoft Windows NT %u.%u (Build: %u)",
                 (DWORD)(LOBYTE(LOWORD(dwVersion))),
                 (DWORD)(HIBYTE(LOWORD(dwVersion))),
                 (DWORD)(HIWORD(dwVersion)));
   }
   else if (LOBYTE(LOWORD(dwVersion))<4) {
   // Win32s
       wsprintf (szVersion, "Microsoft Win32s %u.%u (Build: %u)",
                 (DWORD)(LOBYTE(LOWORD(dwVersion))),
                 (DWORD)(HIBYTE(LOWORD(dwVersion))),
                 (DWORD)(HIWORD(dwVersion) & ~0x8000));
   } else {
   // Windows 95
       wsprintf (szVersion, "Microsoft Windows 95 %u.%u (Build: %u)",
                 (DWORD)(LOBYTE(LOWORD(dwVersion))),
                 (DWORD)(HIBYTE(LOWORD(dwVersion))),
                 (DWORD)(HIWORD(dwVersion) & ~0x8000));
   }

   MessageBox( NULL, szVersion, "Version Check", MB_OK );
}
```

The following sample code can be used to test the values returned by GetVersionEx(). NOTE: The actual build number is derived by masking dwBuildNumber with 0xFFFF.

Sample Code 2

```
{
   OSVERSIONINFO osvi;
   char  szVersion [80];

   memset(&osvi, 0, sizeof(OSVERSIONINFO));
   osvi.dwOSVersionInfoSize = sizeof (OSVERSIONINFO);
   GetVersionEx (&osvi);

   if (osvi.dwPlatformId == VER_PLATFORM_WIN32s)
```

```
      wsprintf (szVersion, "Microsoft Win32s %d.%d (Build %d)",
                osvi.dwMajorVersion,
                osvi.dwMinorVersion,
                osvi.dwBuildNumber & 0xFFFF);

   else if (osvi.dwPlatformId == VER_PLATFORM_WIN32_WINDOWS)
      wsprintf (szVersion, "Microsoft Windows 95 %d.%d (Build %d)",
                osvi.dwMajorVersion,
                osvi.dwMinorVersion,
                osvi.dwBuildNumber & 0xFFFF);

   else if (osvi.dwPlatformId == VER_PLATFORM_WIN32_NT)
      wsprintf (szVersion, "Microsoft Windows NT %d.%d (Build %d)",
                osvi.dwMajorVersion,
                osvi.dwMinorVersion,
                osvi.dwBuildNumber & 0xFFFF);

   MessageBox( NULL, szVersion, "Version Check", MB_OK );
}
```

In order to distinguish between Windows NT Workstation and Windows NT Server, use the registry API to query the following:

```
   \HKEY_LOCAL_MACHINE\SYSTEM
   \CurrentControlSet
   \Control
   \ProductOptions
```

The result will be one of the following:

```
   WINNT      Windows NT Workstation is running.
   SERVERNT   Windows NT Server (3.5 or later) is running.
   LANMANNT   Windows NT Advanced Server (3.1) is running.
```

Additional reference words: 1.20 1.30 3.10 3.50 4.00 detect
KBCategory: kbprg
KBSubcategory: BseMisc

Determining the Maximum Allowed Access for an Object

PSS ID Number: Q115945
Authored 07-Jun-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

The AccessCheck() API call can be used to determine the maximum access to an object allowed for a subject. (In this article, a subject means a program running in a specific user's security context.) When using AccessCheck() for this purpose, perform the following steps:

1. Obtain a security descriptor that has owner, group, and DACL information.

2. If you are not impersonating a client, obtain an impersonation token by calling ImpersonateSelf. This token is passed as the client token in the AccessCheck() call.

3. Create a generic mapping structure. The contents of this structure will vary depending on the object being used.

4. Call AccessCheck() and request "MAXIMUM_ALLOWED" as the desired access.

If the AccessCheck() call succeeds after the above steps have been completed, the dwGrantedAccess parameter to AccessCheck() contains a mask of the object-specific rights that are granted by the security descriptor.

MORE INFORMATION

In most situations, you should not use this method of access determination. If you need access to an object to perform a task, simply try to open the object using the required access.

The AccessCheck() API is mainly intended for use with private objects created by an application. However, it can be used with predefined objects. The generic mapping values and specific rights for many of the predefined objects (files and so forth) may be found in WINNT.H.

REFERENCES

Please see the Security Overview in the "Win32 Programmer's Reference" and the "Win32 SDK API Reference" for more information.

Additional reference words: 3.10 3.50

KBCategory: kbprg
KBSubcategory: BseSecurity

Determining the Network Protocol Used By Named Pipes

PSS ID Number: Q126766
Authored 01-Mar-1995 Last modified 03-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 3.5

This article discusses how to determine or set the network protocol used by named pipes.

Named pipes are implemented using the server message block (SMB) redirector and server. As such, they can use whatever protocols are bound into the server and the client.

Both the redirector and the server maintain independent lists of transports that they are active on. The redirector contacts the remote server. The redirector will use the highest priority transport that both the client and the server support.

The priority for the transports is set using the Network control panel applet. Go into the Bindings button, select Workstation, and use the up and down buttons to rearrange the order that the redirector will use while trying to connect to the remote server.

Additional reference words: 3.50
KBCategory: kbprg
KBSubcategory: BseIpc

Determining the Number of Visible Items in a List Box

PSS ID Number: Q78952
Authored 05-Dec-1991 Last modified 16-May-1995

-
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

-

SUMMARY

To determine the number of items that are currently visible in a list box, an application must consider the following cases:

1. There are many items in the list box (some items are not visible).

2. There are few items in the list box (the bottom area of the list box is empty).

3. The heights of the items may vary (an owner-draw list box or use of a font other than the system default).

MORE INFORMATION

The following code segment can be used to determine the number of items visible in a list box:

Sample Code

```
int ntop, nCount, nRectheight, nVisibleItems;
RECT rc;

ntop = SendMessage(hwndList, LB_GETTOPINDEX, 0, 0);
                              // Top item index.

nCount = SendMessage(hwndList, LB_GETCOUNT, 0, 0);
                              // Number of total items.

GetClientRect(hwndList, &rc);
                              // Get list box rectangle.

nRectheight = rc.bottom - rc.top;
                              // Compute list box height.

nVisibleItems = 0;            // Initialize counter.

while ((nRectheight > 0) && (ntop < nCount))
                        // Loop until the bottom of the list box
```

```
                        // or the last item has been reached.
   {
   SendMessage(hwndList, LB_GETITEMRECT, ntop, (DWORD)(&itemrect));
                               // Get current line's rectangle.

   nRectheight = nRectheight - (itemrect.bottom - itemrect.top);
                               // Subtract current line height.

   nVisibleItems++;            // Increase item count.
   ntop++;                     // Move to the next line.
   }
```

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Determining the Topmost Pop-Up Window

PSS ID Number: Q66943
Authored 14-Nov-1990 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

When an application has many pop-up child windows (with a common parent window), the GetNextWindow() function can be used when one pop-up window is closed to determine the next topmost pop-up window that remains.

The following code fragment shows a window procedure for simple pop-up windows (modified from the PARTY program in Petzold's "Programming Windows"). In the WM_CLOSE case, the handle received by the pop-up window procedure is the handle of the pop-up to be closed. This sample activates the topmost pop-up window that remains by giving it the focus.

```
long FAR PASCAL PopupWndProc (hWnd, iMessage, wParam, lParam)
     HWND     hWnd;
     unsigned iMessage;
     WORD     wParam;
     LONG     lParam;
     {
     HWND     hWndPopup;

     switch (iMessage)
        {
        case WM_CLOSE:
            hWndPopup = GetNextWindow(hWnd, GW_HWNDNEXT);
            if (hWndPopup)
                SetFocus(hWndPopup);
            break;
        }

     return DefWindowProc (hWnd, iMessage, wParam, lParam) ;
     }
```

NOTE: In Windows 3.1, two messages are sent to an application when its Z-order is changing: WM_WINDOWPOSCHANGING and WM_WINDOWPOSCHANGED. When a window is closed (as in the example shown above) these two message will be sent to all window procedures.

For additional information on changing the Z-order of an MDI window, query on the following words in the Microsoft Knowledge Base:

   WM_WINDOWPOSCHANGED and MDICREATESTRUCT and WS_EX_TOPMOST

Additional reference word: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrWndw

Determining the Visible Area of a Multiline Edit Control

PSS ID Number: Q88387
Authored 24-Aug-1992 Last modified 27-Jun-1995

-
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

-

SUMMARY

The multiline edit control provided with Microsoft Windows versions 3.0 and 3.1 does not provide a mechanism that allows an application to determine the currently visible lines of text. This article outlines an algorithm to provide that functionality.

MORE INFORMATION

The general idea is to determine the first and last visible lines and obtain the text of those lines from the edit control. The following steps detail this process:

1. A Windows-3.1-based application can use the newly available message, EM_GETFIRSTVISIBLELINE, to determine the topmost visible line.

   A Windows-3.0-based application can use a technique described in the following Microsoft Knowledge Base article to determine the line number of the first visible line:

      ARTICLE-ID: Q68572
      TITLE : Caret Position & Line Numbers in Multiline Edit Controls

2. Obtain the edit control's formatting rectangle using EM_GETRECT. Determine the rectangle's height using this formula:

      nFmtRectHeight = rect.bottom - rect.top;

3. Obtain the line spacing of the font used by the edit control to display the text. Use the WM_GETFONT message to determine the font used by the edit control. Select this font into a display context and use the GetTextMetrics function. The tmHeight field of the resulting TEXTMETRIC structure is the line spacing.

4. Divide the formatting rectangle's height (step 2) with the line spacing (step 3) to determine the number of lines. Compute the line number of the last visible line based on the first visible line (step 1) and the number of visible lines.

5. Use EM_GETLINE for each line number from the first visible line to the last visible line to determine the visible lines of text. Remember that the last visible line may not necessarily be at the bottom of the edit control (the control may only be half full). To detect this case, use EM_GETLINECOUNT to know the last line and compare its number with the last visible line. If the last line number is less than the last visible line, your application should use EM_GETLINE only on lines between the first and the last line.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Determining Visible Window Area When Windows Overlap

PSS ID Number: Q75236
Authored 15-Aug-1991 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

There is no Windows API that reports the portion of an application's window that is not obscured by other windows. To determine which areas of the window are covered, it is necessary to walk through the window list managed by Windows.

Each window that precedes the application's window is "above" that window on the screen. Using the IntersectRect() function, check the rectangle of the window with any windows above to see if they intersect. Any window that is above the application's window and intersects its window rectangle obscures part of the application's window. By accumulating the positions of all windows that overlap the application's window, it is possible to determine which areas of the window are covered and which are not.

MORE INFORMATION

The following sample code demonstrates this procedure:

```
  GetWindowRect(hWnd, &rMyRect);    /*  Get the window dimensions
                                     *  for the current window.
                                     */
      /*  Start from the current window and use the GetWindow()
       *  function to move through the previous window handles.
       */
  for (hPrevWnd = hWnd;
      (hNextWnd = GetWindow(hPrevWnd, GW_HWNDPREV)) != NULL;
       hPrevWnd = hNextWnd)
    {
      /*  Get the window rectangle dimensions of the window that
       *  is higher Z-Order than the application's window.
       */
    GetWindowRect(hNextWnd, &rOtherRect);

      /*  Check to see if this window is visible and if intersects
       *  with the rectangle of the application's window. If it does,
       *  call MessageBeep(). This intersection is an area of this
       *  application's window that is not visible.
       */
    if (IsWindowVisible(hNextWnd) &&
          IntersectRect(&rDestRect, &rMyRect, &rOtherRect))
```

```
      {
      MessageBeep(0);
      }
    }
```

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrWndw

Determining Whether a WOW App is Running in Enhanced Mode

PSS ID Number: Q101893
Authored 26-Jul-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

Windows NT and Windows NT Advanced Server will run a Windows 3.1 application in 386 enhanced mode on X86 machines (standard mode on RISC machines). The proper way to determine whether a Windows 3.1 application is in enhanced mode is to call GetWinFlags() and do a bit test for WF_ENHANCED. This method is described on pages 486-487 in the Windows 3.1 Software Development Kit (SDK) "Programmer's Reference, Volume 2: Functions."

MORE INFORMATION

Calling Interrupt 2F with AX=1600h

This method, which is described in the Windows 3.1 Device Development Kit (DDK) checks to see whether a 386 memory manager is running. If Windows 3.1 is running in enhanced mode, it returns AL = 3 and 2 for standard mode. Windows NT's WOW (Windows 16 on Windows NT) returns AL=0, which means enhanced mode Windows is not running.

DWORD GetWinFlags()

The GetWinFlags() function retrieves the current Windows system and memory configuration.

The configuration returned by GetWinFlags() can be a combination of the following values:

Value	Meaning
WF_80x87	System contains an Intel math coprocessor.
WF_CPU286	System CPU is an 80286.
WF_CPU386	System CPU is an 80386.
WF_CPU486	System CPU is an i486.
WF_ENHANCED	Windows is running in 386-enhanced mode. The WF_PMODE flag is always set when WF_ENHANCED is set.
WF_PAGING	Windows is running on a system with paged memory.
WF_PMODE	Windows is running in protected mode. In Windows 3.1, this flag is always set.
WF_STANDARD	Windows is running in standard mode. The WF_PMODE

```
                flag is always set when WF_STANDARD is set.
   WF_WIN286    Same as WF_STANDARD.
   WF_WIN386    Same as WF_ENHANCED.
```

NOTE: When running in Windows NT, WF_WINNT will also be returned to tell the 16-bit Windows-based application that you are running in Windows NT.

Example:

The following example uses the GetWinFlags() function to display information about the current Windows system configuration:

Sample Code

```
int len;
char szBuf[80];
DWORD dwFlags;

dwFlags = GetWinFlags();

len = sprintf(szBuf, "system %s a coprocessor",
    (dwFlags & WF_80x87) ? "contains" : "does not contain");
TextOut(hdc, 10, 15, szBuf, len);

len = sprintf(szBuf, "processor is an %s",
    (dwFlags & WF_CPU286) ? "80286" :
    (dwFlags & WF_CPU386) ? "80386" :
    (dwFlags & WF_CPU486) ? "i486" : "unknown");
TextOut(hdc, 10, 30, szBuf, len);

len = sprintf(szBuf, "running in %s mode",
    (dwFlags & WF_ENHANCED) ? "enhanced" : "standard");
TextOut(hdc, 10, 45, szBuf, len);

len = sprintf(szBuf, "%s WLO",
    (dwFlags & WF_WLO) ? "using" : "not using");
TextOut(hdc, 10, 60, szBuf, len);
```

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: SubSys

Determining Whether an Application is Console or GUI

PSS ID Number: Q90493
Authored 15-Oct-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

In order to determine whether an application is console or GUI, you must parse the EXEheader. The header contains a field called 'Subsystem'. This field determines both the subsystem the application is to run under and the type of interface it requires. The values consist of:

IMAGE_SUBSYSTEM_NATIVE	1
IMAGE_SUBSYSTEM_WINDOWS_GUI	2
IMAGE_SUBSYSTEM_WINDOWS_CUI	3
IMAGE_SUBSYSTEM_OS2_CUI	5
IMAGE_SUBSYSTEM_POSIX_CUI	7

MORE INFORMATION

Sample Code

```
#include <windows.h>
#include <winnt.h>

VOID  main(int, char **);
DWORD AbsoluteSeek(HANDLE, DWORD);
VOID  ReadBytes(HANDLE, LPVOID, DWORD);
VOID  WriteBytes(HANDLE, LPVOID, DWORD);
VOID  CopySection(HANDLE, HANDLE, DWORD);

#define XFER_BUFFER_SIZE 2048

VOID
main(int argc, char *argv[])
{
    HANDLE hImage;

    DWORD  bytes;
    DWORD  iSection;
    DWORD  SectionOffset;
    DWORD  CoffHeaderOffset;
    DWORD  MoreDosHeader[16];

    ULONG  ntSignature;
```

```
IMAGE_DOS_HEADER      image_dos_header;
IMAGE_FILE_HEADER     image_file_header;
IMAGE_OPTIONAL_HEADER image_optional_header;
IMAGE_SECTION_HEADER  image_section_header;

if (argc != 2)
{
    printf("USAGE: %s program_file_name\n", argv[1]);
    exit(1);
}

/*
 *  Open the reference file.
 */
hImage = CreateFile(argv[1],
                    GENERIC_READ,
                    FILE_SHARE_READ,
                    NULL,
                    OPEN_EXISTING,
                    FILE_ATTRIBUTE_NORMAL,
                    NULL);

if (INVALID_HANDLE_VALUE == hImage)
{
    printf("Could not open %s, error %lu\n", argv[1], GetLastError());
    exit(1);
}

/*
 *  Read the MS-DOS image header.
 */
ReadBytes(hImage,
          &image_dos_header,
          sizeof(IMAGE_DOS_HEADER));

if (IMAGE_DOS_SIGNATURE != image_dos_header.e_magic)
{
    printf("Sorry, I do not understand this file.\n");
    exit(1);
}

/*
 *  Read more MS-DOS header.       */
ReadBytes(hImage,
          MoreDosHeader,
          sizeof(MoreDosHeader));

/*
 *  Get actual COFF header.
 */
CoffHeaderOffset = AbsoluteSeek(hImage, image_dos_header.e_lfanew) +
                   sizeof(ULONG);

ReadBytes (hImage, &ntSignature, sizeof(ULONG));
```

```
    if (IMAGE_NT_SIGNATURE != ntSignature)
    {
     printf("Missing NT signature. Unknown file type.\n");
     exit(1);
    }

    SectionOffset = CoffHeaderOffset + IMAGE_SIZEOF_FILE_HEADER +
                    IMAGE_SIZEOF_NT_OPTIONAL_HEADER;

    ReadBytes(hImage,
              &image_file_header,
              IMAGE_SIZEOF_FILE_HEADER);

    /*
     *  Read optional header.
     */
    ReadBytes(hImage,
              &image_optional_header,
              IMAGE_SIZEOF_NT_OPTIONAL_HEADER);

    switch (image_optional_header.Subsystem)
    {
    case IMAGE_SUBSYSTEM_UNKNOWN:
        printf("Type is unknown.\n");
        break;

    case IMAGE_SUBSYSTEM_NATIVE:
        printf("Type is native.\n");
        break;

    case IMAGE_SUBSYSTEM_WINDOWS_GUI:
        printf("Type is Windows GUI.\n");
        break;

    case IMAGE_SUBSYSTEM_WINDOWS_CUI:
        printf("Type is Windows CUI.\n");
        break;

    case IMAGE_SUBSYSTEM_OS2_CUI:
        printf("Type is OS/2 CUI.\n");
        break;

    case IMAGE_SUBSYSTEM_POSIX_CUI:
        printf("Type is POSIX CUI.\n");
        break;

    default:
        printf("Unknown type %u.\n", image_optional_header.Subsystem);
        break;
    }
}

DWORD
AbsoluteSeek(HANDLE hFile,
             DWORD  offset)
```

```
{
    DWORD newOffset;

    if ((newOffset = SetFilePointer(hFile,
                                    offset,
                                    NULL,
                                    FILE_BEGIN)) == 0xFFFFFFFF)
    {
        printf("SetFilePointer failed, error %lu.\n", GetLastError());
        exit(1);
    }

    return newOffset;
}

VOID
ReadBytes(HANDLE hFile,
          LPVOID buffer,
          DWORD  size)
{
    DWORD bytes;

    if (!ReadFile(hFile,
                  buffer,
                  size,
                  &bytes,
                  NULL))
    {
        printf("ReadFile failed, error %lu.\n", GetLastError());
        exit(1);
    }
    else if (size != bytes)
    {
        printf("Read the wrong number of bytes, expected %lu, got %lu.\n",
               size, bytes);
        exit(1);
    }
}
```

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseMisc

Determining Whether App Is Running as Service or .EXE

PSS ID Number: Q94994
Authored 28-Jan-1993 Last modified 13-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1 and 3.5

SUMMARY

When debugging Windows NT services, it may be necessary for the service application to be run interactively.

In this case, the application may determine whether it is being run as a service or as an executable (interactively) by checking GetLastError() after the call to StartServiceCtrlDispatcher() in the application's startup code.

If the application is being run as an executable, the call to GetLastError() will return with the following:

```
ERROR_FAILED_SERVICE_CONTROLLER_CONNECT
```

An alternative method is to check the return value from GetConsoleMode(). For example:

```
ret = GetConsoleMode (GetStdHandle (STD_OUTPUT_HANDLE), &mode);
```

Although the std handles may exist, they almost certainly will not be console handles unless there is a console attached. GetConsoleMode() will fail (with ERROR_INVALID_HANDLE) for non-console handles.

Sample Code

```
// Call StartServiceCtrlDispatcher() to set up the control
// interface. The API won't return until all services have been
// terminated. At that point, we just exit. See the
// StartServiceCtrlDispatcher() entry in Windows Help.

if (!StartServiceCtrlDispatcherW(ElfSvcDispatchTable) &&
   GetLastError() == ERROR_FAILED_SERVICE_CONTROLLER_CONNECT) {

   // Set a flag indicating you're running as an .EXE, not a service.

}
```

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseService

Determining Whether the User is an Administrator

PSS ID Number: Q118626
Authored 25-Jul-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

To determine whether or not a user is an administrator, you need to examine the user's access token with GetTokenInformation(). The access token represents the user's privileges and the groups to which the user belongs.

When a user starts an application, the access token associated with that process is the user's access token. To get the process token (and therefore the user's token), use OpenProcessToken().

The sample code below uses the APIs mentioned in the previous paragraph to test whether or not the current user is an administrator on the local machine:

Sample code

```
/* BOOL IsAdmin(void)

   returns TRUE if user is an admin
           FALSE if user is not an admin
*/

BOOL IsAdmin(void)
{
   HANDLE hProcess, hAccessToken;
   UCHAR InfoBuffer[1024];
   PTOKEN_GROUPS ptgGroups = (PTOKEN_GROUPS)InfoBuffer;
   DWORD dwInfoBufferSize;
   PSID psidAdministrators;
   SID_IDENTIFIER_AUTHORITY siaNtAuthority = SECURITY_NT_AUTHORITY;
   UINT x;

   hProcess = GetCurrentProcess();

   if(!OpenProcessToken(hProcess,TOKEN_READ,&hAccessToken))
      return(FALSE);

   if(!GetTokenInformation(hAccessToken,TokenGroups,InfoBuffer,
      1024, &dwInfoBufferSize)) return(FALSE);

   AllocateAndInitializeSid(&siaNtAuthority, 2,
      SECURITY_BUILTIN_DOMAIN_RID,
      DOMAIN_ALIAS_RID_ADMINS,
      0, 0, 0, 0, 0, 0,
      &psidAdministrators);
```

```
      for(x=0;x<ptgGroups->GroupCount;x++)
      {
         if( EqualSid(psidAdministrators, ptgGroups->Groups[x].Sid) )
         {
            FreeSid(&psidAdministrators);
            return( TRUE );
         }

      }
      FreeSid(&psidAdministrators);
      return(FALSE);
   }
```

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

Development Tools Do Not Accept Unicode Text

PSS ID Number: Q106065
Authored 31-Oct-1993 Last modified 24-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1 and 3.5

SUMMARY

Neither the Win32 SDK tools or Visual C++ (VC++) 32-bit Edition tools support Unicode text. In fact, the C/C++ Language specification says that the source files are to be written in 7-bit ANSI.

MORE INFORMATION

For example, language-specific resources cannot be specified in Unicode in the .RC file because RC does not accept the Unicode text. Although the message compiler has flags for Unicode, the flags are not implemented.

To convert to and from Unicode text, use the UCONVERT utility included in your MSTOOLS\BIN directory. The source for UCONVERT in the 3.1 SDK is in MSTOOLS\SAMPLES\SDKTOOLS\UCONVERT. The source for UCONVERT in the 3.5 SDK is in MSTOOLS\SAMPLES\SDKTOOLS\WINNT\UCONVERT.

The long term solution that Microsoft is working on are Resource Localization Tools and other methods that will allow the user to localize the strings in a GUI editor, running on the target machine.

Note that it is possible to specify Unicode escapes in L-quoted strings. The following is quoted from "Common Statement Parameters" in RC.HLP:

> By default, the characters listed between the double quotation marks are ANSI characters and escape sequences are interpreted as byte escape sequences. If the string is preceded by the L prefix, the string is a wide-character string and escape sequences are interpreted as two-byte escape sequences that specify Unicode characters. If a double quotation mark is required in the text, you must include the double quotation mark twice or use the \" escape sequence.

Another alternative is to use user-defined resources and include a binary (Unicode) file.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: WIntlDev

DEVMODE and dmSpecVersion

PSS ID Number: Q96282
Authored 14-Mar-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The dmSpecVersion field of the DEVMODE structure is intended for printer driver use only; no application programs should test this field. The purpose of this field is for new printer drivers to be able to recognize and handle DEVMODE structures created according to previous DEVMODE structure specification.

MORE INFORMATION

The DEVMODE structure is used for printer and (occasionally) display drivers when initializing. This structure is tied to the driver--not the operating system. The dmSpecVersion field does not allow an application to determine which platform (Windows version 3.1, Windows on Windows, Win32) the application is running in.

When an application fills a DEVMODE structure, it should set the dmSpecVersion field to DM_SPECVERSION. This identifies the version of the DEVMODE structure the application is generating.

If the application is querying to understand an unknown device, then special attention should be paid to the dmFields, dmSize, and dmDriverExtra fields. These fields are a reliable means of understanding what fields in the DEVMODE structure are readable.

The DEVMODE structure consists of public and private parts. The dmSpecVersion field applies to the public part. Any previously defined fields are not altered when the DEVMODE specification is updated--more fields are merely added to the end of the structure. This can mean fields used in the previous specification are ignored in a later specification. This functionality is managed by one bitfield describing what fields a driver actually uses. The new drivers just switch off the obsolete fields.

Applications using DEVMODE should always use the dmSize and dmDriverExtra fields for allocating/storing/manipulating the structure. These fields define the sizes of the public and private parts of the structure, respectively.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiPrn

Dialog Box Frame Styles

PSS ID Number: Q74334
Authored 16-Jul-1991 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

Dialog boxes can have either the WS_DLGFRAME or the WS_BORDER style. If a dialog box is created with both of these styles, it will have a caption bar instead of the expected frame and border. This is because WS_BORDER | WS_DLGFRAME is equal to WS_CAPTION.

To create a dialog box with a modal dialog frame and a caption, use DS_MODALFRAME combined with WS_CAPTION.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDlgs

DIB_PAL_INDICES and CBM_CREATEDIB Not Supported in Win32s

PSS ID Number: Q108497
Authored 13-Dec-1993 Last modified 05-May-1995

The information in this article applies to:

- Microsoft Win32s versions 1.0, 1.1, and 1.2
- Microsoft Win32 SDK (Windows 95 only)

The Windows NT implementation of the Win32 application programming interface (API) includes two new flags, DIB_PAL_INDICES and CBM_CREATEDIB, that can be used with various device-independent bitmap (DIB) APIs. These flags are not supported by Windows 3.1, Win32s, or Windows 95.

DIB_PAL_INDICES

In Windows NT, the DIB_PAL_INDICES flag can be used with the following APIs:

```
CreateDIBitmap()
CreateDIBPatternBrush()
CreateDIBPatternBrushPt()
SetDIBits()
GetDIBits()
SetDIBitsToDevice()
StretchDIBits()
```

When the dwUsage parameter is DIB_PAL_INDICES, the associated DIB does not have a color table. In this case, the bitmap bits are indices into the device palette.

Applications written to run on Windows 3.1 or Win32s should use DIB_PAL_COLORS or DIB_RGB_COLORS instead of DIB_PAL_INDICES.

Windows 95 does not support DIB_PAL_INDICES.

CBM_CREATEDIB

In Windows 3.1, CreateDIBitmap() creates a device-dependent bitmap (DDB) from a DIB definition and optionally initializes the DDB. In Windows NT, the CBM_CREATEDIB flag can be used with CreateDIBitmap() to create a new DIB instead of a DDB.

This functionality is not present in Windows 3.1 or Win32s.

Windows 95 does not support CBM_CREATEDIB. Equivalent functionality is provided in Windows 95 by CreateDIBSection().

Additional reference words: 1.00 1.10 1.20 4.00
KBCategory: kbprg
KBSubcategory: GdiBmp W32s

Differences Between hInstance on Win 3.1 and Windows NT

PSS ID Number: Q103644
Authored 26-Aug-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, 4.0

SUMMARY

In Microsoft Windows version 3.1, an instance handle can be used to uniquely identify the instance of an application because instance handles are unique within the scope of an address space. Because each instance of an application runs in its own address space on Windows NT, instance handles cannot be used to uniquely identify an instance of an application running on the system. This article explains why, and some alternative calls that might assist in uniquely identifying an application instance on Windows NT.

MORE INFORMATION

Although the concepts for an instance handle are similar on Windows NT and Windows 3.1, the results you see regarding them might be very different from what you expect.

With Windows 3.1, when you start several instances of the same application, they all share the same address space. You have multiple instances of the same code segment; however, each of these instances has a unique data segment associated with it. Using an instance handle (hInstance) is a way to uniquely identify these different instances and data segments in the address space.

Instance handles are unique to the address space. On Windows NT, when looking at the value of the instance handle, or the value returned from GetWindowLong(hWnd, GWL_HINSTANCE), a developer coming from a Windows 3.1 background might be surprised to see that most of the windows on the desktop return the same value. This is because the return value is the hInstance for the instance of the application, which is running it its own address space. (An interesting side note: The hInstance value is the base address where the application's module was able to load; either the default address or the fixed up address.)

On Windows NT, running several instances of the same application causes the instances to start and run in their own separate address space. To emphasize the difference: multiple instances of the same application on Windows 3.1 run in the same address space; in Windows NT, each instance has its own, separate address space. Using an instance handle to uniquely identify an application instance, as is possible on Windows 3.1, does not apply in Windows NT. (Another interesting side note: Remember that even if there are multiple instances of an application, if they are able to load at

their default virtual address spaces, the virtual address pages of the different applications' executable code will map to the same physical memory pages.)

In Windows NT, instance handles are not unique in the global scope of the system; however, window handles, thread IDs, and process IDs are. Here are some calls that may assist in alternative methods to uniquely identify instance of applications on Windows NT:

- GetWindowThreadProcessID() retrieves the identifier of the thread that created the given window and, optionally, the identifier of the process that created the window.

- OpenProcess() returns a handle to a process specified by a process ID.

- GetCurrentProcessID() returns the calling process's ID.

- EnumThreadWindows() returns all of the windows associated with a thread.

- The FindWindow() function retrieves the handle of the top-level window specified by class name and window name.

- To enumerate all of the processes on the system, you can query the Registry using RegQueryValueEx() with key HKEY_PERFORMANCE_DATA, and the Registry database index associated with the database string "Process".

For further details on using these calls, please see the Win32 SDK help file.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMisc

Differences Between the Win32 3.1 SDK and VCNT 1.0

PSS ID Number: Q105679
Authored 22-Oct-1993 Last modified 18-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 3.1
- Microsoft Visual C++ 32-bit Edition, version 1.0

SUMMARY

The following is a list of items included in the retail Win32 SDK that are not included in the 32-bit Edition of Visual C++ (Intel only):

MORE INFORMATION

Tools

- WinDbg/WinDbgRM
- Process Walker
- Working Set Tuner (WST)
- WinObj
- Setup Toolkit
- Microsoft Test
- Software Compatibility Test (SCT)
- API Profilers: CAP and WAP
- Symedit
- masm386
- Font Editor

Documentation

- LM API Reference (LMAPI.HLP)
- SNMP Programmer's Reference (SNMP.TXT)
- Generic Thunks (GENTHUNK.TXT)
- File Formats (CUSTCNTL.TXT, ENHMETA.TXT, PE.TXT, RESFMT.TXT)

Samples

- BNDBUF
- BOB (named EXITWIN in VC++)
- CDTEST
- CPL
- DYNDLG
- GUIGREP
- LARGEINT
- LOGGING
- LOWPASS
- MFEDIT
- MIDIMON
- MINREC
- MSGTABLE
- NTFONTS
- NTSD
- PDC
- RESDLL
- REVERSE

MANDEL	SCRNSAVE
MAPI	SEMAPHOR
MAZELORD	SPINCUBE
MCITEST	WINNET

Other

- Device Driver Kit (DDK) headers and libraries
- Checked build of Windows NT
- RPC Toolkit
- POSIX headers and libraries
- Microsoft Foundation Classes (MFC) 1.0

The following is a list of items included in the 32-bit Edition of Visual C++ that are not included in the retail Win32 SDK (Intel only):

Tools

- Visual Workbench/AppStudio/Wizards
- bscmake
- Pharlap TNT DOS-Extender
- Spy++
- Source Profiler
- CodeView for Win32S

Samples

- BOUNCE
- CVTMAKE
- EXITWIN (named BOB in the SDK)

Other

- MFC 2.0

The preliminary Win32 SDK contained the compiler tools, while the retail Win32 SDK does not. In addition, the preliminary Win32 DDK was available separately, while the retail DDK is bundled with the retail SDK.

The Win32 SDK has a separate "Win32s Programmer's Reference," while VC++ has the same chapters as part of the "Programming Techniques" manual.

There are tools whose names have changed and tools that are no longer needed. The SDK linker is LINK32, the VC++ linker is LINK. The SDK librarian is LIB32, the VC++ librarian is LIB. CVTRES existed in the SDK to convert the .RES file produced by RC so that it could be used by the linker, while VC++ has this functionality built into its linker. These changes may affect your makefiles.

For more information on switching from the Win32 SDK to 32-bit VC++,
see the VC++ file MIGRATE.HLP.

Additional reference words: 3.10
KBCategory: kbtool
KBSubcategory: TlsMisc

Differences Between the Win32 3.5 SDK and Visual C++ 2.0

PSS ID Number: Q125474
Authored 29-Jan-1995 Last modified 30-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 3.5
- Microsoft Visual C++ 32-bit Edition, version 2.0

SUMMARY

Microsoft Visual C++ version 2.0 contains the compiler tools, headers, and libraries necessary to develop Win32-based applications. In addition, it has an integrated development environment (Microsoft Foundation Classes (MFC) version 3.0) and Wizards to make programming easier. However, there is still a separate Win32 SDK available through MSDN Level II.

This article lists the things that are included in the Win32 SDK that are not included in Visual C++ version 2.0.

MORE INFORMATION

The following is a list of items included in the retail Win32 SDK that are not included in Visual C++ version 2.0 (Intel only):

Tools

WinDbg/WinDbgRM
Process Walker
Working Set Tuner
Software Compatibility Test
Microsoft Test
API Profilers
POSIX headers and libraries
Help Indexing
masm386

Toolkits

RPC Toolkit (and samples)
Setup Toolkit

Documentation

SNMP Programmer's Reference (PROGREF.RTF)
Generic Thunks (GENTHUNK.TXT)
Multicast Extensions to Windows Sockets for Win32
Windows Sockets for Appletalk

File Formats (CUSTCNTL.TXT, ENHMETA.HLP, PE.TXT, RESFMT.TXT)
Writing Great 32-bit Applications for Windows
POSIX Conformance Document
Microsoft Windows NT Version 3.5 Hardware Compatibility List

Additional Samples

Win32:

BNDBUF	NTSD		
CDTEST	RASBERRY		
CPL	REBOOT		
DYNDLG	RNR		
GLOBCHAT	SD_FLPPY		
INTEROP	SERVENUM		
IOCOMP	SIMPLEX		
IPXCHAT	SNMP		
LARGEINT	SOCKETS		
MANDEL	SPINCUBE		
MSGTABLE	WDBGEXTS		
NETDDE			

Multimedia:

AVIEDI32	MIXAPP		
AVIVIEW	MOVPLAY		
CAPTEST	MPLAY		
DSEQFILE	PALMAP		
ICMAPP	REVERSE		
ICMWALK	TEXTOUT		
LANGPLAY	WAVEFILE		
MCIPLAY	WRITEAVI		
MCIPUZZL			

SDK Tools:

ANIEDIT	RSHELL		
FONTEDIT	TLIST		
IMAGE	UCONVERT		
IMAGEDIT	WALKER		
NETWATCH	WINAT		
REMOTE			

Additional reference words: 2.00 3.50
KBCategory: kbtool
KBSubcategory: TlsMisc

Differentiating Between the Two ENTER Keys

PSS ID Number: Q77550
Authored 20-Oct-1991 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

An application may find it useful to differentiate between the user pressing the ENTER key on the standard keyboard and the ENTER key on the numeric keypad. Either action creates a WM_KEYDOWN message and a WM_KEYUP message with wParam set to the virtual key code VK_RETURN. When the application passes these messages to TranslateMessage, the application receives a WM_CHAR message with wParam set to the corresponding ASCII code 13.

To differentiate between the two ENTER keys, test bit 24 of lParam sent with the three messages listed above. Bit 24 is set to 1 if the key is an extended key; otherwise, bit 24 is set to 0 (zero). The contents of lParam for these messages is documented in the "Microsoft Windows Software Development Kit Reference Volume 1" for version 3.0 of the SDK and in the SDK Reference Volume 3, "Messages, Structures, and Macros."

Because the keys in the numeric keypad (along with the function keys) are extended keys, pressing ENTER on the numeric keypad results in bit 24 of lParam being set, while pressing the ENTER key on the standard keyboard results in bit 24 clear.

The following code sample demonstrates differentiating between these two ENTER keys:

```
   case WM_KEYDOWN:
      if (wParam == VK_RETURN)    // ENTER pressed
         if (lParam & 0x1000000L) // Test bit 24 of lParam
            {
            // ENTER on numeric keypad
            }
         else
            {
            // ENTER on the standard keyboard
            }
      break;
```

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbcode
KBSubcategory: UsrInp

Direct Drive Access Under Win32

PSS ID Number: Q100027
Authored 14-Jun-1993 Last modified 21-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

To open a physical hard drive for direct disk access (raw I/O) in a Win32-based application, use a device name of the form

```
\\.\PhysicalDriveN
```

where N is 0, 1, 2, and so forth, representing each of the physical drives in the system.

To open a logical drive, direct access is of the form

```
\\.\X:
```

where X: is a hard-drive partition letter, floppy disk drive, or CD-ROM drive.

MORE INFORMATION

You can open a physical or logical drive using the CreateFile() application programming interface (API) with these device names provided that you have the appropriate access rights to the drive (that is, you must be an administrator). You must use both the CreateFile() FILE_SHARE_READ and FILE_SHARE_WRITE flags to gain access to the drive.

Once the logical or physical drive has been opened, you can then perform direct I/O to the data on the entire drive. When performing direct disk I/O, you must seek, read, and write in multiples of sector sizes of the device and on sector boundaries. Call DeviceIoControl() using IOCTL_DISK_GET_DRIVE_GEOMETRY to get the bytes per sector, number of sectors, sectors per track, and so forth, so that you can compute the size of the buffer that you will need.

Note that a Win32-based application cannot open a file by using internal Windows NT object names; for example, attempting to open a CD-ROM drive by opening

```
\Device\CdRom0
```

does not work because this is not a valid Win32 device name. An application can use the QueryDosDevice() API to get a list of all valid Win32 device names and see the mapping between a particular Win32 device name and an

internal Windows NT object name. An application running at a sufficient privilege level can define, redefine, or delete Win32 device mappings by calling the DefineDosDevice() API.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseFileio

Disabling the Mnemonic on a Disabled Static Text Control

PSS ID Number: Q66946
Authored 14-Nov-1990 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

Disabling a static text control that contains a mnemonic does not prevent the control from responding to that key. For more information on static text controls with mnemonics, query in this Knowledge Base on the word "mnemonic".

To keep a static text control from processing mnemonics, it must be subclassed. When the control is disabled, WM_GETTEXT messages must be intercepted and the subclass procedure should return an empty string in response to the message.

MORE INFORMATION

When a static text control with a mnemonic is disabled using EnableWindow(), it turns gray but it does not stop responding to the mnemonic. This can cause problems because Windows processes the mnemonic by setting the focus to the next nonstatic, enabled control.

It is necessary to resort to subclassing to prevent the static control from processing the mnemonic. The subclass procedure should process the WM_GETTEXT message as follows:

```
...
// Windows asks the control, hChild, for its text.
case WM_GETTEXT:
    if (!IsWindowEnabled(hChild))
        {
        *(LPSTR)(lParam) = 0;    // A null terminated empty string
        return 0L;
        }
    break;
....
```

When the ALT key is held down and a key is pressed, Windows scans the text of each control to see if the key corresponds to a mnemonic. A WM_GETTEXT message is sent to each control. Normally, the control processes this message by returning its text. By returning an empty string in response to this message, Windows does not find the mnemonic.

Because the mnemonic must work when the control is enabled, the IsWindowEnabled() function is used to determine the state of the control.

If it is enabled, default processing occurs. Otherwise, the control is disabled and no text is returned, effectively disabling the mnemonic.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Displaying on the Screen What Will Print

PSS ID Number: Q22553
Authored 09-Oct-1987 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

It is possible for an application to closely simulate on the screen how an block of text will appear when printed. This article provides some references and techniques to accomplish this goal.

NOTE: WYSIWYG functionality can be attained in Windows 3.1 by using TrueType fonts. The following information should be used with non-TrueType fonts.

MORE INFORMATION

To create a what-you-see-is-what-you-get (WYSIWYG) screen image, the application must combine the services of the Windows graphics device interface (GDI), the printer, the display, and fonts. The following three references provide information that may be helpful:

1. "Pocket Pal -- A Graphic Arts Production Handbook" from International Paper Company; discusses issues related to type and typesetting.

2. "Bookmaking" by Marsha Lee; discusses issues related specifically to making books.

3. "Phototypesetting, A Design Manual" by James Craig.

These books contain information about type, fonts, and stringing characters together to make text. The following are three points to remember:

1. Printer manufacturers produce excellent fonts on their printers. Use printer fonts as much as possible.

2. GDI must provide some fonts for use with font-deficient display drivers.

3. Displays (even high-end ones) have much lower resolution than printers.

The best displays attached to microcomputers are approximately 120

dots-per-inch (dpi), while the typical laser printer is 300 dpi. To produce the same size image (for example, a 9-point capital A), the printer will illuminate/paint more of its pixels than will the screen. In particular, the printer will more closely match the requested size than the display. This leads to integer round-off errors.

Most graphics output devices are raster devices. Due to integer round-off errors associated with sampling the ideal "A" for different device resolutions, the origin of characters and words and the ends of lines on the display will seldom be the same as on a printer. For example, using a 75-dpi display and 300-dpi printer, the display might choose a 6 pixel-width for the character "A", while the printer might choose a 25 or 23 pixel-width for that same character. This mismatch necessitates adjusting the text on the display to match the output on the printer.

GDI provides various approaches to find the information needed to perform the adjustments. The following applications perform this task with increasing sophistication:

1. Windows Notepad application (does a less-than-ideal job)

2. Windows Write application (does better; however, the point at which various screen lines word wrap with different fonts is jagged)

3. Word for Windows (does an excellent job of handling text)

The first two applications are included with the Windows operating environment.

When Windows starts up, GDI asks each device whether it can support any fonts. For devices that provide some intrinsic (driver-based) fonts, GDI enumerates the available fonts and creates a table that describes them. When an application requests a font from GDI (using the CreateFont and SelectObject functions), GDI selects the font in its table that most closely matches the requested font. If no device font matches well, GDI will attempt to use one of its own fonts.

Ideally, the requested font will be available for all devices. More realistically, GDI provides a similar font, within the limits of the device capabilities.

The best way to imitate the appearance of printer fonts on a display is to assume that the printer has more fonts and greater resolution than the display. The following nine steps describe one way to implement a WYSIWYG display:

1. Open a device context (DC) and enumerate the fonts available on the printer. Use information from the GetDeviceCaps function with the TEXTCAPS parameter to determine how the device can alter the appearance of the fonts it provides. Together, the EnumFonts and GetDeviceCaps functions will allow the application to determine which fonts the device and GDI can provide. The text capabilities of the device serve as a filter in the enumeration process.

2. Provide a user interface in the application to allow the user to choose one of the fonts (this will result in "printer-centered WYSIWYG"). If appropriate, provide a method to choose between sizes and other attributes (bold, italic, and so forth). Fonts are most commonly selected by point size. One point equals 1/72 of an inch. Enumerating fonts returns LOGFONT and TEXTMETRIC structures. The quantities in these structures are in logical units that depend on the current mapping mode. Assuming that MM_TEXT (the default mapping mode) is selected, one logical unit equals one device unit (pixel, or pel). Font height and internal leading define the point size of the font as follows:

$$\text{point\_size} = \frac{72 * (\text{tmHeight} - \text{tmInternalLeading})}{\text{GetDeviceCaps(hDC, LOGPIXELSY)}}$$

3. Create a LOGFONT structure for the selected font. To ensure successful selection of that font into the printer DC, the lfHeight, lfWeight, and lfFacename fields must be specified. Weight and face name can be copied directly from the LOGFONT structure that was returned during the enumeration. The height should be computed using the following formula:

   ```
   lfHeight = -(point_size * GetDeviceCaps(hDC, LOGPIXELSY) / 72)
   ```

   In normal situations, set lfWidth to zero.

4. Once a font has been chosen, select it into the printer DC. Use GetTextMetrics and GetTextFace to verify the selection. If the steps above are followed, the process should fail only when very little memory is available in the system.

5. Create a device context for the display. Use the equation listed in step 2 above to compute the logical height for the font to be selected into the screen DC. Use CreateFont and SelectObject to select this logical font into the display DC, and use GetTextMetrics and GetTextFace to retrieve a TEXTMETRIC data structure and the face name for the selected font.

6. The font selected for the display is generally not the same as the font selected for the printer. To achieve WYSIWYG and the highest possible quality of the printed output, it is best to perform all page layout computations based on the metrics obtained from the printer DC, and force the output on the screen to match the printer output as closely as possible. This process generally causes some degradation of quality. It is assumed in the remainder of this discussion that text quality is most important.

7. Check whether either device supports the GDI escape codes that enhance the usability of fonts. One escape code, for example, returns the width table for proportionally spaced fonts. These escapes are listed in chapter 12 of the "Microsoft Windows Software Development Kit Reference, Volume 2." Use the escape code QUERYESCSUPPORT to discover whether a device supports a particular escape code. The width tables provide data to determine the

physical extent of character strings to be sent to the printer and how to match that extent on the display.

8. If the devices do not support those GDI escapes, select the desired font into a printer DC and use the GetTextExtent function to get the extent a string will occupy when printed.

9. With the methods outlined in steps 7 and 8, the dimensions of any string can be computed for the printer device. This information is used to compute page breaks and line breaks. Once the placement and extent of a string of text have been determined on the printed page, it is possible to create a scaled image on the screen. There are three methods of forcing a match between printer and screen:

   a. Take no special action. Put the text on the screen based on screen DC text metrics. With this method, only minimal matching is obtained.

   b. Use either the GetTextExtent function or the combination of the GetCharWidths, SetTextJustification, and SetTextCharacterExtra functions to create the same line breaks and justification on the screen as on the printed page. This method uses white space (usually the space character) to stretch or shrink a string of text (action of SetTextJustification) and adds a constant value to the width of every character in the font (action of SetTextCharExtra). This method achieves reasonable WYSIWYG.

   c. Use the ExtTextOut function and pass a width array to achieve the exact placement of each and every character in the string. This method provides the highest degree of WYSIWYG; however, it also requires character placement algorithms that do not degrade the speed of text output too much.

The following functions related to this article are documented in chapter 4 of the "Microsoft Windows Software Development Kit Reference, Volume 1:"

   CreateFont
   CreateFontIndirect
   EnumFonts
   Escape
   GetCharWidths
   GetDeviceCaps
   GetTextExtent
   GetTextFace
   GetTextMetrics
   SelectObject
   SetMappingMode
   SetViewportExtent
   SetViewportOrigin
   SetWindowExtent
   SetWindowOrigin

The LOGFONT and TEXTMETRIC data structures are documented in Chapter 7 of the SDK reference, volume 2.

The following device escape functions are documented in Chapter 12 of the SDK reference, volume 2:

   ENABLEPAIRKERNING
   ENABLERELATIVEWIDTHS
   EXTTEXTOUT
   GETEXTENDEDTEXTMETRICS
   GETEXTENTTABLE
   GETPAIRKERNTABLE
   GETTRACKKERNTABLE
   QUERYESCSUPPORT
   SETKERNTRACK

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiPrn

Distinguishing Between Keyboard ENTER and Keypad ENTER

PSS ID Number: Q96242
Authored 11-Mar-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

It is possible using ReadConsoleInput() or PeekConsole() to distinguish between the ENTER key on the main keyboard and the ENTER key on the numeric keypad. The KEY_EVENT_RECORD structure in the INPUT_RECORD structure must be used to distinguish between the two keys.

MORE INFORMATION

The following illustrates what the KEY_EVENT_RECORD structure is filled with after a keyboard ENTER key versus a numeric keypad ENTER key is pressed.

```
Keyboard ENTER Key
------------------

   KeyEvent.wRepeatCount     = 1
   KeyEvent.wVirtualKeyCode  = 13
   KeyEvent.wVirtualScanCode = 28
   KeyEvent.dwControlKeyState= 00000000

Numeric Keypad ENTER Key
------------------------

   KeyEvent.wRepeatCount     = 1
   KeyEvent.wVirtualKeyCode  = 13
   KeyEvent.wVirtualScanCode = 28
   KeyEvent.dwControlKeyState= 00000100
```

In the case of the numeric keypad ENTER key, in dwControlKeyState, the ENHANCED_KEY bit is set.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseCon

DLC Information on LLC_DIR_SET_MULTICAST_ADDRESS Command

PSS ID Number: Q129022
Authored 17-Apr-1995 Last modified 18-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

The DLC programming interface for Windows NT supports Ethernet multicast addressing via the LLC_DIR_SET_MULTICAST_ADDRESS command. This command must be successfully issued before Ethernet multicasts can be received. This article shows by example how to use the command.

MORE INFORMATION

The following function demonstrates the command.

Sample Code

```
BOOL DlcSetMulticastAddress( BYTE bAdapter, BYTE *pbResult, BYTE *pbAddress
)

{
    LLC_CCB Ccb;

    Ccb.uchAdapterNumber  = bAdapter;
    Ccb.uchDlcCommand     = LLC_DIR_SET_MULTICAST_ADDRESS;

    // note:  Dlc expects Ethernet addresses to be specified in the
    //        non-canonical form.  In other words, reverse the bits
    //        before passing an Ethernet address.
    //
    //        Also, the first byte of the canonical form of an
    //        Ethernet multicast address must be 0x01.

    Ccb.u.pParameterTable = (PLLC_PARMS) pbAddress;

    if(!DlcSyncCall( &Ccb ))
        return FALSE;
    else
    {
        *pbResult = Ccb.uchDlcStatus;
        return TRUE;
    }
}
```

```
// AcsLan wrapper function used by DlcSetMulticastAddress

BOOL DlcSyncCall( PLLC_CCB pCcb )
{
   BOOL  fResult = FALSE;
   DWORD dwResult;

   pCcb->hCompletionEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

   if (!pCcb->hCompletionEvent)
      return FALSE;

   int iStatus = (int) AcsLan( pCcb, NULL );
   if ( iStatus != ACSLAN_STATUS_COMMAND_ACCEPTED )
      goto done;

   dwResult = WaitForSingleObject( pCcb->hCompletionEvent, INFINITE );

   if ( dwResult == WAIT_OBJECT_0 )
      fResult = TRUE;

done:
   CloseHandle( pCcb->hCompletionEvent );

   return fResult;
}
```

Additional reference words: 3.10 3.50
KBCategory: kbnetwork kbprg kbcode
KBSubcategory: NtwkMisc

DlgDirList on Novell Drive Doesn't Show Double Dots [..]

PSS ID Number: Q99339
Authored 26-May-1993 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The Microsoft Windows application programming interface (API) function DlgDirList() can be used to add directories, drives, and/or files to a list box. On a Novell network with the default login script, the Windows API may not add the string "[..]" used to represent the parent directory to the list box. This is due to Novell implementation, and is not a bug in the Windows API. To make the entry appear, add the following line to the Novell login script (usually called SHELL.CFG located in the root directory of the boot drive)

```
SHOW DOTS = ON
```

MORE INFORMATION

A Novell NetWare file server does not include the directory entries dot (.) and double dot (..) as MS-DOS does. However, the NetWare shell (version 3.01 or later) can emulate these entries when applications attempt to list the files in a directory. Turning on Show Dots causes problems with earlier versions of some 286-based NetWare utilities, such as BINDFIX.EXE and MAKEUSER.EXE. Make sure you upgrade these utilities if you upgrade your NetWare shell. For more information, contact your Novell dealer.

NOTE: With Novell NetWare version 3.1.1, the line SHOW DOTS=ON/OFF can be added to the NET.CFG file for the same effect.

The same behavior is shown with the API DlgDirListComboBox, and the messages LB_DIR and CB_DIR.

Additional reference words: 3.10 3.50 3.51 4.00 95 listbox
KBCategory: kbprg
KBSubcategory: UsrCtl

Do Not Call the Display Driver Directly

PSS ID Number: Q77402
Authored 15-Oct-1991 Last modified 10-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK (Windows 95 only)

SUMMARY

In general, a Windows-based application cannot call the Windows display driver directly to perform graphics primitives. This article details the reasons this restriction is in place.

MORE INFORMATION

The Windows display driver communicates with the Graphics Device Interface (GDI) to perform primitive graphics operations. The parameters of the entry points (or exported functions) in the display driver are set up according to the standard interface between GDI and the display driver. The parameters passed by GDI to the display driver are only meaningful to GDI and to the display driver. A Windows-based application has no way to obtain these parameters. For example, the parameter most-commonly passed by GDI to the display driver is a pointer to a structure called PDEVICE. Memory for this structure is allocated by GDI, and its contents are specified by the display driver during the driver's initialization. The pointer to the PDEVICE structure is private to GDI; furthermore, the structure of PDEVICE varies among display drivers.

To give another example, when a primitive is to be done to a memory bitmap, instead of passing a pointer to PDEVICE, GDI passes to the display driver a pointer to a structure; the structure is usually referred to as a physical bitmap. Note that this physical bitmap structure is also called "BITMAP"; do not confuse it with the BITMAP structure defined in the Windows Software Development Kit. Again, this physical bitmap structure is not designed to be used by a Windows-based application. Although the information described in this structure is somewhat related to the bitmap that the application uses, the pointer to the physical bitmap structure is private to GDI and cannot be obtained by the application.

Additional reference words: 3.00 3.10
KBCategory: kbprg
KBSubcategory: GdiMisc

Do Not Forward DDEML Messages from a Hook Procedure

PSS ID Number: Q89828
Authored 30-Sep-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

If an application for Windows uses the Dynamic Data Exchange Management Library (DDEML) in addition to a message hook [for example, by calling SetWindowsHook() or SetWindowsHookEx()], it is possible that your hook procedure will receive messages that are intended for the DDEML libraries.

For the DDEML libraries to work properly, you must make sure that your hook function does not forward on any messages that are intended for the DDEML libraries.

MORE INFORMATION

If your hook procedure receives a code of type MSGF_DDEMGR, you should return FALSE instead of calling the CallNextHookEx() function.

The way to handle this situation is to use the following code:

```
if (MSGF_DDEMGR == code)
   return FALSE;
else
{
...
}
```

In cases where the callback function processes the message, it should return TRUE.

Note, however, how the message filter function is called from within DDEML:

```
while (TimeOutHasntExpired) {
    GetMesage (&msg, (HWND)NULL, 0, 0);
    if ( !CallMsgFilter (&msg, MSGF_DDEMGR))
         DispatchMessage (&msg);
 }
```

Given this, a callback function that just returns would cause the CallMsgFilter() call above to return TRUE, and never dispatch the message. This inevitably causes an infinite loop in the application, because GetMessage() ends up retrieving the same message over and

over, without dispatching it to the appropriate window for processing.

Therefore, a callback function that processes the message may not just return TRUE, but should also translate and dispatch messages appropriately.

The Windows 3.1 SDK's DDEMLCL sample demonstrates how to do this correctly in its MessageFilterProc() found in DDEMLCL.C:

```
   if (nCode == MSGF_DDEMGR) {

   /*
   * If a keyboard message is for MDI, let MDI client take care of it.
   * Otherwise, check to see if it is a normal accelerator key.
   * Otherwise, just handle the message as usual.
   */

       if ( !TranslateMDISysAccel (hWndMDIClient, lpmsg) &&
             !TranslateAccelerator (hWndFrame, hAccel, lpmsg))  {
             TranslateMessage (lpmsg);
             DispatchMessage (lpmsg);
         }
        return 1;
    }
```

For more information about message hooks and DDEML, please see the above mentioned functions in the Windows SDK manual or the online help facility.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDde

DOCERR: AddPort, ConfigurePort, DeletePort Fail Remotely

PSS ID Number: Q131223
Authored 06-Jun-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.5, 3.51, and 4.0

SYMPTOMS

Using AddPort, ConfigurePort, or DeletePort with the Universal Naming Convention (UNC) name of a remote server as the first parameter results in a return value of FALSE. This indicates that a failure has occurred. The error returned by GetLastError is ERROR_NOT_SUPPORTED. If the first parameter is the UNC name of the local machine or NULL, the functions succeed.

CAUSE

Calls to remote servers via AddPort, ConfigurePort and DeletePort are not supported. The documentation gives the impression that the UNC name for a remote server is permissible as the first parameter.

RESOLUTION

Use the UNC name to the local server or NULL as the first parameter.

MORE INFORMATION

These functions display a dialog associated with the hWnd that is supplied in the second parameter. Because the call displays a dialog box and requires a handle to a window in the second parameter, it is only supported locally.

Additional reference words: 3.50 4.00 95
KBCategory: kbprg kbdocerr
KBSubcategory: GdiPrn

DOCERR: BUFFER.CREATE Command Not Documented

PSS ID Number: Q123462
Authored 01-Dec-1994 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 3.5

SUMMARY

This article documents the BUFFER.CREATE command. This is a command that Microsoft added, so it is not documented in IBM LAN Technical Reference.

MORE INFORMATION

The AcsLan function topic in the Win32 Help file indicates that there are some differences between Windows NT DLC and the CCB2 interface documented in the IBM LAN Technical Reference. The most notable is that the buffer pool must be specified on an open adapter instance basis, not per SAP. After an adapter has been opened by using the DIR.OPEN.ADAPTER command, a buffer pool must be given to the DLC driver by using BUFFER.CREATE.

The IBM LAN Technical Reference manual has no reference to BUFFER.CREATE. However, the DLCAPI.H file contains the following structure with no other information explaining how to use it:

```
typedef struct {
    HANDLE hBufferPool;               // handle of new buffer pool
    PVOID pBuffer;                    // any buffer in memory
    ULONG cbBufferSize;               // buffer size in bytes
    ULONG cbMinimumSizeThreshold;     // minimum locked size
} LLC_BUFFER_CREATE_PARMS, *PLLC_BUFFER_CREATE_PARMS;
```

Unlike DLC as described in IBM LAN Technical Reference, Windows NT DLC does not support a buffer pool per SAP, but rather a buffer pool per process. All buffers given to the application are allocated from the same pool. (You can also allocate send buffers from the buffer pool, but this is not the case in general.)

The application gives the buffer pool to DLC by using the BUFFER.CREATE request. The parameters are:

- hBufferPool: the returned handle of the buffer pool. Supposedly, DLC can share buffer pools between processes but it doesn't, so it's useless.

- pBuffer: the pointer to application allocated buffer pool. This can be allocated by any scheme you wish and can be any size.

- cbBufferSize: the size of pBuffer.

- cbMinimumSizeThreshold: the minimum locked size.

Because the buffer is passed on kernel mode, it must be mapped into system space. It is divided into pages and further subdivided into 256-byte segments (unknown to the application). DLC tries to return as many contiguous segments as it can, but you may end up with a fragmented buffer, containing a relatively long chain of small segments. To your advantage, the largest DLC receive frame is usually around 4K.

DLC aligns the buffer to a page boundary, discarding any partial page buffer after the last page. This is not a problem unless you allocate a buffer less than a page, or one that is not page-aligned. In these cases, you are in danger of not having a buffer.

Because the buffer must be mapped to system space, DLC needs to lock the buffers into memory when receiving or sending from the buffers. To increase performance, DLC tries to keep a certain amount of buffer locked at all times. It will lock successive pages as the need arises. (In Windows NT, the smallest lockable region is a page.) You can control the initial locked area by setting the cbMinimumSizeThreshold value. If DLC does indeed lock your pages in excess of cbMinimumSizeThreshold, it will try to unlock them as they are freed up.

Additional reference words: 3.50
KBCategory: kbprg kbdocerr
KBSubcategory: NtwkMisc

DOCERR: CloseClipboard() Suggests Calling DuplicateHandle()

PSS ID Number: Q103240
Authored 19-Aug-1993 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

The documentation for CloseClipboard() suggests using DuplicateHandle() on a clipboard object before closing the clipboard, in order to use that object after the clipboard is closed. Doing so results in a return of ERROR_INVALID_HANDLE.

The Win32 SDK 3.1 documentation is in error. DuplicateHandle() is specified to work only on console input, console output, events, files, file mappings, mutexes, pipes, processes, semaphores, and thread handles.

The Win32 SDK 3.5 documentation has been corrected.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbdocerr
KBSubcategory: UsrClp

DOCERR: CreateFile() and Mailslots

PSS ID Number: Q131493
Authored 12-Jun-1995 Last modified 22-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 3.5

SUMMARY

The documentation for CreateFile() API gives incorrect possible return values while opening a client end of a mailslot. The documentation states:

> If CreateFile opens the client end of a mailslot, the function always returns a valid handle, even if the mailslot does not exist. In other words, there is no relationship between opening the client end and opening the server end of the mailslot.

Actually, CreateFile() returns INVALID_HANDLE_VALUE for a mailslot if the mailslot client is being created using the "\\." notation to communicate with a mailslot server on the local system when the server is not up and running.

MORE INFORMATION

The following code always returns INVALID_HANDLE_VALUE for a handle value from CreateFile() while opening the client end of the mailslot if the mailslot server is not up and running to read mailslot messages:

```
CreateFile("\\\\.\\mailslot\\testslot",
           GENERIC_WRITE,
           FILE_SHARE_READ,
           NULL,
           OPEN_EXISTING,
           FILE_ATTRIBUTE_NORMAL,
           NULL);
```

GetLastError() in this case returns Error 2: "The system cannot find the specified file."

This is an expected behavior. Windows NT implementation of local mailslots does not allow you to open the mailslot if the receiver has not created the server end with CreateMailslot() API.

Additional reference words: 3.50
KBCategory: kbprg kbnetwork kbdocerr
KBSubcategory: BseIpc

DOCERR: CS_PARENTDC Class Style Description Incorrect

PSS ID Number: Q111005
Authored 03-Feb-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The documentation relating to the CS_PARENTDC window class style in the Windows SDK versions 3.0 and 3.1 states that CS_PARENTDC "Gives the display context of the parent window to the window class." The documentation for the Win32 SDK contains a similar statement. These statements are incorrect.

MORE INFORMATION

A window with the CS_PARENTDC style bit will receive a regular device context (DC) from the system's cache of device contexts. CS_PARENTDC merely sets the clipping rectangle of the child to that of the parent so that the child can draw on the parent. It does not give the child the parent's DC or DC settings.

CS_PARENTDC is used with all of Windows's standard controls such as edit controls and list boxes because it can help improve performance when drawing, especially in a dialog box. For example, dialog box units are not exact and if a child is drawing in a very small space it might accidentally draw outside its own border. This style provides a little room for error and prevents the child from being clipped unexpectedly.

Additional reference words: 3.00 3.10 3.50 4.00 95 WNDCLASS RegisterClass
KBCategory: kbprg kbdocerr
KBSubcategory: UsrCls

DOCERR: DdeCreateDataHandle Documentation Errors

PSS ID Number: Q109131
Authored 22-Dec-1993 Last modified 10-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, version 3.1

The documentation for the cbInitData and offSrcBuf parameters to the DdeCreateDataHandle function are incorrect. Here is the description given for these two parameters in the current documentation:

cbInitData	Specifies the amount, in bytes, of memory to allocate for the global memory object. If this parameter is zero, the lpvSrcBuf parameter is ignored.
offSrcBuf	Specifies an offset, in bytes, from the beginning of the buffer pointed to by the lpvSrcBuf parameter. The data beginning at this offset is copied from the buffer to the global memory object.

These parameters should be documented as follows:

cbInitData	Specifies the amount, in bytes, of memory to copy from the source buffer specified by lpvSrcBuf.
offTargetBuf	Specifies an offset, in bytes, from the beginning of the global memory object* allocated by DdeCreateDataHandle. Data from the source buffer is copied to the global memory object, beginning at this offset. *The size of the global memory object allocated by DdeCreateDataHandle is equal to the sum of cbInitData and offTargetBuf.

Additional reference words: 3.10
KBCategory: kbprg kbdocerr
KBSubcategory: UsrDde

DOCERR: DdeCreateStringHandle() lpszString param

PSS ID Number: Q102570
Authored 04-Aug-1993 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.10
- Microsoft Win32 SDK versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The documentation for the DdeCreateStringHandle() function in the Windows 3.1 Software Development Kit (SDK) "Programmer's Reference, Volume 2: Functions" incorrectly states that the lpszString parameter may point to a buffer of a null-terminated string of any length, when the string is actually limited to 255 characters.

DDEML string-management functions are internally implemented using global atom functions. DdeCreateStringHandle() in particular, internally calls GlobalAddAtom(), and therefore inherits the same limitation as atoms to a maximum of 255 characters in length.

MORE INFORMATION

DDEML applications use string handles extensively to carry out DDE tasks. To obtain a string handle for a string, an application calls DdeCreateStringHandle(). This function registers the string with the system by adding the string to the global atom table, and returns a unique value identifying the string.

The global atom table table in Windows can maintain strings that are less than or equal to 255 characters in length. Any attempt to add a string of greater length to this global atom table will fail. Hence, a call to DdeCreateStringHandle() fails for strings over 255 characters long.

This limitation is by design. DDEML applications that use the DdeCreateStringHandle() function should conform to the 255-character limit.

This limitation has been preserved on the Windows NT version of DDEML for compatibility reasons.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbdocerr
KBSubcategory: UsrDde

DOCERR: DeviceIoControl Requires OVERLAPPED Struct w/ Async.

PSS ID Number: Q126282
Authored 19-Feb-1995 Last modified 22-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1 and 3.5

The documentation for DeviceIoControl is incomplete. It should warn you that unexpected behavior may occur when both the following are true:

- A DeviceIoControl request is followed by another DeviceIoControl request on the same handle.

- There was no OVERLAPPED structure passed into DeviceIoControl after the handle to the device or file was opened with FILE_FLAG_OVERLAPPED.

The following is a list of some of the unexpected behavior that may occur:

- Requests complete before expected.

- There's undefined data in the returned buffers.

- Unknown error codes are returned.

- IRPs are held in a driver while the DeviceIoControl call has already returned.

When a handle is requested to a driver or file with FILE_FLAG_OVERLAPPED specified, the executive prepares itself for all requests on that handle to be asynchronous. Usually, when the request is sent down with an OVERLAPPED structure, an event is placed in that OVERLAPPED structure. This event is then stored in the FILE_OBJECT in kernel mode to use later to signal the user-mode application when that IRP has been completed. If an event is not specified, the value will be 0 in the FILE_OBJECT, when a second request is sent down before the first request completes (and completes), the IO manager will not have separate signals for the completion of the requests. Therefore, the request will appear to be completed, while in reality the IRP has not been completed by the underlying driver.

Please see the current documentation of GetOverlappedResult for more information on the behavior of the executive when no OVERLAPPED structure is specified when the handle to the driver or file was opened with FILE_FLAG_OVERLAPPED.

Additional reference words: 3.10 3.50 FILE_FLAG_OVERLAPPED DDK EVENT
KBCategory: kbprg kbdocerr
KBSubcategory: BseFileio

DOCERR: DEVMODE dmPaperSize Member Documentation Error

PSS ID Number: Q108924
Authored 20-Dec-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

The dmPaperSize member of the DEVMODE structure is documented incorrectly. The documentation states that the dmPaperSize member may be set to zero if the length and width of the paper are specified by the dmPaperLength and dmPaperWidth members, respectively. However, the correct value to use for user-defined paper sizes is DMPAPER_USER.

DMPAPER_USER is correctly listed in the Microsoft Windows 3.1 SDK documentation as meaning a user-defined paper size, but is completely omitted from the Microsoft Windows 3.0 SDK documentation.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg kbdocerr
KBSubcategory: GdiPrn

DOCERR: DragQueryFile() Return Code Can Be Misleading

PSS ID Number: Q115081
Authored 18-May-1994 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, 4.0

SYMPTOMS

The documentation for DragQueryFile() in the "Win32 Programmer's Reference" indicates that the return value is "the count of the characters copied" or "the required size, in characters, of the buffer". However, the actual return values do not include the trailing null character in this count.

RESOLUTION

The correct description for the return value is "the length of the string returned, not including the terminating null character" or "the length of the string that would be returned, not including the terminating null character".

The Win32 SDK 3.5 documentation has been corrected.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbref kbdocerr
KBSubcategory: UsrDnd

DOCERR: EofChar Field of DCB Structure Is Not Supported

PSS ID Number: Q115083
Authored 18-May-1994 Last modified 28-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SYMPTOMS

The documentation for the device control block (DCB) structure says that the EofChar field specifies the value of the character used to signal the end of the data over the communications resource. However, this setting seems to have no effect.

RESOLUTION

DCB.EofChar is not currently supported in Windows NT.

MORE INFORMATION

When DCB.EofChar is supported, you should call ClearCommError() after every read to see whether COMSTAT.fEof is set. When COMSTAT.fEof is set, this means that DCB.EofChar has been received.

Additional reference words: 3.10 3.50
KBCategory: kbref kbdocerr
KBSubcategory: BseCommapi

DOCERR: Error 87 When NetMessageBufferSend Attempts Broadcast

PSS ID Number: Q129289
Authored 24-Apr-1995 Last modified 08-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5

SUMMARY

The documentation incorrectly states that applications may use NetMessageBufferSend() to send broadcast messages on the network. This API does not support broadcasts. If an application calls the NetMessageBufferSend() API with the second parameter set to "L"*"", the API fails with error 87 (Invalid Parameter). The "*" parameter is used to broadcast a message to all the machines on the network.

MORE INFORMATION

Programmers can specify the domain name as the second parameter to send broadcasts on a particular domain. For example:

```
   NetMessageBufferSend(NULL, // Run command on local machine.
              L"MYDOMAIN*",   // Domain name.
              L"MYNAME",      // Sender of message.
              buffer,         // Buffer to send.
              buflen);        // Size of buffer.
```

If broadcasts are necessary, use the NetServerEnum() or WNetOpenEnum() API in conjunction with the WNetEnumResource() API to list domains, and use the NetMessageBufferSend() API to send a message on each domain as shown above.

REFERENCES

"Message APIs" help file in the Windows Networking API Definitions section of the online "Win32 SDK Programmer's Reference."

Additional reference words: 3.50
KBCategory: kbnetwork kbdocerr
KBSubcategory: NtwkLmapi

DOCERR: Errors in Win32s Compatibility

PSS ID Number: Q113679
Authored 11-Apr-1994 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32s versions 1.2 and 1.25a

SUMMARY

The following Win32 APIs are not listed in the Win32 Programmer's Reference as being supported under Win32s. This is incorrect. These Win32 APIs are supported under Win32s:

FlushInstructionCache
GetFileInformationByHandle
MoveMemory
TlsAlloc
RegCreateKeyEx
RegOpenKeyEx
RegSetValueEx
WNetAddConnection
WNetGetConnection

In addition, all of the BS_ button styles are marked as not supported under Win32s. However, Win32s supports all of the button styles that Windows 3.1 supports.

MORE INFORMATION

The following Win32 APIs and CRT (C Run-time) routines are supported under Win32s, with the restrictions noted.

Function	Associated Restriction
CreateFile	FILE_FLAG_NO_BUFFERING, FILE_FLAG_WRITE_THROUGH, FILE_FLAG_OVERLAPPED, FILE_DELETE_ON_CLOSE, and the security flags are not supported. Passing dwShareMode=0 opens the file in compatibility mode, not in exclusive mode.
CreateFileMapping	SEC_COMMIT is supported, but not SEC_IMAGE, SEC_NOCACHE or SEC_RESERVE. The size of named shared memory is limited by Windows memory.
CreateDIBitmap CreateDIBPatternBrush CreateDIBPatternBrushPt	DIB_PAL_INDICES DIB_PAL_PHYSINDICES DIB_PAL_LOGINDICES are not supported.

GetDIBits SetDIBits SetDIBitsToDevice StretchDIBits	
CreatePolyPolygonRgn	The polygon must be closed.
CreateProcess	Handle inheritance is not supported. Only creation flags supported are DEBUG_PROCESS and DEBUG_ONLY_THIS_PROCESS. The only priority supported is NORMAL_PRIORITY_CLASS. The security attributes are ignored.
GetFileInformationByHandle	ftCreationTime, ftLastAccessTime, dwVolumeSerialNumber, nFileIndexHigh, and nFileIndexLow are all 0.
GetFileTime SetFileTime	Only lpLastWriteTime is supported.
GetPrivateProfileString GetProfileString	lpszSection parameter cannot be NULL.
GetSystemMetrics	SM_CMOUSEBUTTONS is not supported.
GetSystemPaletteEntries	The fourth parameter cannot be NULL.
GetVolumeInformation	Volume ID is not supported.
OpenProcess	Settings for fdwAccess and fInherit are ignored. In the implementation, fdwAccess=PROCESS_ALL_ACCESS and fInherit=TRUE.
PeekMessage	hWnd cannot be -1.
PlaySound	SND_ALIAS, SND_FILENAME, and SND_NOWAIT are not supported.
RegSetValueEx	Only supports the REG_SZ entries.
SetClipboardData	Use only a global handle.
SetWindowsHookEx	dwThreadId parameter is ignored.
signal	SIGBREAK is not supported.
spawn	P_WAIT is not supported.
WaitForDebugEvent	Any dwTimeOut other than 0 is treated as INFINITE.
WNetAddConnection	**Password cannot be NULL, but can be "".**

===

Additional reference words: 1.20
KBCategory: kbref kbdocerr
KBSubcategory: W32s

DOCERR: GetWindowPlacement Function Always Returns an Error

PSS ID Number: Q89569
Authored 27-Sep-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SYMPTOMS

When an application developed for the Microsoft Windows graphical environment uses the GetWindowPlacement() function to retrieve the show state and position information for a window, the function always returns FALSE, indicating an error.

CAUSE

The length member of the specified WINDOWPLACEMENT data structure is not initialized.

RESOLUTION

Initialize the length member and call the GetWindowPlacement() function as follows:

```
BOOL             bResult;
WINDOWPLACEMENT  lpWndPl;

lpWndPl.length = sizeof(WINDOWPLACEMENT);
bResult = GetWindowPlacement(hWnd, &lpWndPl);
```

MORE INFORMATION

The need to initialize the length member of the WINDOWPLACEMENT structure is documented on page 422 of the Microsoft Windows Software Development Kit (SDK) "Programmer's Reference, Volume 3: Messages, Structures, and Macros" manual for version 3.1. This information is not listed in the documentation for the GetWindowPlacement() function on page 479 of the "Programmer's Reference, Volume 2: Functions" manual.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbdocerr
KBSubcategory: UsrWndw

DOCERR: GLYPHMETRICS Is in Device Units Not Logical Units

PSS ID Number: Q126141
Authored 15-Feb-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), version 3.1
- Microsoft Win32s version 1.2
- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

The units of the values in the GLYPHMETRICS structure are documented in the "Win32 Programmer's Reference vol. 5" page 425 and the "Windows 3.1 SDK Programmer's Reference vol. 3" page 297 as being in logical units. This is incorrect. The values in the GLYPHMETRICS structure returned by the GetGlyphOutline function call are in device units, not in logical units.

Additional reference words: 1.20 3.10 3.50 4.00 95
KBCategory: kbprg kbdocerr
KBSubcategory: GdiFnt

DOCERR: Important Information on RPC 1.0 Missing from the

PSS ID Number: Q129142
Authored 18-Apr-1995 Last modified 19-Apr-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows for Workgroups version 3.11
- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SUMMARY

This article documents important information about Microsoft RPC version 1.0 that is not included in the Microsoft RPC programming documentation or in Help. This information is identical to the information contained in the file RPCREAD.ME included with the Win32 SDK for Windows NT version 3.1.

MORE INFORMATION

Introduction

Microsoft RPC version 1.0 is a toolkit for developing network-aware distributed applications in C/C++. The RPC toolkit includes:

- MIDL compilers for Microsoft Windows NT and Microsoft Windows 3.x.
- C/C++ language header files (.H) and run-time libraries (.LIB and .DLL) for Microsoft Windows NT, Microsoft Windows 3.x, and MS-DOS.
- Sample programs for Microsoft Windows NT, Microsoft Windows 3.x, Microsoft Windows for Workgroups, and MS-DOS.
- RPC reference Help files, Windows Write files, and PostScript files.

The Win32 SDK contains the Microsoft Windows NT and Microsoft Windows 3.x versions of the RPC SDK.

Installation

The Microsoft Win32 SDK installs the components of the Microsoft RPC toolkit as part of its standard installation. No additional installation is required.

To develop client-side distributed applications for MS-DOS, Microsoft Windows 3.x, and Windows for Workgroups version 3.1, you must install the Microsoft Windows 3.x/MS-DOS version of the RPC toolkit. Cross-compilation of Windows version 3.x and MS-DOS clients with Microsoft Windows NT requires a 16-bit C compiler in the Microsoft Windows NT environment. This

development environment is not installed during RPC SDK setup. The Microsoft RPC toolkit for MS-DOS and Microsoft Windows 3.x requires:

- A 16-bit compiler such as Microsoft Visual C++ Development System for Windows or the Microsoft C/C++ 7.0 compiler.

- One of the following:

    - Microsoft Windows for Workgroups version 3.1 (or later) with named pipes, NetBIOS, or TCP/IP.

      -or-

    - Microsoft LAN Manager version 2.1 (or later) with named pipes, NetBIOS, or TCP/IP.

      -or-

    - DEC PATHWORKS 4.0 (or later) with NetBIOS, TCP/IP, or DECNet.

      -or-

    - Novell Netware 3.x with SPX.

      -or-

    - Other networking software compatible with the Windows Sockets API standard.

To install the Microsoft Windows 3.x/MS-DOS version of the RPC toolkit, run the Setup program in the directory MSTOOLS\RPC_DOS. To start the Setup program, choose the Run command from the File menu in the Microsoft Windows 3.x Program Manager.

When you install the RPC toolkit in a directory different from the directory you used for Microsoft C/C++ version 7.0, you must set the environment variables INCLUDE, LIB, and PATH to point to the directories that contain the RPC header files, libraries, and DLLs and binaries, respectively. You cannot install the RPC toolkit in the same directory as the Visual C++ compiler binaries because of name conflicts.

RPC Documentation

The following Microsoft RPC version 1.0 reference materials are available in Windows Write format and in PostScript file format:

Microsoft RPC Programmer's Guide and Reference:

- Part I: Programmer's Guide
- Part II: MIDL Language Reference
- Part III: Run-Time API Reference
- Part IV: Installing RPC
- Part V: Appendixes

Use the PostScript files to print individual chapters of the documentation on your PostScript printer.

The following run-time and MIDL reference Help file is available on line:

   RPC.HLP   WinHelp MIDL and run-time API reference

RPC sample-program source files are available in the directory MSTOOLS\SAMPLES\RPC. MS-DOS and Microsoft Windows 3.x versions of some samples are available when you install the Windows 3.x/MS-DOS version of the RPC toolkit. The file MSTOOLS\SAMPLES\RPC\README.TXT describes the available samples.

Release Notes for MIDL compiler

1.0 The following release notes relate to the MIDL compiler and to building distributed applications.

1.1 Packing and Alignment Considerations

You must use the same packing and alignment settings (/Zp switch) for both the C compiler and the MIDL compiler. Using different packing levels for the two compilers causes undefined results. Specify the /Zp switch to verify that the correct packing and alignment settings are used on both compilers.

This release of the MIDL compiler does not support the switches /Zp1 and /Zp2 in the MIPS environment, although the compiler does not prevent the use of /Zp1 and /Zp2.

Use /Zp1 or /Zp2 for 16-bit client platforms. Objects of types with natural alignment greater than 2 that are allocated on the stack as local variables in the client application are not necessarily allocated on 4- and 8-byte boundaries by the C compiler. Because the C compiler does not guarantee alignment on the stack, marshalling from and unmarshalling into such objects may cause problems.

Generic stubs (/env generic) must be specified with /Zp1 or /Zp2 in 16-bit client environments. Generic stubs specified with /Zp1 or /Zp2 cannot be used in the MIPS environment. MIDL uses /Zp4 by default for generic stubs.

1.2 C Stub Source Code Causes Compilation Warnings

The stub files generated by the MIDL compiler may generate warnings when they are compiled at compiler warning-level 3 and higher. These warnings can generally be safely ignored.

When your C compiler does not use the same default character sign as the MIDL compiler, use the MIDL compiler switch /char to generate explicit declarations in the header file. For more information, see the Microsoft RPC programming documentation.

1.3 Use Six-Character Filenames on the FAT File System

Because RPC version 1.0 appends _C, _X, and similar extensions to filenames, limit your filenames to six characters or less. Filenames that total more than eight characters are too long for some file systems and can fail.

1.4 Specifying Local and UUID Attributes

If the base IDL file contains no procedures, you don't have to specify local or UUID attributes.

1.5 MIDL Extra Server Files in the Windows 3.x/MS-DOS Environment

MIDL does not produce server files in the Windows 3.x/MS-DOS environment. For this reason, if you specify the /env switch as /env dos or /env win16, server stubs are not produced. To produce server stubs, specify that the /env switch is either /env win32 or /env generic.

1.6 Working with Visual C++ on 16-Bit Machines

Do not install the 16-bit RPC toolkit in the same directory as Visual C++. MIDL requires the Microsoft C 7.0 front end for C preprocessing. The installer will install the Microsoft C 7.0 front end if needed. Use the /cpp_cmd switch to make sure MIDL is using the right C compiler.

1.7 Memory Leak Possible with Multiple Context Handles

Memory can leak when data argument(s) precede context-handle argument(s) and the call is directed by another handle. The leak happens on the server side if the data requires memory allocation and if the context handle that is used (as opposed to initialized) is invalid. The stub raises an exception as it is supposed to, but it doesn't do the clean up.

1.8 Use Zero or Positive Values for the size_is and length_is Variables

You must use a zero or a positive value for the size_is and length_is variables. A negative value for the size_is or length_is variable causes an exception.

1.9 RPC Cannot Pass More than 63K Worth of Data on 16-Bit Platforms

An MS-DOS or Windows 3.x system cannot pass more than 63K worth of data in a single remote procedure call. Trying to pass more than 63K worth of data results in undefined behavior.

1.10 Windows 3.x Applications Using the [callback] Attribute

If you use the [callback] attribute for a procedure specified in the IDL file and if your application runs with Windows 3.x, you must compile all stubs with the /GA C-compiler switch. Note that the /GA switch should not be used for Windows callback functions (as opposed to RPC callback functions) that are called in the context of another process.

1.11 Building RPC Samples with Visual C++ for Microsoft Windows NT

You can build RPC applications with the Visual C++ SDK for Microsoft Windows NT using the RPC*.H files distributed with that SDK. To build RPC samples with Visual C++ for Windows NT, add the following definition to RPC.H (this applies to Intel processors only):

```
#define     _CRTAPI1    _cdecl
```

Release Notes for RPC Run-Time/Transport Libraries & Windows NT Services

2.0 The following release notes are related to the RPC run-time libraries, transport libraries, and Windows NT services provided with Microsoft RPC version 1.0.

2.1 RpcServerUseAllProtseqs Requires a Null Security Descriptor

The RpcServerUseAllProtseqs security-descriptor parameter must be set to NULL in this release of Microsoft RPC version 1.0. The null parameter allows everyone access.

2.2 Named-Pipes Security Descriptor

Named pipes (ncacn_np) allows everyone access when a null security descriptor is supplied. This accessibility is independent of whether or not the account used to start the server has a default ACL.

2.3 Multiple Networks

The Microsoft Locator does not work with a router.

2.4 RpcNsBindingExport IP Addresses

If a server has two IP addresses and as a result is on two subnets, RpcNsBindingExport adds only one of the two addresses to the name service. For this reason, clients on one of the two networks cannot import a valid handle to that server. Clients that already know the server address will work using either well-known or dynamic endpoints.

2.5 SPX Transport Limitations

The MS-DOS SPX transport does not function in a Windows DOS box or Windows NT DOS box. The Windows SPX transport does not function in Windows standard mode or in emulation mode with Windows NT.

2.6 All Machines Must Use the Same SPX Packet Size

To use the ncacn_spx protocol sequence (RPC over SPX), both the client and the server must use the same maximum IPX packet size. Otherwise, multipacket RPC calls will fail with RPC_S_CALL_FAILED. To adjust the packet size on a machine running MS-DOS, Windows 3.x, or Windows for Workgroups, add the following line to your NET.CFG or SHELL.CFG file:

```
IPX PACKET SIZE LIMIT=xxxx
```

Here xxxx is the packet size in bytes.

Consult your Novell documentation for more information. Note that some older drivers do not support setting IPX PACKET SIZE LIMIT.

To adjust the maximum packet size on machines running Windows NT, use REGEDT32.EXE to set HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\NWLINKIPX\NetConfig\Driver01\MaxPktSize to the proper value. Consult the Windows NT Resource Kit for more information on the registry.

2.7 Windows 3.x Applications Using TCP/IP Must Call RpcWinSetYieldInfo

Applications based on Windows 3.x that use TCP/IP (ncacn_ip_tcp) must call the RpcWinSetYieldInfo routine. Applications making RPC calls that don't call RpcWinSetYieldInfo will always hit an exception. The exception occurs because Windows Sockets API standard requires that applications yield while using the network.

2.8 When Writing MS-DOS Applications, Avoid Calling _exit Directly

Always write your RPC applications for MS-DOS to call the complete C-library termination function exit or _cexit rather than the quick C-library termination function _exit or _c_exit because the quick-termination functions do not call the atexit function. The MS-DOS RPC run-time libraries use the atexit function to clean up system resources. When you call the _exit or _c_exit function, the atexit function is not invoked and resources are not freed correctly.

Release Notes for Installation & Configuration Issues for This Release

3.0 The following release notes are related to installation and configuration issues for this release:

3.1 Using Microsoft RPC with Microsoft Windows for Workgroups

To successfully run Microsoft RPC distributed applications with Microsoft

Windows for Workgroups version 3.1, you must use the Windows for Workgroups network services. Stop all real-mode network services before starting Windows for Workgroups. At the MS-DOS prompt, enter:

```
net stop workstation /y
win
```

3.2 Creating Installation Disks for Your Distributed Application

After you have developed your distributed application using Microsoft RPC, you should provide a way for your users to install your application.

To enable your users to install your application, perform the following steps when installing RPC:

1. Copy your executable files.

2. Copy Microsoft RPC run-time and transport DLLs.

3. Set Microsoft RPC-related registry entries as needed.

To provide an installation tool for your users, use the Microsoft Setup Toolkit for Windows. Microsoft Setup provides important version-control features that prevent users from overwriting newer versions of the RPC run-time libraries with older, incompatible versions.

You can also use the template batch files provided with Microsoft RPC for MS-DOS and Windows 3.x to help your users install your distributed applications. The files DRUNDISK.BAT and WRUNDISK.BAT copy the Microsoft RPC Setup program and associated files and direct the Microsoft RPC Setup program to install all needed RPC system files. You must customize the .INF file for your application. For more information about changing the .INF file, see the documentation for the Microsoft Setup Toolkit for Windows.

If you use another installation method, you should implement some form of version control. Version-control methods ensure that you do not distribute incompatible versions of the RPC run-time and transport libraries that can cause software errors in your application and other applications.

Some files include an embedded version-control number for use by the Setup Toolkit for Windows. These files are noted in the lists below.

The following Microsoft Windows 3.x RPC files should be installed in the system directory or in a directory specified by the LIBPATH environment variable:

DNETAPI.DLL	Non-Microsoft environments for DEC PATHWORKS interoperability with Microsoft LAN Manager
NETAPI.DLL	Microsoft LAN Manager transport DLL; has version number for use with Microsoft Setup
RPCNS1.DLL	Microsoft RPC name-service provider
RPCRT1.DLL	Microsoft RPC client run-time library
RPC16C1.DLL	RPC transport DLL for client-side named pipes
RPC16C3.DLL	RPC transport DLL for client-side WinSock TCP/IP

RPC16C3X.DLL RPC transport DLL for client-side WSOCKETS.DLL TCP/IP
RPC16C4.DLL RPC transport DLL for client-side DECnet
RPC16C5.DLL RPC transport DLL for client-side NetBIOS
RPC16C6.DLL RPC transport DLL for client-side SPX

The following MS-DOS RPC files should be installed in a directory that is specified by the PATH environment variable:

RPCNS.RPC Microsoft RPC name-service provider
RPCNSLM.RPC Microsoft RPC name-service provider LAN Manager support
RPCNSMGM.RPC Microsoft RPC name-service provider support module
RPC16C1.RPC RPC transport DLL for client-side named pipes
RPC16C3.RPC RPC transport DLL for client-side TCP/IP
RPC16C4.RPC RPC transport DLL for client-side DECnet
RPC16C5.RPC RPC transport DLL for client-side NetBIOS
RPC16C6.RPC RPC transport DLL for client-side SPX

You need not install the Microsoft Windows NT versions of the Microsoft RPC run-time libraries and transports. Microsoft Windows NT computers support Microsoft RPC version 1.0. If you want to run Microsoft Windows 3.x or MS-DOS RPC applications with Microsoft Windows NT, install the above RPC DLLs on the system.

Setting RPC Registry Entries

Your installation procedure should set any registry entries your application needs. Registry entries are used by the RPC run-time libraries and the RPC name-service provider to obtain information about the transport an application intends to use.

By default, MS-DOS and Windows 3.x registry entries are present in the file RPGREG.DAT in the root directory of the boot drive. You can use a different file by setting the value of the environment variable RPC_REG_DATA_FILE to the path and filename of the alternate file.

The RPC Setup program for MS-DOS and Microsoft Windows 3.x creates the registry file RPCREG.DAT. If you write your own installation program, you must create RPCREG.DAT and set appropriate entries for the name-service and NetBIOS transports supported in that environment.

If the Microsoft Locator is the name-service provider:

```
\Root\Software\Microsoft\Rpc\NameService\Protocol=ncacn_np
\Root\Software\Microsoft\Rpc\NameService\NetworkAddress=\\.
\Root\Software\Microsoft\Rpc\NameService\Endpoint=\pipe\locator
\Root\Software\Microsoft\Rpc\NameService\DefaultSyntax=3
```

If CDS is the name-service provider via NSID:

```
\Root\Software\Microsoft\Rpc\NameService\Protocol=ncacn_ip_tcp
\Root\Software\Microsoft\Rpc\NameService\NetworkAddress=NSID host name
\Root\Software\Microsoft\Rpc\NameService\Endpoint=
\Root\Software\Microsoft\Rpc\NameService\DefaultSyntax=3
```

The NetBIOS transport entries have the following form:

```
\Root\Software\Microsoft\Rpc\NetBios\ncacn_nb_<A><B>=<C>
```

Where:

```
<A> is the NetBIOS sub-protocol sequence (nb, ipx, or tcp).
<B> is a unique digit for each protocol sequence.
<C> is the lana number.
```

For example, if you have two net cards in a system, running NetBEUI on both and TCP/IP on one, and the lana numbers on the system are configured as NetBEUI on card0 is 0, TCP/IP on card0 is 1, and NetBEUI on card1 is 2, then the RPC NetBIOS registry entries are:

```
\Root\Software\Microsoft\Rpc\NetBios\ncacn_nb_nb0=0
\Root\Software\Microsoft\Rpc\NetBios\ncacn_nb_nb1=2
\Root\Software\Microsoft\Rpc\NetBios\ncacn_nb_tcp0=1
```

For more information about the strings generated in the file RPCREG.DAT, run Microsoft RPC Setup and inspect the strings.

REFERENCES

RPCREAD.ME in \MSTOOLS\SAMPLES\RPC in Win32SDK for Windows NT version 3.1 on MSDN.

Additional reference words: 6.20 3.10
KBCategory: kbnetwork kbref kbtshoot kbdocerr
KBSubcategory: NtwkRpc

DOCERR: Important Information on RPC 2.0 Missing from the

PSS ID Number: Q129143
Authored 18-Apr-1995 Last modified 19-Apr-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows for Workgroups version 3.11
- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5

SUMMARY

This article contains important information not included in the Microsoft RPC programming documentation or in Help about Microsoft RPC version 2.0. This information is identical to the information provided in the RPCREAD.ME file in the Win32 SDK (MSTOOLS\SAMPLES\RPC\) for Windows NT version 3.5.

Microsoft RPC is a toolkit for developing distributed applications in C/C++. The toolkit includes:

- MIDL compiler for Microsoft Windows NT.

- C/C++ language header files and run-time libraries for Windows NT, Microsoft Windows versions 3.x, and MS-DOS.

- Sample programs for Windows NT, Windows versions 3.x, and MS-DOS.

MORE INFORMATION

This article is organized into the following sections:

Installation

The Windows NT SDK provides the components of the RPC toolkit as part of its standard installation. No additional installation is required. To produce MS-DOS and Windows 3.x RPC clients, install version 1.50 of the Microsoft Visual C/C++ development environment, and then install the MS-DOS/Win16 RPC tookit from disk. To install the MS-DOS/Windows 3.x version of the RPC toolkit, use the Setup program in the MSTOOLS\RPC_DOS\DISK1 directory.

Because the 16-bit MIDL compiler is no longer supported, you must develop applications using the 32-bit MIDL compiler. However, to write MS-DOS or

Windows 3.x applications without a 32-bit C/C++ compiler capable of targeting MS-DOS, you must compile the IDL file with a 32-bit MIDL compiler on Windows NT. Then, compile the application and stubs using your C/C++ compiler. To write MS-DOS or Windows 3.x applications using a 32-bit C/C++ compiler that is capable of targeting MS-DOS, compile both the IDL file and C/C++ files on Windows NT.

Documentation

The RPC Programmer's Guide and Reference is available in Help and includes conceptual material, MIDL language and command-line references, and run-time API references.

Because many new features have been included in this version of Microsoft RPC, see the New in this Version of RPC help topic.

RPC sample program source files are available in the directory MSTOOLS\SAMPLES\RPC. The MSTOOLS\SAMPLES\RPC\README.TXT file describes each sample.

Before Using RPC

Read the following section on MIDL and run-time API issues before attempting to use this version of RPC. These sections contain important information that is not documented in Help.

MIDL Issues

The 16-bit MIDL compiler for MS-DOS or Windows 3.x is no longer supported. Use the 32-bit MIDL compiler switch /env with either the DOS or WIN16 option.

Note the following when using this version of the MIDL compiler:

- When compiling the generated stubs, warnings about different levels of pointer indirection or different const specifications are benign.

- When using the DOS or WIN option of the -env switch, the compiler will not set the correct packing level to 2. In this case, you must explicitly specify the correct packing level by using the /Zp or /pack switch.

- For Alpha platforms, procedure serialization stubs must be compiled on the C compiler by using the -Od switch. The procedure encoding/decoding stubs on the Alpha platform should not be compiled as optimized (for example, using the -Ox switch). This affects the stubs on the Alpha platform only if the serializing procedure uses one of the following attributes on top-level parameters:

    - array or ptr attributes, such as size_is or length_is.
    - The switch_is union attribute.

- A structure whose only member is a conformant string results in an MIDL compiler assert.

- In the osf mode of the compiler, the default allocate and free routines map to NdrRpcSmClientAllocate and NdrRpcSmClientFree. The signatures for these routines are in RPCNDR.H

Run-Time API Issues

When using the ncacn_spx and ncadg_ipx transports, the server name is exactly the same as the Windows NT server name. However, because the names are distributed using Novell protocols, they must conform to the Novell naming conventions. If a server name is not a valid Novell name, servers will not be able to create endpoints with the ncacn_spx or ncadg_ipx transports. The following is a partial list of characters prohibited in Novell server names:

" * + . / : ; < = > ? [ ] \ |

When using ncadg_ipx on Windows 3.x or MS-DOS platforms, use the Windows NT 3.1 model of server naming. That is, a tilde, followed by the servers eight-digit network number, followed by its twelve-digit Ethernet address.

The ncacn_spx transport is not supported by the version of NWLink supplied with MS Client 3.0; however, ncadg_ipx is supported.

Backslashes are now optional in the host name for ncacn_np, for consistency with the other transports.

The datagram protocols (ncadg_ipx, ncadg_ip_udp) have the following limitations:

- They do not support callbacks. Any functions using the callback attribute will fail.

- They do not support the RPC security API (RpcBindingSetAuthInfo, RpcImpersonateClient, and so on).

Only the ncacn_spx and ncacn_ip_tcp protocols support cancels. For all other protocols, the cancel routines will return RPC_S_OK, but there will be no effect. Specifically:

- RpcCancelThread will alert the specified thread, but will not interrupt a pending RPC.

- RpcTestCancel will return RPC_S_OK if the current thread has been alerted.

- RpcMgmtSetCancelTimeout has no visible effect.

RpcTestCancel, RpcMgmtSetCancelTimeout, and RpcCancelThread are only supported on Windows NT platforms; all other platforms return RPC_S_CANNOT_SUPPORT.

The function RpcBindingSetAuthInfo does not accept the value RPC_C_AUTHN_DEFAULT.

To use the RpcBindingSetAuthInfo routine and, in particular, when using MS-DOS authentication in this version of RPC, note this information:

- The constant RPC_C_AUTHN_WINNT has been defined in the header RPCDCE.H. This constant should be passed as the AuthnSvc parameter to use the Windows NT LAN Manager Security Support Provider (NTLMSSP) service.

- When using the RPC_C_AUTHN_WINNT authentication service, the AuthIdentity parameter should be a pointer to a SEC_WINNT_AUTH_IDENTITY structure, defined in RPCDCE.H. This structure contains strings for the user's domain, username, and password. You can pass a NULL pointer to use the information for the currently logged-in user.

For MS-DOS, you cannot pass NULL because there is no way for RPC to determine the current user's logon information. However, either a structure or NULL is acceptable for Windows versions 3.x and Windows NT.

When using authenticated RPC on a system composed of Novell NetWare and Windows version 3.x (not Windows for Workgroups), you must run the optional NETBIOS.EXE program before starting Windows.

The 16-bit SDK cannot be installed on a Windows NT computer unless the current user is an administrator.

Supported Platforms for Runtime APIs

The following function is supported only by 16-bit Windows 3.x platforms:

RpcWinSetYieldInfo

The following functions are supported only by 32-bit Windows NT platforms:

DceErrorInqTest
RpcBindingInqAuthClient
RpcBindingServerFromClient
RpcCancelThread
RpcEpRegister
RpcEpUnregister
RpcIfIdVectorFree
RpcImpersonateClient
RpcNetworkInqProtseqs
RpcObjectInqType
RpcObjectSetInqFn
RpcObjectSetType
RpcProtseqVectorFree
RpcRevertToSelf
RpcServerInqBindings
RpcServerInqIf
RpcServerListen
RpcServerRegisterAuthInfo
RpcServerRegisterIf

RpcServerUnregisterIf
RpcServerUse*Protseq*
RpcMgmtInqIfIds
RpcMgmtInqStats
RpcMgmtIsServerListening
RpcMgmtInqServerPrincName
RpcMgmtSetAuthorizationFn
RpcMgmtSetCancelTimeout
RpcMgmtSetServerStackSize
RpcMgmtStatsVectorFree
RpcMgmtStopServerListening
RpcMgmtWaitServerListen
RpcMgmtEnableIdleCleanup
RpcMgmtEpElt*
RpcMgmtEpUnregister
RpcNsBindingExport
RpcNsBindingUnexport
RpcTestCancel

All other runtime functions are supported on Windows NT, Windows 3.x, and MS-DOS platforms.

REFERENCES

RPCREAD.ME in \MSTOOLS\SAMPLES\RPC in Win32SDK for Windows NT 3.5 on MSDN.

Additional reference words: 6.20 3.50
KBCategory: kbnetwork kbtshoot kbref kbdocerr
KBSubcategory: NtwkRpc

DOCERR: Media Player Command-Line Switches

PSS ID Number: Q126869
Authored 06-Mar-1995 Last modified 15-May-1995

The information in this article applies to:

- Microsoft Windows operating system versions 3.1, 3.11
- Microsoft Windows for Workgroups versions 3.1, 3.11
- Microsoft Windows NT versions 3.5, 3.51
- Microsoft Windows Software Development Kit (SDK), version 3.1
- Microsoft Win32 SDK, versions 3.5 and 3.51

Media Player (both the 16-bit MPLAYER.EXE and the 32-bit MPLAY32.EXE) supports the following command-line syntax:

```
MPLAYER /play /close /embedding file
```

All of the command-line switches and the file name are optional. Their meanings are as follows:

/play	Play file right away.
/close	Close after playing (only valid with /play).
/embedding	Run as an OLE server.
file	The file or device to open.

This information was inadvertently omitted from the Windows documentation.

In Windows 95 and Windows NT 3.5 and higher, the Media Player also supports the following switch:

/open	Open the file if specified; otherwise, show the File Open dialog.

In addition, the following switches can be used to ensure that the specified file is of a particular type:

/VFW	The file must be a Video for Windows (.AVI) file.
/MID	The file must be a MIDI (.MID) file.
/WAV	The file must be a Wave Audio (.WAV) file.

Additional reference words: 3.10 3.50 4.00 undocumented win31 wfw wfwg
KBCategory: kbdocerr kbmm
KBSubcategory: MMMisc

DOCERR: TabbedTextOut Tab Positions Really in Logical Units

PSS ID Number: Q113253
Authored 29-Mar-1994 Last modified 22-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1 and 3.5

Both the printed and electronic documentation from the Windows 3.0 and 3.1 SDKs, as well as the Win32 SDK, state that the tab positions parameter of TabbedTextOut() is in device units. This is incorrect. The tab position array is in logical units. The Windows 3.0 and 3.1 SDKs refer to the tab position array as lpnTabPositions, while the Win32 SDK calls it lpnTabStopPositions.

The use of logical units can be seen by using a mapping mode other than the default MM_TEXT. The output of tabs using TabbedTextOut() reflects the mapping mode.

Additional reference words: 3.00 3.10 3.50 docerr
KBCategory: kbprg kbdocerr
KBSubcategory: GdiDrw

Drawing a Rubber Rectangle

PSS ID Number: Q114471
Authored 04-May-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

Most drawing software uses what is termed a "rubber rectangle". This term is used to describe the situation where

1. the left mouse button is held down, defining one corner of the rectangle

2. the mouse is dragged and released at the point defining the opposite corner of the rectangle

3. the rectangle is drawn while the mouse is being dragged, so that it looks like the rectangle is being stretched and contracted, like a rubber band

MORE INFORMATION

The key to making this work is in the following call, which should be made in the WM_LBUTTONDOWN case:

```
SetROP2( hDC, R2_NOT )
```

On each WM_MOUSEMOVE message, the rectangle is redrawn in its previous position. Because of the ROP code, the rectangle appears to be erased. The new position for the rectangle is calculated and then the rectangle is drawn.

Note that Windows will only let you draw in the invalid area of the window if you use a DC returned from BeginPaint(). If you want to use the DC returned from BeginPaint(), you must first call InvalidateRect() to specify the region to be updated.

With the DC returned from GetWindowDC(), Windows will restrict your drawing to the client and non-client areas. With the hDC returned from CreateDC(), you can write on the entire display, so you must be careful.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiDrw

Drawing Outside a Window's Client Area

PSS ID Number: Q33096
Authored 18-Jul-1988 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

When an application uses the BeginPaint or GetDC function to obtain a device context (DC) for its client window and draws into this DC, Windows clips the output to the edge of the client window. While this is usually the desired effect, there are circumstances where an application draws outside the client area of its window.

MORE INFORMATION

The GetWindowDC function provides a DC that allows an application to draw anywhere within its window, including the nonclient area.

In the Windows environment, the display is a scarce resource that is shared by all applications running in the system. Most of the time, an application should restrict its output to the area of the screen it has been assigned by the user. However, an application can use the CreateDC function to obtain a DC for the entire display, as follows:

```
hDC = CreateDC("DISPLAY", NULL, NULL, NULL);
```

Device contexts are another scarce resource in the Windows environment. When an application creates a DC in response to a WM_PAINT message, it must call the DeleteDC function to free the DC before it completes processing of the message.

Painting in the nonclient area of a window is not recommended. If an application changes the nonclient area of its window, the user can become confused because the familiar Windows controls change appearance or are not available. An application should not corrupt other windows on the display.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiDc

DSKLAYT2 Does Not Preserve Tree Structure of Source Files

PSS ID Number: Q87947
Authored 12-Aug-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.5
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

The DSKLAYT2 utility in the Microsoft Setup Toolkit for Windows creates disk images using the files specified in the .LYT file produced by the DSKLAYT utility. The created disk images are flat; all files are placed in the root directory. DSKLAYT2 does not preserve the tree structure of the source files.

Having flat directories on the setup disks is a limitation of only the DSKLAYT2 utility. _MSTEST.EXE, the setup driver spawned by SETUP.EXE, supports a tree structure on the setup disks. Once DSKLAYT2 creates disk images, manually create subdirectories on each disk. Then move files from the root directory of the disk to the appropriate position in the tree structure. (Do not move files between disks.) Make corresponding changes to the .INF file. In the file description line of each moved file, change the filename to indicate the file's new location in the tree structure.

Additional reference words: 3.10 3.50 4.00 95 MSSetup tool kit
KBCategory: kbtool
KBSubcategory: TlsMss

Dsklayt2 Does Not Support Duplicate Filenames

PSS ID Number: Q87906
Authored 11-Aug-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The Dsklayt2 utility in the Microsoft Setup Toolkit for Windows does not support duplicate filenames in the source tree. If two or more files have the same name, Dsklayt2 may place more than one of these files into the root directory of the same setup disk. If this happens, Dsklayt2 generates a warning message when it creates a disk image for the disk that warns that the first file is overwritten.

If two or more files in the source tree for an application share the same name, rename all but one of the files before running the Dsklayt utility. Use the Rename Copied File option in the Dsklayt utility to have the filenames changed back to their original names when SETUP.EXE or _MSTEST.EXE copies the files from the setup disk to the destination disk.

When Dsklayt2 creates a compressed file, it adds an underscore (_) to the end of the file extension, replacing the third character if necessary. Therefore, Dsklayt2 does not support compressed files if the names of the uncompressed files differ only in the third character of the file extension.

Additional reference words: 3.10 3.50 4.00 95 MSSetup tool kit
KBCategory: kbtool
KBSubcategory: TlsMss

Dynamic Loading of Win32 DLLs

PSS ID Number: Q90745
Authored 21-Oct-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

When using LoadLibrary() under Win16 or OS/2, the Dynamic Link Library (DLL) is loaded only once. Therefore, the DLL has the same address in all processes. Dynamic loading of DLLs is different under Windows NT.

A DLL is loaded separately for each process because each application has its own address space, unlike Win16 and OS/2. Pages must be mapped into the address space of a process. Therefore, it is possible that the DLL is loaded at different addresses in different processes. The memory manager optimizes the loading of DLLs so that if two processes share the same pages from the same image, they will share the same physical memory.

Each DLL has a preferred base address, specified at link time. If the address space from the base address to the base address plus image size is unavailable, then the DLL is loaded elsewhere and fixups will be applied. There is no way to specify a load address at load time.

To summarize, at load time the system:

1. Examines the image and determines its preferred base address and required size.

2. Finds the address space required and maps the image, copy-on-write, from the file.

3. Applies internal fixups if the image isn't at its preferred base.

4. Fixes up all dynamic link imports by placing the correct address for each imported function in the appropriate entry of the Import Address Table. This table is contiguous with 32-bit addresses, so 1024 imports require dirtying only one page.

MORE INFORMATION

The pages containing code are shared, using a copy-on-write scheme. The term copy-on-write means that the pages are read-only; however, when a process writes the page, instead of an access violation, the memory manager makes a private copy of the page and allows the write to proceed. For example, if two processes start from the same .EXE, both initially have all pages mapped from the .EXE copy-on-write. As the two processes proceed to modify pages, they get their own copies of the modified pages. The memory

manager is free to optimize unmodified pages and actually map the same physical memory into the address space of both processes. Modified pages are swapped to/from the page file instead of the .EXE file.

There are two kinds of fixups. One is the address of an imported function. All these fixups are localized in what the Portable Executable (PE) specification calls the Import Address Table (IAT). This is an array of 32-bit function pointers, one for each imported API. The IAT is located on its own page(s), because it is always modified. Calling an imported function is actually an indirect call through the appropriate entry in this array. In case that the image is loaded at the preferred address, the only fixups needed are for imports.

Note that there is an optimization whereby each import library exports a 32-bit number for each API along with any name and ordinal. This serves as a "hint" to speed the fixups performed at load time. If the hints in the program and the DLL do not match, the loader uses a binary search by name.

The other kind of fixup is needed for references to the image's own code or data when the image can't be loaded at its preferred address. When a page must be taken out of memory, the system checks to see whether the page has been modified. If it has not, then the page is still mapped copy-on-write against the EXE and can be discarded from memory. Otherwise, it must be written to the page file before it can be removed from memory, so that the page file is used as the backing store (where the page is recovered from) rather than the executable image file.

NOTES

The DLL's entry point does not get called for a second LoadLibrary() call in a process (that is, no second DLL_PROCESS_ATTACH entry). There is one call to DllEntry/DLL_THREAD_ATTACH per thread no matter the number of times a thread calls LoadLibrary(). The same goes for FreeLibrary(), but the DLL_PROCESS_DETACH happens only on the last call (that is, reference count back to zero for the process).

Global instance data for the DLL is on a per process basis (only one set per unique process). If it is necessary to ensure that global instance data is unique for each LoadLibrary() performed in a single process, consider thread local storage (TLS) as an alternative. This requires multiple threads of execution, but TLS allows unique data for each ThreadID. There is very little overhead on the DLL's part; just create a global TLS index during process initialization. During thread initialization, allocate memory (via HeapAlloc(), GlobalAlloc(), LocalAlloc(), malloc(), and so on) and store a pointer to the memory using the global TLS index value in the function TlsSetValue. Win32 internally stores each thread's pointer by TLS index and ThreadID to achieve the thread specific storage.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseDll

Efficiency of Using SendMessage Versus SendDlgItemMessage

PSS ID Number: Q66944
Authored 14-Nov-1990 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

The SendDlgItemMessage function is equivalent to obtaining the handle of a dialog control using the GetDlgItem function and then calling the SendMessage function with that handle. The SendDlgItemMessage function therefore takes slightly longer to execute than the SendMessage function for the same message, because an extra call to the GetDlgItem function is required each time the SendDlgItemMessage function is called.

The GetDlgItem function searches through all controls in a given dialog box to find one that matches the given ID value. If there are many controls in a dialog box, the GetDlgItem function can be quite slow.

If an application needs to send more than one message to a dialog control at one time, it is more efficient to call the GetDlgItem function once, using the returned handle in subsequent SendMessage calls. This saves Windows from searching through all the controls each time a message is sent. The SendMessage function should also be used when your application retains handles to controls that receive messages.

However, if your application needs to send one message to many controls, such as sending WM_SETFONT messages to all the controls in a dialog, then the SendDlgItemMessage function will save code in the application because a call to the GetDlgItem function is not made for each control.

Note that if the message sent to a control may result in a lengthy operation (such as sending the LB_DIR message to a list box), then the overhead in the GetDlgItem call is negligible. Either the SendDlgItemMessage or SendMessage can be used, whichever is more convenient.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95 dlgitem
KBCategory: kbprg
KBSubcategory: UsrDlgs

EM_SETHANDLE and EM_GETHANDLE Messages Not Supported

PSS ID Number: Q130759
Authored 25-May-1995 Last modified 22-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SUMMARY

The EM_GETHANDLE and EM_SETHANDLE messages are not supported for edit contols that are created as controls of a 32-bit application under Windows 95. This is due to the way USER is designed under Windows 95. 16-bit applications work the same way they did under Windows version 3.1. That is, they can use the EM_GET/SETHANDLE messages. Also Win32-based applications running under Windows NT will be able to use these messages.

MORE INFORMATION

The EM_GETHANDLE and EM_SETHANDLE messages are used to retrieve and set the handle of the memory currently allocated for a multiline edit control's text. USER under Windows 95 is a mixture of 16- and 32-bit code, so edit controls created inside a 32-bit application cannot use these messages to retrieve or set the handles. Trying to do so causes the application to cause a general protection (GP) fault and thereby be terminated by the System.

One workaround that involves a little code modification is to use the GetWindowTextLengt(), GetWindowText(), and SetWindowText() APIs to retrieve and set the text in a edit control.

NOTE: USER is almost completely 16-bit, so 32-bit applications thunk down to the 16-bit USER. Also note that the EM_GETHANDLE and EM_SETHANDLE messages cannot be used with Win32s-based applications either.

Additional reference words: 4.00 user controls GPF
KBCategory: kbprg
KBSubcategory: UsrCtl

Enumerating Network Connections

PSS ID Number: Q119216
Authored 10-Aug-1994 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

From the MS-DOS prompt, you can enumerate the network connections (drives) by using the following command:

```
net use
```

Programmatically, you would call WNetOpenEnum() to start the enumeration of connected resources and WNetEnumResources() to continue the enumeration.

MORE INFORMATION

The following sample code enumerates the network connections:

Sample Code

```
#include <windows.h>
#include <stdio.h>

void main()
{
   DWORD dwResult;
   HANDLE hEnum;
   DWORD cbBuffer = 16384;
   DWORD cEntries = 0xFFFFFFFF;
   LPNETRESOURCE lpnrDrv;
   DWORD i;

   dwResult = WNetOpenEnum( RESOURCE_CONNECTED,
                            RESOURCETYPE_ANY,
                            0,
                            NULL,
                            &hEnum );

   if (dwResult != NO_ERROR)
   {
      printf( "\nCannot enumerate network drives.\n" );
      return;
   }

   printf( "\nNetwork drives:\n\n" );
```

```
      do
      {
         lpnrDrv = (LPNETRESOURCE) GlobalAlloc( GPTR, cbBuffer );

         dwResult = WNetEnumResource( hEnum, &cEntries, lpnrDrv, &cbBuffer
   );

         if (dwResult == NO_ERROR)
         {
            for( i = 0; i < cEntries; i++ )
            {
               if( lpnrDrv[i].lpLocalName != NULL )
               {
                  printf( "%s\t%s\n", lpnrDrv[i].lpLocalName,
                                      lpnrDrv[i].lpRemoteName );
               }
            }
         }
         else if( dwResult != ERROR_NO_MORE_ITEMS )
         {
            printf( "Cannot complete network drive enumeration" );
            GlobalFree( (HGLOBAL) lpnrDrv );
            break;
         }
         GlobalFree( (HGLOBAL) lpnrDrv );
      }
      while( dwResult != ERROR_NO_MORE_ITEMS );

      WNetCloseEnum(hEnum);
   }
```

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: NtwkWinnet

ERROR_BUS_RESET May Be Benign

PSS ID Number: Q111837
Authored 20-Feb-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

Tape API (application programming interface) functions may return an error code of ERROR_BUS_RESET when operating on SCSI tape devices. In many cases, you can ignore this error value and retry the operation. However, this error is fatal if received during a series of write operations because a tape drive cannot recover from a bus reset and continue writing.

MORE INFORMATION

When Windows NT boots up it resets the SCSI bus. This bus reset is reported by the tape drive in response to the first operation after the reset.

The code fragment shown below in the Sample Code section could be used to check for ERROR_BUS_RESET and clear it. The same technique could be used for other informational errors, such as ERROR_MEDIA_CHANGED, that may not be relevant at application startup.

Sample Code

```
/*
** This is a code fragment only and will not compile and run as is.
*/

   ...
   do {
      dwError = GetTapeStatus(hTape);
   } while (dwError == ERROR_BUS_RESET);
   ...
```

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseFileio

Establishing Advise Loop on Same topic!item!format! Name

PSS ID Number: Q95983
Authored 03-Mar-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

Sometimes more than one DDEML client application might establish an advise loop with a server on the same topic!item!format name set. This article discusses the complexities involved in such a transaction.

MORE INFORMATION

A client application sends an XTYP_ADVSTART to DDEML when it needs periodic updates on a particular data item from a server, typically when that particular data item's value changes.

The server application calls DdePostAdvise whenever the value of the requested data item changes. This results in an XTYP_ADVREQ transaction being sent to the server's DDEML callback function, where the server returns a handle to the changed data.

The client then receives the updated data item during the XTYP_ADVDATA transaction in the case of a hot advise loop. In a warm advise loop, the XTYP_ADVDATA transaction that the client receives in its callback does not contain data; it has to specifically request data with an XTYP_REQUEST transaction.

When more than one client application requests an advise loop on the same topic!item!format name set, DDEML maintains a list of these client applications so that it knows that on one call to DdePostAdvise(), it should send the changed hData to all the requesting applications in its list.

The server's callback receives an XTYP_ADVREQ transaction as a result of DdePostAdvise() with the LOWORD (dwData1) containing a count of the number of ADVREQ transactions remaining to be processed on the same topic!item!format name set.

This count allows the server application to create its hData as HDATA_APPOWNED, whereby it could create a data handle just once and pass the same handle on to its other pending requests on the same topic!item!format name set. Finally, when the count is down to zero, DdeFreeDataHandle() can then be called on this hData.

Note that a server needs to call DdePostAdvise() only once regardless of how many pending advise requests it has on the same

topic!item!format name set. This one call to DdePostAdvise() causes DDEML to send the appropriate number of XTYP_ADVREQ transactions to the server's callback.

All these can be easily illustrated using the Windows version 3.1 Software Development Kit (SDK) DDEML CLIENT and SERVER samples in this manner:

1. Start the SERVER application.

2. a. Start the CLIENT application.
   b. Establish a connection with the SERVER.
   c. Start an advise loop on the item "Rand".

3. a. Start another instance of the client application.
   b. Establish a connection with the SERVER.
   c. Start an advise loop on the item Rand.

4. Bring up DDESPY.

5. Go back to the SERVER and choose ChangeData from the Options menu and watch both CLIENT applications update their data.

Results (from DDESPY main window):

1. Two XTYP_ADVREQs (because you have two pending ADVREQs on the same test!Rand pair.

2. Changed Rand data is then sent to the first CLIENT in the advise list.

3. The first CLIENT in the advise list receives the data via XTYP_ADVDATA.

4. Changed "Rand" data is sent to the second CLIENT in the advise list.

5. The second CLIENT in the advise list receives its XTYP_ADVDATA.

One caveat to this scenario is when an advise loop is invoked with the XTYPF_ACKREQ flag set (that is, the client establishes an XTYP_ADVSTART transaction or'ed with the XTYPF_ACKREQ flag). In this case, the server does not send the next data item until an ACK is received from the client for the first data item. During a particular call to DdePostAdvise(), the server might not necessarily receive XTYP_ADVREQ in its callback for all active links, and the LOWORD(dwData1) might not necessarily reach 0 (zero). When the DDE_FACK from the client finally arrives, DDEML then sends the server an XTYP_ADVREQ with LOWORD(dwData1) set to CADV_LATEACK, identifying the late-arriving ACK appropriately.

Advise links of this kind (with XTYPF_ACKREQ flag set) are best suited to situations where the server sends information faster than a client can process it--setting the XTYPF_ACKREQ bit ensures that the server never outruns the client. However, setting this flag also sets a drawback in circumstances where data transitions may be lost. Thus, in Windows NT or in similar situations where server outrun is highly unlikely, it is recommended that the XTYPF_ACKREQ bit not be used to prevent such data transition loss.

Note that in this delayed ACK update scenario, the count received in the LOWORD (dwData1) may not be relied upon for creating APPOWNED data handles as discussed in the earlier paragraphs; where an hData is created once, and when the count is down to zero, DdeFreeDataHandle() is called on this hData.

This does not, however, imply that the efficiency provided by APPOWNED data handles may not be used at all. In this case, a server could create an APPOWNED data handle once--usually on the first XTYP_ADVREQ it receives--and associate that handle with a topic!item!format name set. It could then return this data handle for all subsequent requests it receives on this topic!item!format set. Each time data changes thereafter, the server should destroy the old data handle and not re-render the data [that is, call DdeCreateDataHandle()] until another request comes through.

This might be better explained as follows:

```
      case XTYP_ADVREQ:
           if (ThisIsForTheTopicItemFormatSpecified)
           {
               if (bFirstTimeRequested)
                  {
                      bFirstTimeRequested = FALSE;
                      hData = DdeCreateDataHandle();
                   }
                return hData;
            }
            break;

     // and then whenever data changes for this topic!item!format
         if (hData)
           {
             DdeFreeDataHandle (hData);
             bFirstTimeRequested = TRUE;
            }
         DdePostAdvise();          // specify topic!item!format here.
                                   // This causes DDEML to send an
XTYP_ADVREQ
                                   // which is handled above.
```

For Microsoft Windows version 3.1 DDEML, the only way for a server application to distinguish which client's advise request is currently being responded to is through the XTYP_ADVREQ's hConv parameter. The hConvPartner field of the CONVINFO structure may be used to distinguish between clients.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDde

Examining the dwOemId Value

PSS ID Number: Q101190
Authored 07-Jul-1993 Last modified 22-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

The Win32 application programming interface (API) GetSystemInfo() fills in the members of a SYSTEM_INFO structure. The dwOemId member represents a computer identifier that is specific to a particular OEM (original equipment manufacturer). Windows NT versions 3.1 - 3.51 and Windows 95 always place a zero in the dwOemId member. In later releases, this behavior will change to include different OEM IDs.

Additional reference words: 3.10 3.50 4.00
KBCategory: kbprg
KBSubcategory: BseMisc

Explanation of the NEWCPLINFO Structure

PSS ID Number: Q103315
Authored 22-Aug-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

The following is an explanation of the NEWCPLINFO structure:

dwSize: Specifies the length of the structure, in bytes. Set this to sizeof(NEWCPLINFO).

dwFlags: Specifies Control Panel flags. This is field is currently unused. Set this field to NULL.

dwHelpContext: Specifies the context number for the topic in the help project (.HPJ) file that displays when the user selects help for the application. If dwData is non-NULL, Windows Help will be invoked with the HELP_CONTEXT fuCommand with dwHelpContext as dwData. If dwHelpContext is NULL, Windows Help is invoked with the HELP_INDEX fuCommand.

lData: Specifies data defined by the application. This is passed back to the application in the CPL_DBLCLK, CPL_SELECT, and CPL_STOP messages via lParam2.

hIcon: Identifies an icon resource for the application icon. This icon is displayed in the Control Panel window.

szName: Specifies a null-terminated string that contains the application name. The name is the short string displayed below the application icon in the Control Panel window. The name is also displayed in the Settings menu of Control Panel.

szInfo: Specifies a null-terminated string containing the application description. The description is displayed at the bottom of the Control Panel window when the application icon is selected.

szHelpFile: Specifies a null-terminated string that contains the path of the help file, if any, for the application. If this field is unused, set it to NULL. The Control Panel will invoke a default Windows Help file when this field is NULL.

Additional reference words: 3.10 3.50 3.51 4.00 95 cpl control panel extension
KBCategory: kbprg
KBSubcategory: UsrExt

Exporting Callback Functions

PSS ID Number: Q83706
Authored 19-Apr-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

It is not necessary for Win32-based applications to export callback functions. Windows versions 3.1 and earlier (Win16) need callback functions primarily for fixing references to global data and ensuring that EMS memory is not paged out. Neither of these situations applies to the Windows NT operating system.

MORE INFORMATION

Exports are necessary for any function that must be located at either

- Run time via GetProcAddress()(dynamic linking)

  -or-

- Load time via an import library (static linking)

Both of these linking methods require that the name or ordinal number of the export be known and that their names (or ordinal numbers) be present in the executable's exported entry table. This enables Windows to determine the addresses at run time.

Static linking is done by the loader, which performs this lookup for all of the imported entry points that an executable needs (normally by ordinal number). In dynamic linking, the system scans by ordinal number or by name through the DLL (Dynamic Link Library) exports table.

In Win16, exported entries are automatically fixed by the linker to adjust to the appropriate data segment. Exporting entries on Win32 just adds them to the module's exported names and ordinal numbers table; the linker does not need to "fix" them. For code compatibility with Win16, you may want to continue to use MakeProcInstance() and export all callbacks. This macro does nothing on Windows NT.

In short,

	On Windows	Win32
Callbacks	Export or use MakeProcInstance	Use address of fn
GetProcAddress	Must export	Must export

Static linking	Must export		Must export

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseDll

Exporting Data from a DLL or an Application

PSS ID Number: Q90530
Authored 18-Oct-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

It is possible for a Win32-based application to be able to address DLL global variables directly by name from within the executable. This is done by exporting global data names in a way that is similar to the way you export a DLL function name. Use the following steps to declare and utilize exported global data.

1. Define the global variables in the DLL code. For example:

```
int i = 1;
int *j = 2;
char *sz = "WBGLMCMTP";
```

2. Export the variables in the module-definition (DEF) file. With the 3.1 SDK linker, use of the CONSTANT keyword is required, as shown below:

```
EXPORTS
   i  CONSTANT
   j  CONSTANT
   sz CONSTANT
```

   With the 3.5 SDK linker or the Visual C++ linker, use of the DATA keyword is required, as shown below

```
EXPORTS
   i  DATA
   j  DATA
   sz DATA
```

   Otherwise, you will receive the warning

```
   warning LNK4087: CONSTANT keyword is obsolete; use DATA
```

   Alternately, with Visual C++, you can export the variables with:

```
   _declspec( dllexport ) int i;
   _declspec( dllexport ) int *j;
   _declspec( dllexport ) char *sz;
```

3. If you are using the 3.1 SDK, declare the variables in the modules that will use them (note that they must be declared as pointers because a pointer to the variable is exported, not the variable itself):

```
extern int *i;
extern int **j;
extern char **sz;
```

If you are using the 3.5 SDK or Visual C++ and are using DATA, declare the variables with _declspec(dllimport) to avoid having to manually perform the extra level of indirection:

```
_declspec( dllimport ) int i;
_declspec( dllimport ) int *j;
_declspec( dllimport ) char *sz;
```

4. If you did not use _declspec(dllimport) in step 3, use the values by dereferencing the pointers declared:

```
printf( "%d", *i );
printf( "%d", **j );
printf( "%s", *sz );
```

It may simplify things to use #defines instead; then the variables can be used exactly as defined in the DLL:

```
#define i *i
#define j *j
#define sz *sz

extern int i;
extern int *j;
extern char *sz;

printf( "%d", i );
printf( "%d", *j );
printf( "%s", sz );
```

MORE INFORMATION

NOTE: This technique can also be used to export a global variable from an application so that it can be used in a DLL.

REFERENCE

For more information on the use of EXPORTS and CONSTANT in the Module Definition File (DEF) file for the 3.1 SDK, see Chapter 4 of the Win32 SDK "Tools" manual.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseDll

Extending Standard Windows Controls Through Superclassing

PSS ID Number: Q76947
Authored 03-Oct-1991 Last modified 16-May-1995

-
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

-

SUMMARY

A Windows-based application can extend the behavior of a standard Windows control by using the technique of superclassing. An application can superclass a standard Windows control by retrieving its window class information, modifying the fields of the WNDCLASS structure, and registering a new class. For example, to associate status information with each button control in an application, the buttons can be superclassed to provide a number of window extra bytes.

This article describes a technique to access the WNDCLASS structure associated with the standard "button" class.

MORE INFORMATION

The following five steps are necessary to register a new class that uses some information from the standard windows "button" class:

1. Call GetClassInfo() to fill the WNDCLASS structure.

2. Save the cbWndExtra value in a global variable.

3. Add the desired number of bytes to the existing cbWndExtra value.

4. Change the lpszClassName field.

5. Call RegisterClass() to register the new class.

The first step will fill the WNDCLASS structure with the data that was used when the class was originally registered. In this example, the second step is necessary so that when the "new" extra bytes are accessed, the original extra bytes are not destroyed. Please note that it is NOT safe to assume that the original cbWndExtra value was zero. When accessing the "new" extra bytes, it is necessary to use the original value of cbWndExtra as the base for any new data stored in the extra bytes. The third step allocates the new extra bytes. The fourth step specifies the new name of the class to be registered, and the final step actually registers the new class.

Any new class created in this manner MUST have a unique class name. Typically, this name would be similar but not identical to the original

class. For example, to superclass a button, an appropriate class name might be "superbutton." There is no conflict with class names used by other applications as long as the CS_GLOBALCLASS class style is not specified. The standard Windows "button" class remains unchanged and can still be used by the application as normal. In addition, once a new class has been registered, any number of controls can be created and destroyed with no extra coding effort. The superclass is simply another class in the pool of classes that can be used when creating a window.

The sample code below demonstrates this procedure:

```
BOOL DefineSuperButtonClass(void)
{
#define MYEXTRABYTES 8

   HWND     hButton;
   WNDCLASS wc;

   GetClassInfo(NULL, "button", (LPWNDCLASS)&wc);

   iStdButtonWndExtra = wc.cbWndExtra;   // Save this in a global

   wc.cbWndExtra += MYEXTRABYTES;

   lstrcpy((LPSTR)wc.lpszClassName, (LPSTR)"superbutton");

   return(RegisterClass((LPWNDCLASS)&wc));
}
```

It is important to note that the lpszClassName, lpszMenuName, and hInstance fields in the WNDCLASS structure are NOT returned by the GetClassInfo() function. Please refer to page 4-153 of the "Microsoft Windows Software Development Kit Reference Volume 1" for more information. Also, each time a new class is registered, scarce system resources are used. If it is necessary to alter many different standard classes, the GetProp(), SetProp(), and RemoveProp() functions should be used as an alternative approach to associating extra information with standard Windows controls.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Extracting the SID from an ACE

PSS ID Number: Q102101
Authored 28-Jul-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

To access the security identifier (SID) contained in an access control entry (ACE), the following syntax can be used:

```
PSID pSID;

if((((PACE_HEADER)pTempAce)->AceType) == ACCESS_ALLOWED_ACE_TYPE)
{
   pSID=(PSID)&((PACCESS_ALLOWED_ACE)pTempAce)->SidStart;
}
```

MORE INFORMATION

The "if" statement checks the type of ACE, which is one of the following values:

```
ACCESS_ALLOWED_ACE_TYPE
ACCESS_DENIED_ACE_TYPE
SYSTEM_AUDIT_ACE_TYPE
```

The conditional statement casts pTempAce (the pointer to the ACE) to a PACCESS_ALLOWED_ACE structure. The address of the SidStart member is then cast to a PSID and assigned to the pSID variable. pSID can now be used with any Win32 Security application programming interface (API) that takes a PSID as a parameter.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

File Manager Passes Short Filename as Parameter

PSS ID Number: Q98575
Authored 09-May-1993 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 3.1 and 3.5

When starting an application from File Manager by double-clicking a document associated with the application, if the document resides on an NTFS partition and has a long (non-8.3 form) filename, File Manager will pass the short version of the filename (also known as the MS-DOS alias or 8.3 name) to the associated application if the application is an MS-DOS or 16-bit Windows-based application. This is done for compatibility reasons; applications not aware of long filenames (16-bit Windows-based applications) can still function correctly. 32-bit Windows-based applications will be passed the long file name.

This can create confusion, however, if the application displays the name of the file the application was started with; the short name is displayed even though the long name was double-clicked.

You can avoid possible confusion by always expanding any filenames passed to an application via the command line. Do this by calling the FindFirstFile() application programming interface (API) on these filenames. FindFirstFile() will always return the file system's version of the filename in the WIN32_FIND_DATA.cFileName structure member, which the application can then use in all further references to the file without any problems.

Additional reference words: 3.10 file name 3.50
KBCategory: kbprg
KBSubcategory: BseFileio

FILE_FLAG_WRITE_THROUGH and FILE_FLAG_NO_BUFFERING

PSS ID Number: Q99794
Authored 08-Jun-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1 and 3.5

SUMMARY

The FILE_FLAG_WRITE_THROUGH flag for CreateFile() causes any writes made to that handle to be written directly to the file without being buffered. The data is cached (stored in the disk cache); however, it is still written directly to the file. This method allows a read operation on that data to satisfy the read request from cached data (if it's still there), rather than having to do a file read to get the data. The write call doesn't return until the data is written to the file. This applies to remote writes as well--the network redirector passes the FILE_FLAG_WRITE_THROUGH flag to the server so that the server knows not to satisfy the write request until the data is written to the file.

The FILE_FLAG_NO_BUFFERING takes this concept one step further and eliminates all read-ahead file buffering and disk caching as well, so that all reads are guaranteed to come from the file and not from any system buffer or disk cache. When using FILE_FLAG_NO_BUFFERING, disk reads and writes must be done on sector boundaries, and buffer addresses must be aligned on disk sector boundaries in memory.

These restrictions are necessary because the buffer that you pass to the read or write API is used directly for I/O at the device level; at that level, your buffer addresses and sector sizes must satisfy any processor and media alignment restrictions of the hardware you are running on.

MORE INFORMATION

This code fragment demonstrates how to sector-align data in a buffer and pass it to CreateFile():

```
  char buf[2 * SECTOR_SIZE - 1], *p;

  p = (char *) ((DWORD) (buf + SECTOR_SIZE - 1) & ~(SECTOR_SIZE - 1));
  h = CreateFile(argv[1], GENERIC_READ | GENERIC_WRITE,
      FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, CREATE_ALWAYS,
      FILE_ATTRIBUTE_NORMAL | FILE_FLAG_NO_BUFFERING, NULL);
  WriteFile(h, p, SECTOR_SIZE, &dwWritten, NULL);
```

The pointer p is sector-aligned and points within the buffer.

If you have a situation where you want to flush all open files on the current logical drive, this can be done by:

```
hFile = CreateFile("\\\\.\\c:", ....);
FlushFileBuffers(hFile);
```

This method causes all buffered write data for all open files on the C: partition to be flushed and written to the disk. Note that any buffering done by anything other than the system is not affected by this flush; any possible file buffering that the C Run-time is doing on files opened with C Run-time routines is unaffected.

When opening a remote file over the network, the server always caches and ignores the no buffering flag specified by the client. This is by design. The redirector and server cannot properly implement the full semantics of FILE_FLAG_NO_BUFFERING over the network. In particular, the requirement for sector-sized, sector-aligned I/O cannot be met. Therefore, when a Win32-based application asks for FILE_FLAG_NO_BUFFERING, the redirector and server treat this as a request for FILE_FLAG_WRITE_THROUGH. The file is not cached at the client, writes go directly to the server and to the disk on the server, and the read/write sizes on the network are exactly what the application asks for. However, the file is cached on the server.

Not caching the client can have a different effect, depending on the type of I/O. You eliminate the cache hits or read ahead, but you also may reduce the size of transmits and receives. In general, for sequential I/O, it is a good idea to cache on the client. For small, random access I/O, it is often best not to cache.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseFileio

FILE_READ_EA and FILE_WRITE_EA Specific Types

PSS ID Number: Q102104
Authored 28-Jul-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

The FILE_READ_EA and FILE_WRITE_EA specific types provide access to read and write a file's extended attributes. Specific access types are represented as bits in the access mask and are specific to the object type associated with the mask.

Please note that these specific types are used in the definition of constants such as FILE_GENERIC_READ, and are not intended to be generally used when specifying access (generic access types are much more appropriate).

Additional reference words: 3.10
KBCategory: kbprg
KBSubcategory: BseFileio

Filenames Ending with Space or Period Not Supported

PSS ID Number: Q115827
Authored 05-Jun-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

CreateFile() removes trailing spaces and periods from file and directory names. This is done for compatibility with the FAT and HPFS file systems.

Problems can arise when a Macintosh client creates a file on a Windows NT server. The code to remove trailing spaces and periods is not carried out and the Macintosh user gets the correctly punctuated filename. The Win32 APIs FindFirstFile() and FindNextFile() return a filename that ends in a space or in a period; however, there is no way to create or open the file using the Win32 API.

Applications such as File Manager and Backup check to see whether the filename ends with a space or period. If the filename does end in a space or a period, then File Manager and Backup use the alternative name found in WIN32_FIND_DATA.cAlternateFileName to create and open the file. Therefore, the full filename is lost.

Additional reference words: 3.10 3.50 4.00 95 winfile ntbackup
KBCategory: kbprg
KBSubcategory: BseFileio

FileTimeToLocalFileTime() Adjusts for Daylight Saving Time

PSS ID Number: Q128126
Authored 27-Mar-1995 Last modified 27-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 3.5

SUMMARY

Under NTFS, the API GetFileTime() returns the create time, last access time, and last write time for the specified file. The times returned in the FILETIME structures are in Universal Coordinated Time (UTC). This is also the time that NTFS uses. You can use FileTimeToLocalFileTime() to convert a file time to a local time. However, if you automatically adjust for Daylight Saving Time, FileTimeToLocalFileTime() will adjust for Daylight Saving Time based on whether the current date should be adjusted for Daylight Saving Time, not based on whether the date represented by the FILETIME structure should be adjusted.

The behavior in this situation is different under FAT, but may be changed to match the behavior under NTFS in a future version of Windows NT.

MORE INFORMATION

The result of this behavior, which is by design, is that reported file times under NTFS may change with the start and end of Daylight Saving Time. For example, suppose that the file TEST.C has a last write FILETIME representing Jan 1, 1995 9:00pm (UTC), it is not Daylight Saving Time, and you are in the Pacific time zone. Both the DIR command and the following sample code report the file time as 1:00pm (LocalTime = UTC - 8).

Sample Code 1

```
#include <windows.h>

void main()
{
   HANDLE hFile;
   FILETIME ftCreate, ftLastAccess, ftLastWrite, ftLocal;
   SYSTEMTIME st;

   char buf[80];

// Open the file.

   hFile = CreateFile( "test.c",
                       GENERIC_READ,
                       0,
                       NULL,
```

```
                        OPEN_EXISTING,
                        FILE_ATTRIBUTE_NORMAL,
                        NULL );

   // Get the file time (in UTC) and convert to local time.

      GetFileTime( hFile, &ftCreate, &ftLastAccess, &ftLastWrite );
      FileTimeToLocalFileTime( &ftLastWrite, &ftLocal );

   // Display the time, as a test.

      FileTimeToSystemTime( &ftLocal, &st );
      GetTimeFormat( LOCALE_USER_DEFAULT, 0, &st, NULL, buf, sizeof(buf) );
      MessageBox( NULL, buf, " FILE TIME", MB_OK );
   }
```

Now, set the date to 7/1/95 and enable Automatically Adjust for Daylight Saving Time. The DIR command and the sample code above will report the file time as 2:00pm, because FileTimeToLocalFileTime() has adjusted for Daylight Saving Time (LocalTime = UTC - 7).

The following sample code will correctly report the file time of TEST.C with the date set to 7/1/95 under NTFS. The FILETIME structure is converted to a SYSTEMTIME structure with FileTimeToSystemTime(). Then the time is converted using SystemTimeToTzSpecificLocalTime(). If you need to convert back to a FILETIME structure, use SystemTimeToFileTime() after the conversion to local time.

Sample Code 2

```
   #include <windows.h>

   void main()
   {
      HANDLE hFile;
      FILETIME ftCreate, ftLastAccess, ftLastWrite;
      SYSTEMTIME stUTC, st;
      char buf[80];

   // Open the file.

      hFile = CreateFile( "test.c",
                        GENERIC_READ,
                        0,
                        NULL,
                        OPEN_EXISTING,
                        FILE_ATTRIBUTE_NORMAL,
                        NULL );

   // Get the file time (in UTC) and convert to local time.

      GetFileTime( hFile, &ftCreate, &ftLastAccess, &ftLastWrite );
      FileTimeToSystemTime( &ftLastWrite, &stUTC );
      SystemTimeToTzSpecificLocalTime( NULL, &stUTC, &st);
```

```
   // Display the time, as a test.

      GetTimeFormat( LOCALE_USER_DEFAULT, 0, &st, NULL, buf, sizeof(buf) );
      MessageBox( NULL, buf, "FILE TIME", MB_OK );
   }
```

The FAT file system stores local time, not UTC. GetFileTime() gets cached UTC times from FAT. In this sample, the time stored is 1pm and the cached time is 9pm. When it becomes Daylight Saving Time, sample codes 1 and 2 will demonstrate the same behavior that they do under NTFS, because 9pm is still used. However, when you restart the machine, the new cached time will be 8pm (UTC = LocalTime + 7). The call to FileTimeToLocalFileTime() cancels the adjustment made by GetFileTime() (LocalTime = UTC - 7). Therefore, sample code 1 will report the correct time under FAT, but sample code 2 will not.

On the other hand, FindFirstFile() on FAT always reads the time from the file (stored as local time) and converts it into UTC, adjusting for DST based on the current date. So if FindFirstFile() is called, the date is changed to a different DST season, and then FindFirstFile() is called again, the UTC returned by the two calls will be different.

Additional reference words: 3.50
KBCategory: kbprg kbcode
KBSubcategory: BseMisc

First and Second Chance Exception Handling

PSS ID Number: Q105675
Authored 22-Oct-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

Structured exception handling (SEH) takes a little getting used to, particularly when debugging. It is common practice to use SEH as a signaling mechanism. Some application programming interfaces (APIs) register an exception handler in anticipation of a failure condition that is expected to occur in a lower layer. When the exception occurs, the handler may correct or ignore the condition rather than allowing a failure to propagate up through intervening layers. This is very handy in complex environments such as networks where partial failures are expected and it is not desirable to fail an entire operation simply because one of several optional parts failed. In this case, the exception can be handled so that the application is not aware that an exception has occurred.

However, if the application is being debugged, it is important to realize that the debugger will see all exceptions before the program does. This is the distinction between the first and second chance exception. The debugger gets the "first chance," hence the name. If the debugger continues the exception unhandled, the program will see the exception as usual. If the program does not handle the exception, the debugger will see it again (the "second chance"). In this latter case, the program normally would have crashed had the debugger not been present.

If you do not want to see the first chance exception in the debugger, then disable the feature. Otherwise, during execution, when the debugger gets the first chance, continue the exception unhandled and allow the program to handle the exception as usual. Check the documentation for the debugger that you are using for descriptions of the commands to be used.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseExcept

FIX: AllocConsole() Does Not Set Error Code on Failure

PSS ID Number: Q105564
Authored 20-Oct-1993 Last modified 19-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SYMPTOMS

The documentation for AllocConsole() states the following:

> If the function succeeds, the return value is TRUE; otherwise, it is FALSE. To get extended error information, use the GetLastError() function.

Upon failure, AllocConsole() does not set the error code.

STATUS

Microsoft has confirmed this to be a bug in Windows NT 3.1. This problem was corrected in Windows NT 3.5.

MORE INFORMATION

The following sample code demonstrates the problem:

```
#include <stdio.h>
#include <windows.h>

void main()
{
   BOOL bSuccess;

/* Comment out the following line and GetLastError() will   */
/* return 999; otherwise, GetLastError() returns 1812.      */

   FreeConsole();

   SetLastError( 999 );

   bSuccess = AllocConsole();
   if( !bSuccess )
      puts( "AllocConsole failed" );
   printf( "The last error is: %d\n", GetLastError() );
   getchar();
}
```

Additional reference words: 3.10

KBCategory: kbprg kbfixlist kbbuglist
KBSubcategory: BseCon

FIX: APIs Do Incorrect Comparisons

PSS ID Number: Q105804
Authored 25-Oct-1993 Last modified 28-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SYMPTOMS

The functions getservbyport() and getprotobynumber() do an incorrect comparison when two port APIs take a parameter as an integer then do comparisons of the integer against an unsigned short that is incorrectly sign-extended. If a port or protocol number greater than 127 is passed into these routines, they will not return any information.

STATUS

Microsoft has confirmed this to be a problem in Windows NT and Windows NT Advanced Server version 3.1. This problem was corrected in the latest U.S. Service Pack for Windows NT and Windows NT Advanced Server version 3.1. For information on obtaining the Service Pack, query on the following word in the Microsoft Knowledge Base (without the spaces):

S E R V P A C K

Additional reference words: 3.10 buglist3.10 fixlist3.10.001
KBCategory: kbprg kbbuglist kbfixlist
KBSubCategory: NtwkWinsock

FIX: Cannot Compile from Win32 SDK M Editor (MEP.EXE)

PSS ID Number: Q98918
Authored 18-May-1993 Last modified 18-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SYMPTOMS

The Win32 Software Development Kit (SDK) M Editor compile function uses the correct extmake line and spawns the compiler correctly, but does not produce the desired result. The terminating beep is issued, but no messages appear in the <compile> pseudo file and no object file is produced.

STATUS

Microsoft has confirmed this to be a problem in the products listed at the beginning of this article. This problem was corrected in the Win32 SDK version 3.5.

MORE INFORMAION

The method typically used to invoke the compiler from the M Editor is

1. Include the following text switch in the TOOLS.INI

   extmake:<ext> <command>

   where <ext> is the file extension and <command> is the command to be carried out when compiling a file of this type. For example:

   extmake:c cl /DWIN32 %s

   NOTE: %s is the current file.

2. To invoke the switch, use Arg Compile (ALT+A CTRL+F3)

Additional reference words: 3.10 mep
KBCategory: kbtool kbprb
KBSubcategory: TlsMep

FIX: GetPrivateProfileSection() Can Read Only 32K Sections

PSS ID Number: Q105681
Authored 22-Oct-1993 Last modified 28-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SYMPTOMS

The documentation for GetProfileSection(), GetProfileString(), GetPrivateProfileSection(), GetPrivateProfileString(), and GetPrivateProfileSection() indicate that the application programming interface (API) can read all the keys and values of a section, regardless of size. However, these functions only seem to handle sections that are smaller than 32K, even though the size of the buffer is a DWORD.

CAUSE

The code is casting this value to a signed short, and therefore you are having problems with sections that are greater than 32K in size.

STATUS

Microsoft has confirmed this to be a problem in Windows NT 3.1. This problem was corrected in Windows NT 3.5.

Additional reference words: 3.10
KBCategory: kbprg kbfixlist kbbuglist
KBSubcategory: BseMisc

FIX: Global Constructors Not Called in Alpha DLLs

PSS ID Number: Q98732
Authored 12-May-1993 Last modified 29-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SYMPTOMS

C++ constructors on global variables are not invoked when a dynamic link library (DLL) is loaded on the Alpha AXP platform.

STATUS

Microsoft has confirmed this to be a problem in Windows NT and Windows NT Advanced Server version 3.1. This problem was corrected in the latest U.S. Service Pack for Windows NT and Windows NT Advanced Server version 3.1. For information on obtaining the Service Pack, query on the following word in the Microsoft Knowledge Base (without the spaces):

S E R V P A C K

Additional reference words: 3.10 buglist3.10 fixlist3.10.003
KBCategory: kbtool kbfixlist kbbuglist
KBSubCategory: TlsMisc

FIX: High CPU Usage When SNMP Retrieves Performance Counters

PSS ID Number: Q130563
Authored 23-May-1995 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 3.5

SYMPTOMS

When an SNMP extension agent tries to retrieve performance counters, CPU utilization jumps to 100 percent.

STATUS

Microsoft has confirmed this to be a bug in the Microsoft products listed at the beginning of this article. This problem was corrected in Windows NT version 3.51.

MORE INFORMATION

If an SNMP extension agent tries to retrieve performance counters as in the following example, the CPU utilization is observed to jump to 100 percent and stay there until the SNMP service is stopped.

```
RegOpenKeyEx(...);
.
.
RegQueryValueEx(HKEY_PERFORMANCE_DATA,
...)
```

To see this, use the Performance Monitor tool in the Administrative Tools program group.

The same problem can occur when an application is trying to retrieve TCP/IP performance counters and the Internet MIB II agent is being used to retrieve the counters.

REFERENCES

For more information, please see the Windows NT 3.5 Resource Kit Vol IV.

Additional reference words: 3.50
KBCategory: kbnetwork kbfixlist kbbuglist
KBSubcategory: NtwkSnmp

FIX: Internal Error from NetUserEnum()

PSS ID Number: Q108055
Authored 02-Dec-1993 Last modified 29-Nov-1994

--
The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

--

SYMPTOMS

If you make a NetUserEnum() call from a down-level computer remoted to a Windows NT machine, the error return is:

2140 (NERR_InternalError).

This happens only if the down-level computer remotes the API to the Windows NT computer and provides a buffer size that is greater than zero but less than the size of one element of the data structure.

STATUS

Microsoft has confirmed this to be a problem in Windows NT and Windows NT Advanced Server version 3.1. This problem was corrected in the latest U.S. Service Pack for Windows NT and Windows NT Advanced Server version 3.1. For information on obtaining the Service Pack, query on the following word in the Microsoft Knowledge Base (without the spaces):

S E R V P A C K

Additional reference words: 3.10 buglist3.10 fixlist3.10.003
KBCategory: kbtool kbbuglist kbfixlist
KBSubCategory: TlsLmapi

FIX: Low Memory Condition Cause APIs to Return Random Values

PSS ID Number: Q105575
Authored 20-Oct-1993 Last modified 29-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SYMPTOMS

Low memory on a Windows NT computer can cause USER APIs to return random values. For example, CreateWindowEx() returns a bad value, and SetWindowLong() and GetWindowLong() do not receive the correct value.

STATUS

Microsoft has confirmed this to be a problem in Windows NT and Windows NT Advanced Server version 3.1. This problem was corrected in the latest U.S. Service Pack for Windows NT and Windows NT Advanced Server version 3.1. For information on obtaining the Service Pack, query on the following word in the Microsoft Knowledge Base (without the spaces):

S E R V P A C K

Additional reference words: 3.10 buglist3.10 fixlist3.10.001
KBCategory: kbprg kbbuglist kbfixlist
KBSubCategory: UsrMisc

FIX: MFC-Based App Cannot Run Under International Win32s

PSS ID Number: Q125457
Authored 29-Jan-1995 Last modified 11-Apr-1995

The information in this article applies to:

- Microsoft Win32s version 1.2

SYMPTOMS

When activating a Microsoft Foundations Classes (MFC)-based application built with Visual C++ version 2.0 under Win32s version 1.2, if the language setting in Windows or Windows for Workgroups Control panel is set to anything other than English, German, or French, the following error occurs:

```
Win32s - Error: Initialization of a dynamic link library failed.
The process is terminating abnormally.
```

This error has also been reported with non-MFC-based applications as well.

CAUSE

CRT startup code causes an unhandled exception in the Win32s DLL (W32SCOMB.DLL).

RESOLUTION

For an international version of Windows 3.1 or Windows for Workgroups, if the country and language settings are changed to United States and English(American), 32-bit MFC-based applications are able to run.

STATUS

Microsoft has confirmed this to be a bug in Win32s version 1.2. This problem has been corrected in Win32s version 1.25.

Additional reference words: 1.20 w32scomb
KBCategory: kbinterop kbfixlist kbbuglist
KBSubcategory: W32s WIntlDev

FIX: MIDL 2.0 Does Not Handle VAX Floating Point Numbers

PSS ID Number: Q129064
Authored 18-Apr-1995 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 3.5

SYMPTOMS

RPC applications (client and server) built using MIDL 2.0 (shipped with Windows NT version 3.5) may get spurious characters or report memory violations when dealing with VAX floating point numbers.

RESOLUTION

There are two possible resolutions to this problem:

- Use MIDL 1.0 (shipped with Windows NT version 3.1) to build RPC client and server applications.

  -or-

- On the VAX side, use a compiler switch to make sure the application represents its floating point numbers in IEEE format.

STATUS

Microsoft has confirmed this to be a bug in the Microsoft products listed at the beginning of this article. This problem was corrected in Windows NT version 3.51.

MORE INFORMATION

Note that Microsoft RPC is binary compatible with DCE RPC. There are four types of representations for floating point numbers: IEEE, VAX, IBM, and CRAY. An application built using MIDL 1.0 can recognize IEEE and VAX representations but not IBM and CRAY. Applications built using MIDL 2.0 can recognize only IEEE representaion as mentioned above. Both, MIDL 1.0 and 2.0, use IEEE representations to send data.

Additional reference words: 3.50
KBCategory: kbnetwork kbfixlist kbbuglist
KBSubcategory: NtwkRpc

FIX: MIPS Compiler Assertion with C version 8.0

PSS ID Number: Q102764
Authored 10-Aug-1993 Last modified 29-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SYMPTOMS

You may get the following assertion from the C version 8.0 compiler on Windows NT under MIPS:

```
Microsoft (R) C Centaur Optimizing Compiler Version 8.00.081
Copyright (c) Microsoft Corp 1984-1992. All rights reserved.

t516a.cxx
Assertion failed: prld->symbol->symbolnumber >= 0, file
E:\src\centaur\drop81\msas\as1coff.c, line 459

abnormal program termination
```

CAUSE

You can get an assertion running C 8.0 on Windows NT MIPS if you did not compile with /Zi or /Z7 or if your external declarations are not at file scope level (that is, declare them as global).

RESOLUTION

The easiest workaround is to recompile with /Zi or /Z7. Another workaround is to move all extern declarations to file scope level.

STATUS

There is a fix for this but the problem wasn't deemed critical enough to change right before the Windows NT 3.1 release. This problem was corrected in the Win32 SDK, version 3.5.

Additional reference words: 3.10
KBCategory: kbtool kbfixlist kbbuglist
KBSubcategory: TlsMisc

FIX: NDISCloseFile() Does Not Free Image Buffer

PSS ID Number: Q105785
Authored 24-Oct-1993 Last modified 28-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SYMPTOMS

The NDISCloseFile() function does not free the buffer that it allocates to hold the image that was read in.

STATUS

Microsoft has confirmed this to be a problem in Windows NT and Windows NT Advanced Server version 3.1. This problem was corrected in the latest U.S. Service Pack for Windows NT and Windows NT Advanced Server version 3.1. For information on obtaining the Service Pack, query on the following word in the Microsoft Knowledge Base (without the spaces):

S E R V P A C K

Additional reference words: 3.10 buglist3.10 fixlist3.10.001
KBCategory: kbprg kbbuglist kbfixlist
KBSubCategory: NtwkMisc

FIX: Owner Drawn Items on the Menu Bar Hang Windows NT

PSS ID Number: Q126720
Authored 28-Feb-1995 Last modified 03-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5

SYMPTOMS

Windows NT supports owner drawn items on the menu bar. However, Windows NT may stop responding (hang) if multiple windows of same class are created with the owner drawn menu bar.

STATUS

Microsoft has confirmed this to be a bug in Windows NT version 3.50. This problems was corrected in Windows NT version 3.51.

MORE INFORMATION

Steps to Reproduce Problem
---

1. Assign a menu to a window class via RegisterClass().

2. Modify the menu bar to be owner draw in WM_CREATE.

3. Create multiple windows of the same window class.

Windows NT may stop responding, in which case, you must turn off the computer.

Additional reference words: 3.50
KBCategory: kbprg kbfixlist kbbuglist
KBSubcategory: UsrMen UsrOwn

FIX: POSIX Does Not Distinguish stdout & stderr

PSS ID Number: Q92769
Authored 15-Nov-1992 Last modified 29-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SYMPTOMS

Programs under POSIX do not distinguish between stdout and stderr. This causes a variety of symptoms, such as fprints to stdout and stderr ending up in the same place.

CAUSE

When output is destined for the terminal (or a pipe to a windows application), the output becomes generic output instead of being destined for a specific file descriptor, and so forth.

STATUS

Microsoft has confirmed this to be a problem in Windows NT, version 3.1. This problem was corrected in Windows NT, version 3.5.

MORE INFORMATION

The following sample code can be used to demonstrate the problem.

Sample code

```
#include <stdio.h>

void main( )
{
   freopen( "freopen.out", "w", stdout );
   fprintf( stdout, "Hey there.\n" );
   fprintf( stderr, "Hey! Not me too!\n" );
}
```

Additional reference words: 3.10 buglist3.10 fixlist3.50
KBCategory: kbprg kbbuglist kbfixlist
KBSubcategory: SubSystem

FIX: Redirecting Output to an MS-DOS-Based Application

PSS ID Number: Q105303
Authored 17-Oct-1993 Last modified 28-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SYMPTOMS

The following method is used to redirect output to a child process when it is started from a GUI application:

1. Declare STARTUPINFO si.

   a. Set the hStdIn, hStdOut, and/or hStdErr field as desired.
   b. Set the dwFlags field to STARTF_USESTDHANDLES.

2. In CreateProcess(), set inherit handles to TRUE.

NOTE: This method does not work when starting MS-DOS-based applications.

RESOLUTION

As a workaround, use AllocConsole(), then the following method:

1. Use SetStdHandle() to set the desired handles to be inherited.

   -or-

   Use DuplicateHandle() to change the inheritance property of handles that should not be inherited.

3. In CreateProcess(), set inherit handles to TRUE.

This method creates a blank console window; however, this is necessary because the method doesn't work otherwise.

STATUS

Microsoft has confirmed this to be a bug in Windows NT 3.1. This problem was corrected in Windows NT 3.5.

MORE INFORMATION

Note that if you are opening a handle that will be inherited by the child, set SECURITY_ATTRIBUTES.bInheritHandle = TRUE in the call to CreateFile(), CreatePipe(), and so forth.

For an example of both methods of redirection, see the INHERIT SDK sample.

Additional reference words: 3.10 buglist3.10 fixlist3.50
KBCategory: kbprg kbfixlist kbbuglist
KBSubcategory: BseProcThrd

FIX: SetCaret API May Not Work Correctly in Win32-Based Apps

PSS ID Number: Q105733
Authored 24-Oct-1993 Last modified 28-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SYMPTOMS

The SetCaret() API does not work correctly in multithreaded Win32-based applications. Code with the following logic fails:

Thread 1: Creates a window and starts another thread.

Thread 2: Uses an API to synchronize the two threads' "Input State," attempts to write to the window of Thread 1, and then tries to set the caret through SetCaret().

STATUS

Microsoft has confirmed this to be a problem in Windows NT and Windows NT Advanced Server version 3.1. This problem was corrected in the latest U.S. Service Pack for Windows NT and Windows NT Advanced Server version 3.1. For information on obtaining the Service Pack, query on the following word in the Microsoft Knowledge Base (without the spaces):

S E R V P A C K

Additional reference words: 3.10 buglist3.10 fixlist3.10.001
KBCategory: kbprg kbbuglist kbfixlist
KBSubCategory: GdiCurico

FIX: Setsockopt() for Winsock over Appletalk Returns Error

PSS ID Number: Q129062
Authored 18-Apr-1995 Last modified 06-Jun-1995

The information in this article applies to

- Microsoft Win32 Software Development Kit (SDK) version 3.5

SYMPTOMS

After you open a socket of type SOCK_STREAM by using the ATPROTO_ADSP protocol, and bind to a dynamic socket, the setsockopt() function fails under these conditions:

- The setsockopt() function uses the zone name returned by getsockopt() (also found by looking at the control panel network entry) with the SO_LOOKUP_MYZONE option.

- The zone name is supplied to setsockopt() with the SO_REGISTER_NAME option.

Error code 10022 (WSAINVAL :Invalid Argument) is returned on calling GetLastError().

RESOLUTION

Instead of passing the string returned by getsockopt() for the zone name, use the character "*" for the ZoneName member of the WSH_REGISTER_NAME struct. For example, use this:

```
WSH_REGISTER_NAME regName;
........
strcpy( regName.ZoneName, "*");
```

instead of this:

```
strcpy( regName.ZoneName, "BLDG/1");
```

STATUS

Microsoft has confirmed this to be a bug in the Microsoft products listed at the beginning of this article. This problem was corrected in Windows NT 3.51.

REFERENCES

Windows Sockets for Appletalk (SFMWSHAT.WRI version 1.2).

Additional reference words: 3.50

KBCategory: kbnetwork kbfixlist kbbuglist
KBSubcategory: NtwkWinsock

FIX: SNMP Sample Generates an Application Error

PSS ID Number: Q124961
Authored 17-Jan-1995 Last modified 07-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 3.5

SYMPTOMS

The Win32 SDK for Windows NT 3.5 contains a SNMP sample called SNMPUTIL (\MSTOOLS\SAMPLES\WIN32\SNMP\SNMPUTIL). Building this sample generates an executable file that causes an access violation. The message box displayed resembles the following:

```
                    snmputil.exe - Application Error
   ------------------------------------------------------------------
   The instruction at <address> referenced memory at <address>.  The
   memory could not be "read".
```

CAUSE

One part of the SNMP run times uses the C run-time routines provided by CRTDLL.DLL. The other part of the SNMP run times uses the C run-time routines that the application uses. Because many people are using Visual C++ to build the samples, the run time routines used are in MSVCRT20.DLL. The access violation occurs when memory allocated using one run time version are freed using a different copy of the run time library.

RESOLUTION

One possible solution is to build the sample application using the CRTDLL.DLL version of the run time library. If you are using the makefile supplied with the Win32 SDK, one way to accomplish this is to:

1. Copy NTWIN32.MAK from your include directory to the working directory for the sample.
2. Change all references to MSVCRT.LIB in NTWIN32.MAK to CRTDLL.LIB.

If you are using a Visual C++ make file, you can:

1. Select the "Ignore All Default Libraries" check box in the Linker Project Settings property page.
2. Add CRTDLL.LIB to the "Object/Library Modules" section in the Linker Project Settings property page.

NOTE: CRTDLL.LIB does not ship with Visual C++. You can get the CRTDLL.LIB file from the Win32 SDK in the \MSTOOLS\LIB\CRT\<PLATFORM> directory, where <PLATFORM> would be replaced with the type of machine you are using (I386, alpha, and so forth).

If you are using MFC 3.0, you will not be able to use CRTDLL.DLL because MFC 3.0 uses some functions that are not supplied in CRTDLL.DLL.

STATUS

Microsoft has confirmed that this is a problem in the Microsoft products listed at the beginning of this article. This problem was corrected in the Win32 SDK version 3.51.

Additional reference words: 3.50
KBCategory: kbprg kbnetwork kbfixlist kbbuglist
KBSubcategory: NtwkSnmp

FIX: StreBlt Sample Causes Windows NT to Stop Responding

PSS ID Number: Q104834
Authored 29-Sep-1993 Last modified 28-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SYMPTOMS

When you run the STREBLT sample, Windows NT may stop responding (hang).

STATUS

This problem was corrected in Windows NT 3.5.

MORE INFORMATION

Steps to Reproduce Problem
---

1. Build the StreBlt sample using the supplied makefile.

2. The sample displays a dialog box for the main window. There are three drop-down list controls titled StretchBltMode, Pattern, and Standard ROPs. Select the following in these drop-down lists:

StretchBltMode	BLACKONWHITE
Pattern	NULL_BRUSH
Standard ROPs	PATPAINT

At this point, Windows NT stops responding.

Additional reference words: 3.10
KBCategory: kbprg kbfixlist kbbuglist
KBSubCategory: GdiBmp

FIX: SVCGUID.H Has Wrong UDP Port for SNMP Traps

PSS ID Number: Q129061
Authored 18-Apr-1995 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 3.5

SYMPTOMS

The file SVCGUID.H used by the Registration and Name Resolution (RnR) APIs has the UDP trap port for SNMP set to 167:

```
#define SVCID_SNMP_TRAP_UDP                    SVCID_UDP(167)
```

RESOLUTION

The correct port number for SNMP traps is 162. However, please note that this entry is not looked at by the SNMP service for Windows NT. Hence, it cannot cause traps to be sent to the wrong port. SVCGUID.H is used only by Registration and Name Resolution (RnR) APIs.

STATUS

Microsoft has confirmed this to be a bug in the Microsoft products listed at the beginning of this article. This problem was corrected in Windows NT 3.51.

Additional reference words: 3.50
KBCategory: kbnetwork kbfixlist kbbuglist
KBSubcategory: NtwkWinsock

FIX: VirtualLock() on File-Mapped Pages Hangs Computer

PSS ID Number: Q107642
Authored 23-Nov-1993 Last modified 28-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SYMPTOMS

Your computer may stop responding (hang) if you use the VirtualLock() application programming interface (API) command to lock a file-mapped page.

STATUS

Microsoft has confirmed this to be a problem in Windows NT and Windows NT Advanced Server version 3.1. This problem was corrected in the latest U.S. Service Pack for Windows NT and Windows NT Advanced Server version 3.1. For information on obtaining the Service Pack, query on the following word in the Microsoft Knowledge Base (without the spaces):

S E R V P A C K

Additional reference words: 3.10 buglist3.10 fixlist3.10.003
KBCategory: kbprg kbfixlist kbbuglist
KBSubCategory: BseMm

FIX: Win32s 1.25a Fix List

PSS ID Number: Q130139
Authored 11-May-1995 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Win32s, version 1.25a

The following is a list of the known bugs in Win32s version 1.2 that were fixed in Win32s version 1.25.

- GlobalAlloc(GMEM_FIXED) from 32-bit .EXE locks memory pages. It is more efficient to use GlobalAlloc(GMEM_MOVEABLE) and call GlobalFix() if necessary.

- WINMM16.DLL has no version information.

- CreateFileMapping() with SEC_NOCOMMIT returns ERROR_INVALID_PARAMETER.

- PolyPolygon() does not close the polygons.

- OpenFile() only searches the current directory when only a filename is given, not the application directory, the system directory, the windows directory, or the directories listed on the path.

- GetFileInformationByHandle() doesn't return the correct file attribute.

- GlobalUnlock() sets an error of ERROR_INVALID_PARAMETER.

- lstrcmp()/lstrcmpi() do not use the collate table correctly.

- FreeLibrary() in DLL_PROCESS_DETACH crashes the system.

- FindResource(), LoadResource(), GetProcAddress(), GetModuleFileName(), EnumResourceNames(), and other APIs, fail with a NULL hInstance.

- sndPlaySound() with SND_ASYNC | SND_MEMORY may cause a crash or may work poorly.

- Thread Local Storage (TLS) data is not initialized to 0 in TlsAlloc().

- The pointer received in the lParam of WM_INITDIALOG in the common dialog hook function becomes invalid in the following messages if the pointer is "remembered" in a static variable.

- Stubbed API GetFileAttributesW() does not return -1 on error.

- Code page CP_MACCP not supported.

- Invalid LCIDs are not recognized.

- CreateFile() fails to open an existing file in share mode.

- GetLocaleInfo() for locale returns system defaults from WIN.INI.

- GetVolumeInformation() fails with ERROR_INVALID_NAME for volumes without a label.

- VirtualProtect() may miss the last page in an address range.

- GetLocaleInfo() returns incorrect information for most non-US locales.

- ANSI/OEM conversions always use code page 437.

- GetProcAddress() for printer driver APIs is case sensitive.

- The LanMan APIs are unsupported, but they return 0, which indicates that the API was successful. They should return NERR_InvalidAPI (2142).

- CreateFileW() returns 0 instead of -1 (HFILE_ERROR).

- lstrcpyn() copies n bytes from source to destination, then appends a NULL terminator, instead of copying n-1 bytes and appending the NULL terminator.

- GetDriveType() doesn't report detecting a CD-ROM or a RAM DISK.

- CRTDLL calls TlsFree() upon each process detach, not just the last.

- If a DllEntryPoint calls FreeLibrary() when using universal thunks, the system can crash.

- Not all 32-bit DLLs have correct version numbers.

- GetCurrentDirectory() returns the wrong directory after calling GetOpenFileName(). The workaround is to call SetCurrentDirectory(".") right after returning from the call to GetOpenFileName().

- RegEnumValue() and other Registry functions return ERROR_SUCCESS even though they are not implemented. Win32s implements only the registry functions supported by Windows.

- AreFileApisANSI()/SetFileApisToANSI()/SetFileApisToOEM() are not exported. AreFileApisANSI() should always return TRUE, SetFileApiToOEM() should always fail, and SetFileApiToANSI() should always succeed.

- SetLocaleInfoW()/SetLocateInfoA() are not implemented.

- GetScrollPos() sets the last error if the scroll position is 0.

- SetScrollPos() sets the last error if the last scroll position is 0.

- LoadString() leaks memory if the string is a null string.

- GetFileVersionInfoSize() fails if the resource section is small and close to the end of the file.

- MoveFile() doesn't call SetLastError() on failure or sets a different

error than on Windows NT.

- SetCurrentDirectory() does not work on a CD-ROM drive.

- GetFileVersionInfoSize() fails if the 2nd parameter is NULL.

- WSOCK32.DLL is missing exported stubs for unimplemented APIs.

- Win32s fails to load 64x64 monochrome (black and white) icons.

- CreateFile() fails when called with a filename with an international character.

- GetCurrentDirectory() returns an OEM string.

- PrintDlg() causes GP fault if hDevMode!=NULL and another printer is selected that uses a larger DevMode buffer.

- Unicode resources are not properly converted to 8-bit characters.

- CreateDC() returns an incorrect DEVMODE. This can cause a variety of symptoms, like the inability to do a Landscape Print Preview from an MFC application or the displayed paper width and height not changing, even when you change the paper size.

- OpenFile() fails on filenames with OEM characters in the name.

- GetDriveType() fails on a Stacker 3.1 drive.

- SetCurrentDirectory() fails on Novell client machines.

- GetProp() returns 0 in the second instance of an app in certain cases.

- fopen(fn, "w") fails on second call.

- TLS indices allocated by a module are released when that module is freed.

- CreateWindow() handles STARTUPINFO incorrectly if the application starts minimized.

- CreateFile() creates files with incorrect attributes.

- Resource sections are now read/write to emulate the behavior of Windows NT and Windows 95.

- Removed the 128K stack limitation.

- CompareStringW() sometimes uses incorrect locale, primarily Swedish and other Scandinavian locales.

- Added dummy _iob to CRTDLL for applications that reference standard handles.

- Added support for OPENCHANNEL, CLOSECHANNEL, SEXGDIXFORM, and

DOWNLOADHEADER escapes.

Additional reference words: 1.25 1.25a
KBCategory: kbprg kbfixlist kbbuglist
KBSubcategory: W32s

FIX: WinDbg FIND Dialog Box Slows Down the System

PSS ID Number: Q105586
Authored 20-Oct-1993 Last modified 29-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SYMPTOMS

Leaving the WinDbg FIND dialog box displayed after searching for some string causes the CPU to go to 100 percent busy, making the system very slow.

RESOLUTION

Choosing Cancel in the FIND dialog box allows the system to return to normal.

STATUS

Microsoft has confirmed this to be a problem in WinDbg. This problem was corrected in the Win32 SDK, version 3.5.

Additional reference words: 3.10 buglist3.10 fixlist3.50
KBCategory: kbtool kbbuglist kbfixlist
KBSubcategory: TlsWindbg

FIX: Winsock Over Appletalk (DDP) Leaks Memory

PSS ID Number: Q131159
Authored 05-Jun-1995 Last modified 07-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5

SYMPTOMS

A memory leak occurs when you open a socket with address family AF_APPLETALK, type SOCK_DGRAM, and protocol ATPROTO_BASE + x (where x is a user-defined number except the ones that are reserved - please refer to the SDK header file ATALKWSH.H); then once a server is located to send data to, you send data to it.

Observing with PerfMon, you can see that non-paged memory use keeps on increasing. It does not decrease even after the application is stopped. If the application is allowed to continue for a long time, the results are unpredictable and it is neccessary to restart the computer.

STATUS

Microsoft has confirmed this to be a bug in the Microsoft products listed at the beginning of this article. This bug was corrected in Windows NT version 3.51.

Additional reference words: 3.50 3.51
KBCategory: kbnetwork kbfixlist kbbuglist
KBSubcategory: NtwkWinsock

FixBrushOrgEx() and Brush Origins under Win32s

PSS ID Number: Q124191
Authored 21-Dec-1994 Last modified 05-May-1995

The information in this article applies to:

- Microsoft Win32s versions 1.15, 1.15a, and 1.2

FixBrushOrgEx() is not implemented in the Win32 API, but it is provided for compatibility with Win32s. If called, the function does nothing, and returns FALSE.

A brush's origin relates to the origin of the window being painted. If you move a window, the brush origin needs to be updated or else newly painted patterns won't line up with the old patterns. On Windows version 3.1, the system does not automatically update the brush origin when it is selected into a device context (DC), so applications have to call SetBrushOrg(). On Windows NT, the system automatically fixes brush origins when necessary.

Win32s uses FixBrushOrgEx() to hide this difference in system behavior. On Win32s, FixBrushOrgEx() calls SetBrushOrgEx(). A Win32-based application can check the platform and call SetBrushOrgEx() only if it is Win32s, or it could simply always call FixBrushOrgEx() wherever a Windows-based application would call SetBrushOrg() for brush origin tracking.

Additional reference words: 1.20
KBCategory: kbprg
KBSubcategory: W32s

FlushViewOfFile() on Remote Files

PSS ID Number: Q95043
Authored 31-Jan-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

When flushing a memory-mapped file over a network, FlushViewOfFile() guarantees that the data has been written from the workstation, but not that the data resides on the remote disk.

This is because the server may be caching the data on the remote end. Therefore, FlushViewOfFile() may return before the data has been physically written to disk.

However, if the file was created via CreateFile() with the flag FILE_FLAG_WRITE_THROUGH, the remote file system will not perform lazy writes on the file, and FlushViewOfFile() will return when the actual physical write is complete.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseFileio

Format for LANGUAGE Statement in .RES Files

PSS ID Number: Q89822
Authored 30-Sep-1992 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The syntax for the LANGUAGE statement in the resource script file is given as follows on page 289 of the Win32 SDK "Tools User's Guide" manual:

```
LANGUAGE major, minor

major
   Language identifier. Must be one of the constants from WINNLS.H

minor
   Sublanguage identifier. Must be one of the constants from WINNLS.H
```

For example, suppose that you want to set the language for the resources in a file to French. For the major parameter, you would choose the following constant from the list of language identifiers

```
#define LANG_FRENCH                 0x0c
```

and you would have the choice of any of the sublanguages that begin with SUBLANG_FRENCH in the list of sublanguage identifiers. They are:

```
#define SUBLANG_FRENCH              0x01
#define SUBLANG_FRENCH_BELGIAN      0x02
#define SUBLANG_FRENCH_CANADIAN     0x03
#define SUBLANG_FRENCH_SWISS        0x04
```

RC.EXE does not directly place these constants in the .RES file. It uses the macro MAKELANGID to turn the parameters into a WORD that corresponds to a language ID.

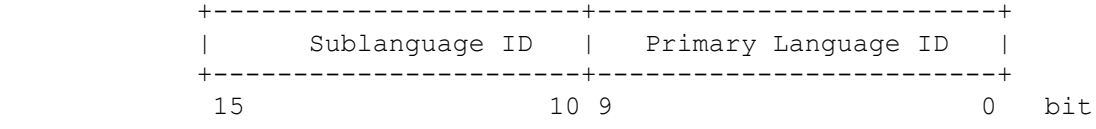
NOTE: The following three combinations have special meaning:

Primary language ID	Sublanguage ID	Meaning
LANG_NEUTRAL	SUBLANG_NEUTRAL	Language neutral
LANG_NEUTRAL	SUBLANG_DEFAULT	User default language
LANG_NEUTRAL	SUBLANG_SYS_DEFAULT	System default language

MORE INFORMATION

The following information is taken from the WINNLS.H file.

A language ID is a 16-bit value that is the combination of a primary language ID and a secondary language ID. The bits are allocated as follows:



Language ID creation/extraction macros:

```
MAKELANGID    - Construct language ID from primary language ID and
                sublanguage ID.
PRIMARYLANGID - Extract primary language ID from a language ID.
SUBLANGID     - Extract sublanguage ID from a language ID.
```

The macros are defined as follows

```
#define MAKELANGID(p, s)       ((((USHORT)(s)) << 10) | (USHORT)(p))
#define PRIMARYLANGID(lgid)    ((USHORT)(lgid) & 0x3ff)
#define SUBLANGID(lgid)        ((USHORT)(lgid) >> 10)
```

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsRc

FormatMessage() Converts GetLastError() Codes

PSS ID Number: Q94999
Authored 28-Jan-1993 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The FormatMessage() application programming interface (API) allows you to convert error codes returned by GetLastError() into error strings, using FORMAT_MESSAGE_FROM_SYSTEM in the dwFlags parameter.

MORE INFORMATION

The following code fragment demonstrates how to get the system message string:

```
LPVOID lpMessageBuffer;

FormatMessage(
  FORMAT_MESSAGE_ALLOCATE_BUFFER |
  FORMAT_MESSAGE_FROM_SYSTEM,
  NULL,
  GetLastError(),
  MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), //The user default language
  (LPTSTR) &lpMessageBuffer,
  0,
  NULL );

//... now display this string

// Free the buffer allocated by the system

LocalFree( lpMessageBuffer );
```

REFERENCES

For more information on language identifiers, please see the topic MAKELANGID in the Win32 Programmer's Reference.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseMisc

Fractional Point Sizes Not Supported in ChooseFont()

PSS ID Number: Q77843
Authored 28-Oct-1991 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

The ChooseFont() common dialog box library routine does not support fractional point sizes. When a fractional point size is entered, it is rounded to the nearest integral point size.

Rounding point sizes affects certain printers that support fractional font sizes. For example, one particular HP LaserJet font cartridge contains an 8.5-point font. The ChooseFont() dialog box displays this font as an 8-point font.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCmnDlg

Freeing Memory for Transactions in a DDEML Client App

PSS ID Number: Q83912
Authored 23-Apr-1992 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

A Dynamic Data Exchange Management Library (DDEML) client application can request data from a server both synchronously and asynchronously by calling the DdeClientTransaction function.

To make a synchronous request, the client application specifies XTYP_REQUEST as the value for the uType parameter to DdeClientTransaction, and any reasonable value for the uTimeout parameter.

To make an asynchronous request, the client application specifies XTYP_REQUEST as the value for the uType parameter to DdeClientTransaction, and TIMEOUT_ASYNC as the value for the uTimeout parameter.

The client can also establish an advise loop with a server application by specifying XTYP_ADVSTART as the value for the uType parameter. In an advise loop, the client application's callback function receives an XTYP_ADVDATA transaction each time the specified data item changes in the server application. (NOTE: This article discusses only hot advise loops in which changed data is communicated to the application. No data is transferred for a warm advise loop, only a notification that the data changed.)

The client application must free the data handle it receives from a synchronous transaction; however, the client application should not free the data handle it receives from an asynchronous transaction or from an advise loop.

MORE INFORMATION

If the client application initiates a synchronous transaction, the DdeClientTransaction function returns a handle to the requested data. If the client application initiates an asynchronous transaction, the DdeClientTransaction function returns either TRUE or FALSE. When the data becomes available, the DDEML sends the client application an XTYP_XACT_COMPLETE notification accompanied by a handle to the requested data. In an active advise loop, the DDEML sends the client application an XTYP_ADVDATA notification accompanied by a handle to the updated data.

In the synchronous case, the client application must call DdeFreeDataHandle before it terminates to free a data handle (and the associated memory) that it received as the return value from DdeClientTransaction. If the DDEML server specified HDATA_APPOWNED when it created the data handle, then the data is invalidated when the client calls DdeFreeDataHandle; the server must call DdeFreeDataHandle before terminating to free the associated memory.

In the asynchronous case, the DDEML sends the client application's callback function an XTYP_XACT_COMPLETE notification when the server has completed the transaction. A handle to the requested data accompanies the notification as the hData parameter to the callback function. This handle is valid until control returns from the client application's callback function. Once the client application's callback function returns control, the DDEML may free the data handle and the client application must not assume that the data handle received in the callback function remains valid. This fact has two implications, as follows:

- The client application cannot call DdeFreeDataHandle on the data handle it receives with an XTYP_XACT_COMPLETE transaction. If the client invalidates the data handle by freeing it in the client's callback function, and the DDEML later attempts to free the handle, a Fatal Exit will result.

- The client application must make a local copy of the data it receives with the XTYP_XACT_COMPLETE transaction to use that data after the callback function returns.

In an advise loop, the client application should not free the data handle that it receives as the hData parameter to the callback function. The DDEML frees the data handle when the client application returns from its callback function. If the client calls DdeFreeDataHandle on the data handle, the DDEML will cause a Fatal Exit when it attempts to free the same data handle.

These rules apply to all data handles, whether or not the server application specified the HDATA_APPOWNED flag when it created the handle.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDde

Freeing Memory in a DDEML Server Application

PSS ID Number: Q83413
Authored 12-Apr-1992 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1

SUMMARY

A Dynamic Data Exchange Management Library (DDEML) server application calls the DdeCreateDataHandle function to allocate a block of memory for data it will send to a client application. DdeCreateDataHandle returns a handle to a block of memory that can be passed between applications.

The server application owns every data handle it creates. However, it is not necessary to call DdeFreeDataHandle under every circumstance. This article details the circumstances under which the server application must call DdeFreeDataHandle and when the DDEML will automatically free a data handle.

MORE INFORMATION

If the server application specifies the HDATA_APPOWNED flag in the afCmd parameter to DdeCreateDataHandle, it must explicitly call DdeFreeDataHandle to free the memory handle. Using HDATA_APPOWNED data handles is convenient when data, such as system topic information, is likely to be passed to a client application more than once, because the server calls DdeCreateDataHandle only once, regardless of the number of times the data handle is passed to a client application.

When it closes down, a server application must call DdeFreeDataHandle for each data handle that it has not passed to a client application. When the server creates a handle without specifying HDATA_APPOWNED, and passes the handle to a client application in an asynchronous transaction, the DDEML frees the data handle when the client returns from its callback function. Therefore, the server is not required to free the data handle it passes to a client because the DDEML frees the handle. However, if the data handle is never sent to a client application, the server must call DdeFreeDataHandle to free the handle. It is the client application's responsibility to call DdeFreeDataHandle for any data provided by a server in a synchronous transaction.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDde

Freeing PackDDElParam() Memory

PSS ID Number: Q94149
Authored 28-Dec-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

When posting DDE messages via PostMessage(), an application first calls PackDDElParam() and sends its return value (a pointer cast to LPARAM) as the lParam in PostMessage().

Normally the receiving application is responsible for freeing the structure [via FreeDDElParam()]. However, if the call to PostMessage() fails, the posting application must free the packed data. This is also the method used by 16-bit Windows.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDde

Gaining Access to ACLs

PSS ID Number: Q102098
Authored 28-Jul-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

To gain access to a security access control list (SACL), a process must have the SE_SECURITY_NAME privilege. When requesting access, the calling process must request ACCESS_SYSTEM_SECURITY in the desired access mask.

There is not a privilege that controls read or write access to a discretionary access control list (DACL). Instead, access to read and write an object's DACL is granted by the READ_CONTROL and WRITE_DAC access rights, respectively. These rights must be specifically granted to the user (or group containing the user) for DACL read or write access to be granted. If the owner of an object requests READ_CONTROL or WRITE_DAC, the access will always be granted.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

GDI Objects and Windows 95 Heaps

PSS ID Number: Q125699
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

Under Windows version 3.1, GDI allocates all resources from a single 64K heap. This limit has caused many applications to run out of GDI resources, especially when using objects that can really take up a lot of the heap, like elliptical regions. This caused GDI resources to be dangerously low when executing several applications at once.

Windows 95 has now introduced a combination of a 16-bit heap and an additional 32-bit heap. The 16-bit heap is still limited to 64K but the 32-bit heap can grow as large as available memory.

As in Windows version 3.1, the 16-bit GDI.EXE of Windows 95 continues to have a 16-bit DGROUP segment with a local heap within it, and most logical objects are still stored in this local heap. The data structures that describe brushes, bitmap headers, and pens, for example, stay in the 16-bit heap. All physical objects, like fonts and bitmaps, are now stored in the 32-bit heap. GDI regions have also been moved to the 32-bit heap. Moving these GDI resources to the 32-bit heap takes the pressure off of the 64K 16-bit heap.

Regions can take up a large amount of resources and were the main source of problems with GDI memory in Windows version 3.1. This will not be a limitation in Windows 95 because regions are stored in the 32-bit heap. Applications will be able to use much more complex regions, and regions will be more useful now that they are not limited to a local 64K heap.

Windows 95, like Windows NT, will free all GDI resources owned by a 32-bit process when that process terminates. Windows 95 will also clean up any GDI resources of 16-bit processes marked as a 4.0 application. Because GDI objects were sharable between applications in Windows version 3.1, Windows 95 will not immediately clean up GDI resources for 16-bit applications marked with a version less than 4.0. However, when all 16-bit applications have finished running, all GDI resources allocated by previous 16-bit applications will be cleaned up.

Additional reference words: 4.00 Heaps
KBCategory: kbprg
KBSubcategory: GdiGeneral

General Overview of Win32s

PSS ID Number: Q83520
Authored 14-Apr-1992 Last modified 30-Mar-1995

The information in this article applies to:

- Microsoft Win32s versions 1.1 and 1.2

SUMMARY

The following is intended as a general introduction to Win32s. More information can be found in the "Win32s Programmer's Reference" and by querying for Knowledge Base articles on "Win32s".

MORE INFORMATION

General Overview

The Win32 API consists of the Window 3.1 (Win16) API, with types stretched to 32 bits, plus the addition of APIs which offer new functionality, like threads, security, services, and virtual memory. Applications developed using the Win32 API are called "Win32-based applications".

Win32s is a set of DLLs and a VxD which allow Win32-based applications to run on top of Windows or Windows for Workgroups version 3.1. Win32s supports a subset of the Win32 API, some directly (like memory management) and some through thunks to the 16-bit systems (particularly GDI and User). Win32s contains function stubs for the APIs that are not supported, which return ERROR_NOT_IMPLEMENTED. Win32s also includes 4 new APIs which support the Universal Thunk (UT). For details on which API are supported under Win32s, refer to the individual API entries in the help file "Win32 API Reference." Among the new features gained from Win32 are structured exception handling (SEH), FP emulation, memory-mapped files, named shared memory, and sparse memory.

Win32-based applications running on Windows 3.1 will generally be faster than their Win16 equivalents on Windows 3.1, particularly if they are memory or floating-point intensive. The actual speed improvement varies with each application, because it depends on how often you cross the thunk layer. Each call which uses a thunk is no more than 10 percent slower than a direct call.

Binary Compatibility

Win32s offers binary compatibility for Win32-based applications on Windows 3.1 and Windows NT.

When you call a Win32 API, two options should be allowed:

- Option A: Your code should allow for a successful return from the function call.

- Option B: Your code should allow for an unsuccessful return from the function call.

For example, if the application is running under Windows 3.1 and a call is made to one of the supported APIs, then the call returns successfully and option A is executed. If the call is made while running under Windows NT, the call again returns successfully and option A should be executed. However, if running under Windows 3.1 and a Win32 API function is called that is unsupported, then an error code is returned and option B should be executed.

If, for example, option A were using a CreateThread() call, then option B would be alternative code, which would handle the task using a single-thread solution.

Programming Issues

Win32-based applications cannot use MS-DOS and BIOS interrupts; therefore, the Win32s VxD has Win32 entries for each Interrupt 21 and the BIOS calls.

The Win32s DLLs may thunk to Win16 when a Win32 application makes a call. The 32-bit parameters are copied from the 32-bit stack to a 16-bit stack and the 16-bit entry point is called. The Win32 application has a 128K stack. When switching to the 16-bit side via UT, the same stack is used, and a 16:16 stack pointer is created which points to the top of the stack. The selector base is set so that there is at least an 8K stack for the 16-bit code.

There are other semantic difference between Windows 3.1 and Win32. Windows 3.1 will run applications for Win32 nonpreemptively in a single, shared address space, while Windows NT runs them preemptively in separate address spaces. It is therefore important that you test your Win32-based application on both Windows 3.1 and Windows NT.

If you need to call routines that reside in a 16-bit DLL or Windows from 32-bit code, you can do this using the Win32s Universal Thunk or other client-server techniques. For a description of UT, please see the "Win32s Programmer's Reference" and the sample UTSAMPLE.

DDE, OLE, WM_COPYDATA, the clipboard, metafiles, and bitmaps can be used between 16-bit Windows-based and Win32-based applications on both Windows 3.1 and Windows NT. RPC is not supported from Win32-based applications.

Additional reference words: 1.10 1.20
KBCategory: kbprg
KBSubcategory: W32s

GetClientRect() Coordinates Are Not Inclusive

PSS ID Number: Q43596
Authored 21-Apr-1989 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

The coordinates returned by GetClientRect() are not inclusive. For example, to draw a border around the edge of the client area, draw it from the coordinates (Rectangle.left, Rectangle.top) to (Rectangle.right-1, Rectangle.bottom-1).

Addtional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrPnt

GetCommandLine() Under Win32s

PSS ID Number: Q102762
Authored 10-Aug-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.0, 1.1, and 1.2

Under Win32s, GetCommandLine() includes the full drive/path of the executable, while under Windows NT GetCommandLine() does not include the full path.

When programs are run from the Program Manager or the File Manager on Windows 3.1, they are spawned using the full path. As a result, argv[0] will have the complete path. When a Win32s application is spawned by a 16-bit application, Windows detects that the application is a Win32s application. The full path is passed to Win32s regardless of whether or not WinExec() was invoked with the full path. As a result, 32-bit applications receive the full path.

When a Win32-based application is spawned from another Win32-based application, the 32-bit kernel passes the information as given by the parent process [that is, if a Win32-based application is started via CreateProcess() from another Win32-based application, argv[0] will contain the path that the spawning application passed in].

Additional reference words: 1.00 1.10 1.20
KBCategory: kbprg
KBSubcategory: W32s

GetCurrentTime and GetTickCount Functions Identical

PSS ID Number: Q45702
Authored 13-Jun-1989 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

The GetCurrentTime() and GetTickCount() functions are identical. Each returns the number of milliseconds (+/- 55 milliseconds) since the user started Windows, and the return value of each function is declared to be a DWORD.

The following code demonstrates these two functions:

```
   DWORD dwCurrTime;
   DWORD dwTickCount;
   char szCurrTime[50];

   dwCurrTime = GetCurrentTime ();
   dwTickCount = GetTickCount ();

   sprintf (szCurrTime, "Current time = %lu\nTick count = %lu",
            dwCurrTime, dwTickCount);

   MessageBox (hWnd, szCurrTime, "Times", MB_OK);
```

NOTE: GetCurrentTime() and GetTickCount() return an unsigned double word (32-bit DWORD) which gives a maximum count of 4,294,967,296 ticks. This number will yield a maximum count of 49.71 days which the system keeps track of running time.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrTim

GetDeviceCaps(hDC, RASTERCAPS) Description

PSS ID Number: Q75912
Authored 09-Sep-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

GetDeviceCaps(hDC, RASTERCAPS) returns the raster capabilities bit field in the GDIINFO structure, which indicates the raster capabilities of the device. The RASTERCAPS index of the GetDeviceCaps() function is documented in the "Microsoft Windows Software Development Kit Reference Volume 1" on page 4-168. The flags available include: RC_BANDING, RC_BITBLT, RC_BITMAP64, RC_DI_BITMAP, RC_DIBTODEV, RC_GDI20_OUTPUT, RC_PALETTE, RC_SCALING, RC_STRETCHBLT, and RC_STRETCHDIB. The GDIINFO structure itself is documented in the "Microsoft Windows Device Development Kit Device Driver Adaptation Guide."

An application should use GetDeviceCaps() to query the printer driver for device capabilities. For example, before printing a bitmap larger than 64K, the application should query the driver using GetDeviceCaps() with the index RASTERCAPS and the flag RC_BITMAP64. If the application fails to test for the capability and prints a bitmap larger than 64K, unexpected printer output may occur if the driver does not support bitmaps larger than 64K.

In particular, if the driver does not support a capability, GDI will attempt to simulate it using a more fundamental capability of the driver. However, the resulting GDI simulation is usually slower, is of lower quality, or differs in some way from a device driver implementation of the capability.

Field Name	Capability or Function	GDI Call to Invoke Capability or Function	Differences in Functionality Resulting from Supporting or not Supporting Capability
0	BitBlt		?
1	Requires banding		?
2	Requires scaling		?
3	Supports		?

	>64K bitmaps		
4	Supports ExtTextOut, FastBorder, GetCharWidth	ExtTextOut	GDI will call StrBlt() once for each character to simulate the ExtTextOut() function's ability to position proportionally-spaced characters. This can be very slow. GDI simulates bold text by overstriking one or more times. This fails on laser printers. Laser printer drivers that support ExtTextOut() offset the text before overstriking.
		FastBorder	
		GetCharWidth	Returns 0 if driver does not support GetCharWidth; otherwise, it calls ExtTextOut() or StrBlt() with count = -1 to obtain the width of each individual character.
5	Has state block		
6	Saves bitmaps in shadow memory		
7	RC_DI_BITMAP	Supports Get and Set DIBs and RLEs	If GDI is called upon to copy a RLE bitmap that contains a transparent window (region not defined by the bitmap), the window will be filled by the current background color. The destination for any GDI DIB operation is a monochrome bitmap. Unidrv offers a variety of halftone dithering techniques to simulate a range of intensities on black and white and color printers. GDI does not. Unidrv offers intensity adjustment to darken and lighten halftone output.

			Unidrv does not support RLE DIBs at present.
8	RC_PALETTE	Performs color palette management	
9	RC_DIBTODEV	Supports SetDIBitsToDevice	See comments above for RC_DI_BITMAP
10	RC_BIGFONTS	Supports Windows 3.0 FONTINFO structure format	This flag specifies the format of the FONTINFO structure passed between GDI's SelectObject() call and the driver's RealizeFont() function. If a bit is set, the Windows 3.0 format is used. Otherwise, the Windows 2.0 font file format is used.
11	RC_STRETCHBLT	Supports StretchBlt	
12	RC_FLOODFILL	Supports flood fill	
13	RC_STRETCHDIB	Supports StretchDIBits	See comments above for RC_DI_BITMAP. Additionally, Unidrv has these limits on the degree of stretching and shrinking supported: The X and Y axes may be scaled independently. The maximum scale-up factor is 256. The maximum product of the X and Y scale-down factors is 256. Therefore, if both axes are scaled down equally, the maximum scale-down factor for each axis is 16.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiDc

GetGlyphOutline() Native Buffer Format

PSS ID Number: Q87115
Authored 22-Jul-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The GetGlyphOutline function provides a method for an application to retrieve the lowest-level information about a glyph in the TrueType environment. This article describes the format of the data the GetGlyphOutline function returns.

MORE INFORMATION

A glyph outline is a series of contours that describe the glyph. Each contour is defined by a TTPOLYGONHEADER data structure, which is followed by as many TTPOLYCURVE data structures as are required to describe the contour.

Each position is described by a POINTFX data structure, which represents an absolute position, not a relative position. The starting and ending point for the glyph is given by the pfxStart member of the TTPOLYGONHEADER data structure.

The TTPOLYCURVE data structures fall into two types: a TT_PRIM_LINE record or a TT_PRIM_QSPLINE record. A TT_PRIM_LINE record is a series of points; lines drawn between the points describe the outline of the glyph. A TT_PRIM_QSPLINE record is a series of points defining the quadratic splines (q-splines) required to describe the outline of the character.

In TrueType, a q-spline is defined by three points (A, B, and C), where points A and C are on the curve and point B is off the curve. The equation for each q-spline is as follows (xA represents the x-coordinate of point A, yA represents the y-coordinate of point A, and so on)

```
   x(t) = (xA-2xB+xC)*t^2 + (2xB-2xA)*t + xA
   y(t) = (yA-2yB+yC)*t^2 + (2yB-2yA)*t + yA
```

where t varies from 0.0 to 1.0.

The format of a TT_PRIM_QSPLINE record is as follows:

- Point A on the q-spline is the current position (either pfxStart in

the TTPOLYGONHEADER, the starting point for the TTPOLYCURVE, or the ending point of the previous TTPOLYCURVE).

- Point B is the current point in the record.

- Point C is as follows:

    - If the record has two or more points following point B, point C is the midpoint between point B and the next point in the record.
    - Otherwise, point C is the point following point B.

The following code presents the algorithm used to process a TT_PRIM_QSPLINE record. While this code demonstrates how to extract q-splines from a TT_PRIM_QSPLINE record, it is not appropriate for use in an application.

```
pfxA = pfxStart;                // Starting point for this polygon

for (u = 0; u < cpfx - 1; u++)  // Walk through points in spline
   {
   pfxB = apfx[u];              // B is always the current point
   if (u < cpfx - 2)            // If not on last spline, compute C
      {
      pfxC.x = (pfxB.x + apfx[u+1].x) / 2;  // x midpoint
      pfxC.y = (pfxB.y + apfx[u+1].y) / 2;  // y midpoint
      }
   else                         // Else, next point is C
      pfxC = apfx[u+1];

                                // Draw q-spline
   DrawQSpline(hdc, pfxA, pfxB, pfxC);
   pfxA = pfxC;                 // Update current point
   }
```

The algorithm above manipulates points directly, using floating-point operators. However, points in q-spline records are stored in a FIXED data type. The following code demonstrates how to manipulate FIXED data items:

```
FIXED fx;
long *pl = (long *)&fx;

// Perform all arithmetic on *pl rather than on fx
*pl = *pl / 2;
```

The following function converts a floating-point number into the FIXED representation:

```
FIXED FixedFromDouble(double d)
{
   long l;

   l = (long) (d * 65536L);
   return *(FIXED *)&l;
```

```
}
```

In a production application, rather than writing a DrawQSpline function to draw each q-spline individually, it is more efficient to calculate points on the q-spline and store them in an array of POINT data structures. When the calculations for a glyph are complete, pass the POINT array to the PolyPolygon function to draw and fill the glyph.

The following example presents the data returned by the GetGlyphOutline for the lowercase "j" glyph in the 24-point Arial font of the 8514/a (Small Fonts) video driver:

```
GetGlyphOutline GGO_NATIVE 'j'
  dwrc            = 208      // Total native buffer size in bytes
  gmBlackBoxX, Y  = 6, 29    // Dimensions of black part of glyph
  gmptGlyphOrigin = -1, 23   // Lower-left corner of glyph
  gmCellIncX, Y   = 7, 0     // Vector to next glyph origin

TTPOLYGONHEADER #1           // Contour for dot on "j"
  cb       = 44              // Total size of dot polygon
  dwType   = 24              // TT_POLYGON_TYPE
  pfxStart = 2.000, 20.000   // Start at lower-left corner of dot

  TTPOLYCURVE #1
    wType  = TT_PRIM_LINE
    cpfx   = 3
    pfx[0] = 2.000, 23.000
    pfx[1] = 5.000, 23.000
    pfx[2] = 5.000, 20.000   // Automatically close to pfxStart


TTPOLYGONHEADER #2   // Contour for body of "j"
  cb       = 164     // Total size is 164 bytes
  dwType   = 24      // TT_POLYGON_TYPE
  pfxStart = -1.469, -5.641

  TTPOLYCURVE #1     // Finish flat bottom end of "j"
    wType  = TT_PRIM_LINE
    cpfx   = 1
    pfx[0] = -0.828, -2.813

  TTPOLYCURVE #2    // Make hook in "j" with spline
                    // Point A in spline is end of TTPOLYCURVE #1
    wType  = TT_PRIM_QSPLINE
    cpfx   = 2              // two points in spline -> one curve
    pfx[0] = -0.047, -3.000  // This is point B in spline
    pfx[1] = 0.406, -3.000   // Last point is always point C

  TTPOLYCURVE #3    // Finish hook in "j"
                    // Point A in spline is end of TTPOLYCURVE #2
    wType  = TT_PRIM_QSPLINE
    cpfx   = 3              // Three points -> two splines
    pfx[0] = 1.219, -3.000   // Point B for first spline
                             // Point C is (pfx[0] + pfx[1]) / 2
```

```
    pfx[1] = 2.000, -1.906   // Point B for second spline
    pfx[2] = 2.000, 0.281    // Point C for second spline

  TTPOLYCURVE #4    // Majority of "j" outlined by this polyline
    wType  = TT_PRIM_LINE
    cpfx   = 3
    pfx[0] = 2.000, 17.000
    pfx[1] = 5.000, 17.000
    pfx[2] = 5.000, -0.250

  TTPOLYCURVE #5    // start of bottom of hook
    wType  = TT_PRIM_QSPLINE
    cpfx   = 2               // One spline in this polycurve
    pfx[0] = 5.000, -3.266   // Point B for spline
    pfx[1] = 4.188, -4.453   // Point C for spline

  TTPOLYCURVE #6    // Middle of bottom of hook
    wType  = TT_PRIM_QSPLINE
    cpfx   = 2               // One spline in this polycurve
    pfx[0] = 3.156, -6.000   // B for spline
    pfx[1] = 0.766, -6.000   // C for spline

  TTPOLYCURVE #7    // Finish bottom of hook and glyph
    wType  = TT_PRIM_QSPLINE
    cpfx   = 2               // One spline in this polycurve
    pfx[0] = -0.391, -6.000  // B for spline
    pfx[1] = -1.469, -5.641  // C for spline

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiTt
```

GetInputState Is Faster Than GetMessage or PeekMessage

PSS ID Number: Q35605
Authored 16-Sep-1988 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

This article describes a method to quickly determine whether an application for the Microsoft Windows graphical environment has any keyboard or mouse messages in its queue without calling the GetMessage or PeekMessage functions.

NOTE: In Win32, GetInputState is thread-local only.

MORE INFORMATION

The GetInputState function returns this information more quickly than GetMessage or PeekMessage. GetInputState returns TRUE (nonzero) if either a keyboard or mouse message is in the application's input queue. If the application must distinguish between a mouse and a keyboard message, GetInputState returns the value 2 for a keyboard and the value 1024 for a mouse message.

Because difficulties may arise if the application loses the input focus, use GetInputState only in tight loop conditions where execution speed is critical.

In Win32, message queues are not global as they are in 16-bit Windows. The message queues are local to the thread. When you call GetInputState, you are checking to see if there are mouse or keyboard messages for the calling thread only. If a window created by another thread in the application has the keyboard input waiting, GetInputState will not be able to check for those messages.

Additional reference words: 3.00 3.10 3.50 4.00 95 yield
KBCategory: kbprg
KBSubcategory: UsrMsg

GetLastError() May Differ Between Windows 95 and Windows NT

PSS ID Number: Q127991
Authored 22-Mar-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

The extended error codes returned by the GetLastError() API are not guaranteed to be the same under Windows 95 and Windows NT. This difference applies to extended error codes generated by calls to GDI, Window Management, and System Services APIs.

The set of potential error codes returned by any particular Win32 API depends on many underlying components, including system kernel-mode components and loaded drivers. The extended error codes are not a part of the Win32 specification. Therefore, they can change as operating system and driver code is changed.

It is impossible to get the error codes for each API in synch across operating system and platforms. Windows NT and Windows 95 have different code bases. Third-party drivers return error codes that are mapped to Win32 error codes. In addition, it would be difficult to accurately maintain error code information in the Win32 Programmer's Reference. Therefore, this information is not included in the documentation.

In general, you should not rely on GetLastError() returning the same values under Windows 95 and Windows NT. Sometimes, an API will fail and GetLastError() will return 0 (ERROR_SUCCESS) under Windows 95. This is because some of the APIs do not set error codes under Windows 95.

Additional reference words: 4.00
KBCategory: kbprg
KBSubcategory: BseMisc UsrMisc GdiMisc

GetSystemMetrics(SM_CMOUSEBUTTONS) Fails Under Win32s

PSS ID Number: Q124836
Authored 12-Jan-1995 Last modified 24-Feb-1995

The information in this article applies to:

- Microsoft Win32s versions 1.15, 1.15a, and 1.20

Under Win32s, the Win32 API GetSystemMetrics() is implemented as a direct thunk to Windows version 3.1. Therefore, the call will return whatever Windows version 3.1 would return for a similar call to the Windows API GetSystemMetrics().

GetSystemMetrics() returns zero for all unrecognized parameters. Under Windows, this includes the new flag SM_CMOUSEBUTTONS, which was introduced in the Win32 API. Therefore, avoid using the SM_CMOUSEBUTTONS flag when your Win32-based application is running under Win32s.

Additional reference words: 1.20
KBCategory: kbprg
KBSubcategory: W32s

Getting a Dialog to Use an Icon When Minimized

PSS ID Number: Q114612
Authored 08-May-1994 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

The standard Windows dialog box does not have an icon when it is minimized. A dialog box can be made to use an icon by replacing the standard dialog box class with a private dialog box class.

MORE INFORMATION

The standard dialog box class specifies NULL as the value of the hIcon field of its WNDCLASS structure. So no icon is drawn when the standard dialog box is minimized.

An icon can be specified by getting the dialog to use a private class as follows:

1. Register a private class.

```
WNDCLASS wc;

wc.style = CS_DBLCLKS | CS_SAVEBITS | CS_BYTEALIGNWINDOW;
wc.lpfnWndProc = DefDlgProc;
wc.cbClsExtra = 0;
wc.cbWndExtra = DLGWINDOWEXTRA;
wc.hInstance = hinst;
wc.hIcon = LoadIcon(hinst, "DialogIcon");
wc.hCursor = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = COLOR_WINDOW + 1;
wc.lpszMenuName = NULL;
wc.lpszClassName = "MyDlgClass";
RegisterClass(&wc);
```

   NOTE: The default dialog window procedure, DefDlgProc(), is used as the window procedure of the class. This causes windows of this class to behave as standard dialogs. The cbWndExtra field has to be assigned to DLGWINDOWEXTRA - the dialog box stores its internal state information in these extra window bytes. The icon to be used when the dialog box is minimized is assigned to the hIncon field.

2. Get the dialog box to use the private class.

   Use the CLASS statement in the dialog box template to get the dialog box to use the private class:

```
IDD_MYDIALOG DIALOG 0, 0, 186, 92
CLASS "MyDlgClass"
:
```

3. Create the dialog box using DialogBox() or CreateDialog().

```
DialogBox (hinst,
           MAKEINTRESOURCE (IDD_MYDIALOG),
           NULL,
          (DLGPROC)MyDlgFunc);
```

MyDlgFunc() is the dialog function implemented by the application. When the dialog box is minimized, it will use the icon specified in the private class.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDlgs

Getting and Using a Handle to a Directory

PSS ID Number: Q105306
Authored 17-Oct-1993 Last modified 05-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

CreateDirectory() can be used to open a new directory. An existing directory can be opened by calling CreateFile(). To open an existing directory with CreateFile(), it is necessary to specify the flag FILE_FLAG_BACKUP_SEMANTICS. The following code shows how this can be done:

```
HANDLE hFile;

hFile = CreateFile( "c:\\mstools",
           GENERIC_READ | GENERIC_WRITE,
           FILE_SHARE_READ | FILE_SHARE_WRITE,
           NULL,
           OPEN_EXISTING,
           FILE_FLAG_BACKUP_SEMANTICS,
           NULL
         );
if( hFile == INVALID_HANDLE_VALUE )
   MessageBox( NULL, "CreateFile() failed", NULL, MB_OK );
```

The handle obtained can be used to obtain information about the directory or to set information about the directory. For example:

```
BY_HANDLE_FILE_INFORMATION fiBuf;
FILETIME ftBuf;
SYSTEMTIME stBuf;
char msg[40];

GetFileInformationByHandle( hFile, &fiBuf );
FileTimeToLocalFileTime( &fiBuf.ftLastWriteTime, &ftBuf );
FileTimeToSystemTime( &ftBuf, &stBuf );
wsprintf( msg, "Last write time is %d:%d  %d/%d/%d",
   stBuf.wHour,stBuf.wMinute,stBuf.wMonth,stBuf.wDay,stBuf.wYear );
MessageBox( NULL, msg, NULL, MB_OK );
```

MORE INFORMATION

Opening directories with CreateFile is not supported on Windows 95.

This code does not work on Win32s, because MS-DOS does not support opening a directory. If you are looking for the creation time of a directory, use FindFirstFile(), because it works on all platforms.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseFileio

Getting Floppy Drive Information

PSS ID Number: Q115828
Authored 05-Jun-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

To get the media type(s) supported by a floppy drive, it is necessary to call CreateFile() to get a handle to the drive and then DeviceIoControl() to get the information. However, if there is no floppy disk in the floppy drive, the following message box may appear when CreateFile() is called for drive A (\\.\a:):

   There is no disk in the drive. Please insert a disk into drive A:

When calling CreateFile(), be sure to use 0 for the access mode and FILE_SHARE_READ for the share mode so that the user will not be prompted to insert a floppy disk:

```
CreateFile(
   szFileName,
   0,
   FILE_SHARE_READ,
   NULL,
   OPEN_ALWAYS,
   0,
   NULL
);
```

Another way to avoid the message box prompt is to put

```
SetErrorMode( SEM_FAILCRITICALERRORS );
```

before the call to CreateFile().

MORE INFORMATION

The following sample code is based on the code in the FLOPPY SDK sample, but it simply displays the media type(s) supported by floppy drive A. The code demonstrates one way to retrieve the supported media without requiring you to insert a floppy disk in the drive.

Sample Code

```
#include <windows.h>
#include <stdio.h>
```

```
#include <winioctl.h>

DISK_GEOMETRY SupportedGeometry[20];
DWORD SupportedGeometryCount;

VOID
GetSupportedGeometrys( HANDLE hDisk )
{
   DWORD ReturnedByteCount;

   if( DeviceIoControl(
          hDisk,
          IOCTL_DISK_GET_MEDIA_TYPES,
          NULL,
          0,
          SupportedGeometry,
          sizeof(SupportedGeometry),
          &ReturnedByteCount,
          NULL
      ))
      SupportedGeometryCount = ReturnedByteCount /  sizeof(DISK_GEOMETRY);

   else SupportedGeometryCount = 0;
}

VOID
PrintGeometry( PDISK_GEOMETRY lpGeometry )
{
   LPSTR MediaType;

   switch ( lpGeometry->MediaType ) {
      case F5_1Pt2_512:
         MediaType = "5.25, 1.2MB,  512 bytes/sector";
         break;
      case F3_1Pt44_512:
         MediaType = "3.5,  1.44MB, 512 bytes/sector";
         break;
      case F3_2Pt88_512:
         MediaType = "3.5,  2.88MB, 512 bytes/sector";
         break;
      case F3_20Pt8_512:
         MediaType = "3.5,  20.8MB, 512   bytes/sector";
         break;
      case F3_720_512:
         MediaType = "3.5,  720KB,  512 bytes/sector";
         break;
      case F5_360_512:
         MediaType = "5.25, 360KB,  512 bytes/sector";
         break;
      case F5_320_512:
         MediaType = "5.25, 320KB,  512 bytes/sector";
         break;
      case F5_320_1024:
         MediaType = "5.25, 320KB,  1024 bytes/sector";
         break;
```

```
      case F5_180_512:
         MediaType = "5.25, 180KB,  512 bytes/sector";
         break;
      case F5_160_512:
         MediaType = "5.25, 160KB,  512  bytes/sector";
         break;
      case RemovableMedia:
         MediaType = "Removable media other than floppy";
         break;
      case FixedMedia:
         MediaType = "Fixed hard disk media";
         break;
      default:
         MediaType = "Unknown";
         break;
   }
   printf("    Media Type %s\n", MediaType );
   printf("    Cylinders %d, Tracks/Cylinder %d, Sectors/Track %d\n",
          lpGeometry->Cylinders.LowPart, lpGeometry->TracksPerCylinder,
          lpGeometry->SectorsPerTrack
         );
}

void main( int argc, char *argv[], char *envp[] )
{
   HANDLE hDrive;
   UINT i;

   hDrive = CreateFile(
                  "\\\\.\\a:",
                  0,
                  FILE_SHARE_READ,
                  NULL,
                  OPEN_ALWAYS,
                  0,
                  NULL
                  );
   if ( hDrive == INVALID_HANDLE_VALUE )
   {
      printf( "Open failed: %d\n", GetLastError() );
      ExitProcess(1);
   }

   GetSupportedGeometrys( hDrive );

   printf( "\nDrive A supports the following disk geometries\n" );
   for( i=0; i<SupportedGeometryCount; i++ )
   {
       printf("\n");
       PrintGeometry( &SupportedGeometry[i] );
   }
   printf("\n");
}
```

Additional reference words: 3.10 3.50

KBCategory: kbprg
KBSubcategory: BseMisc

Getting Real Handle to Thread/Process Requires Two Calls

PSS ID Number: Q90470
Authored 15-Oct-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

The API GetCurrentThread() returns a pseudo-handle rather than the real handle to the thread. To get the real handle to the thread, you need to use DuplicateHandle() using the pseudo-handle that is returned from GetCurrentThread(). In addition, to get the real handle to a process, you need to call DuplicateHandle() after calling GetCurrentProcess().

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseProcThrd

Getting Resources from 16-Bit DLLs Under Win32s

PSS ID Number: Q105761
Authored 24-Oct-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.0, 1.1, and 1.2

A Win32-based application running under Win32s can load a 16-bit dynamic-link library (DLL) using LoadLibrary() and free it with FreeLibrary(). This behavior is allowed primarily so that GetProcAddress() can be called for printer driver application programming interfaces (APIs).

Calling FindResource() with the handle that LoadLibrary() returns to the DLL that it just loaded results in an access violation. However, the Win32-based application can use the following APIs with this handle

```
LoadBitmap
LoadCursor
LoadIcon
```

because this results in USER.EXE (16-bit) making calls to KERNEL.EXE.

If you go through a Universal Thunk to get raw resource data from the 16-bit DLL, it is necessary to convert the resource to 32-bit format, because the resource format is different from the 16-bit format. The 32-bit format is described in the Software Development Kit (SDK) file DOC\SDK\FILEFRMT\RESFMT.TXT.

To determine whether a DLL is a 32-bit or 16-bit DLL, check the DLL header. The DWORD at offset 0x3C indicates where to look for the PE signature. Compare the 4 bytes there to 0x00004550 to determine whether this is a Win32 DLL.

Additional reference words: 1.00 1.10 1.20
KBCategory: kbprg
KBSubcategory: W32s

Getting the Filename Given a Window Handle

PSS ID Number: Q119163
Authored 09-Aug-1994 Last modified 21-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

To find the filename of the program that created a given window under Windows, you would use GetWindowLong(hWnd, GWL_HINSTANCE) to find the module handle and then GetModuleFileName() to find the filename. This method cannot be used under Windows NT because instance handles are not global, but are unique to the address space in which the application is running.

If the application that created the window is a Windows-based application, the name returned is "ntvdm". To get the actual filename, you need to spawn a Win16 application that will call GetModuleFileName() and pass the filename back to your application using some form of interprocess communication (IPC).

MORE INFORMATION

To find the filename of an application once you have its window handle, first use GetWindowThreadProcessId() to find the process ID (PID) of the process that created the window. Using the PID, query the registry for the performance data associated with the process. To do this, you have to enumerate all processes in the system, comparing each PID to the PID of the process that you are looking for, until the data for that process is found. (This data includes the name of the process.)

The following sample code demonstrates how to find the filename of the Program Manager, PROGMAN.EXE, after obtaining its window handle:

Sample Code

```
#include <windows.h>
#include <stdio.h>
#include <string.h>

#define Key "SOFTWARE\\Microsoft\\Windows
NT\\CurrentVersion\\Perflib\\009"

void GetIndex( char *, char * );
void DisplayFilename( DWORD );

/*****************************************************************\
```

```
* Function: void main( )                                                *
*                                                                       *
* Purpose : Application entry point                                     *
*                                                                       *

void main( )
{
   HWND                     hWnd;
   DWORD                    dwActiveProcessId;

   // Get window handle of Program Manager's main window.

   hWnd = FindWindow( "Progman", NULL );

   // Get PID of Program Manager.

   GetWindowThreadProcessId( hWnd, &dwActiveProcessId );

   // Display name of Program Manager's executable file.

   printf( "Searching for filename of Program Manager...\n" );
   DisplayFilename( dwActiveProcessId );
}

/*********************************************************************\
* Function: void DisplayFilename( DWORD )                             *
*                                                                     *
* Purpose : Display executable filename of the process whose PID      *
*           is passed in as a parameter.                              *
*                                                                     *
* Comment : The information is retrieved from the performance         *
*           data in the registry.                                     *
*                                                                     *

void DisplayFilename( DWORD dwProcessId )
{
   DWORD                    CurrentProcessId;
   BOOL                     bContinue = TRUE;
   char                     szIndex[256] = "";
   DWORD                    dwBytes = 12000;
   DWORD                    dwProcessIdOffset;
   int                      i;

   PPERF_DATA_BLOCK         pdb;
   PPERF_OBJECT_TYPE        pot;
   PPERF_INSTANCE_DEFINITION pid;
   PPERF_COUNTER_BLOCK      pcb;
   PPERF_COUNTER_DEFINITION  pcd;

   // Get the index for the PROCESS object.
   GetIndex( "Process", szIndex );

   // Get memory for PPERF_DATA_BLOCK.
   pdb = (PPERF_DATA_BLOCK) HeapAlloc( GetProcessHeap(),
                                       HEAP_ZERO_MEMORY,
```

```
                                        dwBytes);

   // Get performance data.
   while( RegQueryValueEx(HKEY_PERFORMANCE_DATA, (LPTSTR)szIndex, NULL,
                          NULL, (LPBYTE)pdb, &dwBytes) ==
ERROR_MORE_DATA )
   {
      // Increase memory.
      dwBytes += 1000;

      // Allocated memory is too small; reallocate new memory.
      pdb = (PPERF_DATA_BLOCK) HeapReAlloc( GetProcessHeap(),
                                            HEAP_ZERO_MEMORY,
                                            (LPVOID)pdb,
                                            dwBytes);
   }

   // Get PERF_OBJECT_TYPE.
   pot = (PPERF_OBJECT_TYPE)((PBYTE)pdb + pdb->HeaderLength);

   // Get the first counter definition.
   pcd = (PPERF_COUNTER_DEFINITION)((PBYTE)pot + pot->HeaderLength);

   // Get index value for ID_PROCESS.
   szIndex[0] = '\0';
   GetIndex( "ID Process", szIndex );

   for( i=0; i< (int)pot->NumCounters; i++ )
   {
      if (pcd->CounterNameTitleIndex == (DWORD)atoi(szIndex))
      {
         dwProcessIdOffset = pcd->CounterOffset;
         break;
      }

      pcd = ((PPERF_COUNTER_DEFINITION)((PBYTE)pcd + pcd->ByteLength));
   }

   // Get the first instance of the object.
   pid = (PPERF_INSTANCE_DEFINITION)((PBYTE)pot + pot-
>DefinitionLength);

   // Get the name of the first process.
   pcb = (PPERF_COUNTER_BLOCK) ((PBYTE)pid + pid->ByteLength );
   CurrentProcessId = *((DWORD *) ((PBYTE)pcb + dwProcessIdOffset));

   // Find the process object for PID passed in, then print its
   // filename.

   for( i = 1; i < pot->NumInstances && bContinue; i++ )
   {
      if( CurrentProcessId == dwProcessId )
      {
         printf( "The filename is %ls.exe.\n",
                 (char *) ((PBYTE)pid + pid->NameOffset) );
```

```
         bContinue = FALSE;
      }
      else
      {
         pid = (PPERF_INSTANCE_DEFINITION) ((PBYTE)pcb + pcb-
>ByteLength);
         pcb = (PPERF_COUNTER_BLOCK) ((PBYTE)pid + pid->ByteLength);
         CurrentProcessId = *((DWORD *)((PBYTE)pcb +
dwProcessIdOffset));
      }
   }
   if( bContinue == TRUE )
      printf( "Not found.\b" );

   // Free the allocated memory.
   if( !HeapFree(GetProcessHeap(), 0, (LPVOID)pdb) )
      printf( "HeapFree failed in main.\n" );

   // Close handle to the key.
   RegCloseKey( HKEY_PERFORMANCE_DATA );
}

/*****************************************************************\
* Function: void GetIndex( char *, char * )                      *
*                                                                *
* Purpose : Get the index for the given counter                  *
*                                                                *
* Comment : The index is returned in the parameter szIndex       *
*                                                                *

void GetIndex( char *pszCounter, char *szIndex )
{
   char*  pszBuffer;
   char*  pszTemp;
   char   szObject[256] = "";
   DWORD  dwBytes;
   HANDLE hKeyIndex;
   int    i = 0;
   int    j = 0;

   // Open the key.
   RegOpenKeyEx( HKEY_LOCAL_MACHINE,
                 Key,
                 0, KEY_READ,
                 &hKeyIndex );

   // Get the size of the counter.
   RegQueryValueEx( hKeyIndex,
                    "Counters",
                    NULL, NULL, NULL,
                    &dwBytes );

   // Allocate memory for the buffer.
   pszBuffer = (char *) HeapAlloc( GetProcessHeap(),
                                   HEAP_ZERO_MEMORY,
```

```
                                    dwBytes );

    // Get the titles and counters.
    RegQueryValueEx( hKeyIndex,
                     "Counters",
                     NULL, NULL,
                     (LPBYTE)pszBuffer,
                     &dwBytes );

    // Find the index value for PROCESS.
    pszTemp = pszBuffer;

    while( i != (int)dwBytes )
    {
       while (*(pszTemp+i) != '\0')
       {
          szIndex[j] = *(pszTemp+i);
          i++;
          j++;
       }
       szIndex[j] = '\0';
       i++;
       j = 0;
       while (*(pszTemp+i) != '\0')
       {
          szObject[j] = *(pszTemp+i);
          i++;
          j++;
       }
       szObject[j] = '\0';
       i++;
       j = 0;
       if( *(pszTemp+i) == '\0' )
          i++;
       if( strcmp(szObject, pszCounter) == 0 )
          break;
    }

    // Deallocate the memory.
    HeapFree( GetProcessHeap(), 0, (LPVOID)pszBuffer );

    // Close the key.
    RegCloseKey( hKeyIndex );
}
```

REFERENCES

For more information on working with the performance data, please see one or all of the following references:

- The "Win32 Programmer's Reference."

- The "Windows NT Resource Kit," volume 3.

- The source code for PView that is included in the Win32 SDK.

- The "Windows/MS-DOS Developer's Journal," April 1994.

Additional reference words: 3.10 3.50 file name
KBCategory: kbprg
KBSubcategory: BseMisc

Getting the MAC Address for an Ethernet Adapter

PSS ID Number: Q118623
Authored 25-Jul-1994 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

To get the Media Access Control (MAC) address for an ethernet adapter programmatically, you can use NetBIOS if your card is bound to NetBIOS. Use the Netbios() NCBASTAT command and provide a "*" as the name in the NCB.ncb_CallName field. This is demonstrated in the sample code below.

With the NetBEUI and IPX transports, the same information can be obtained at a command prompt by using:

```
net config workstation
```

The ID given is the MAC address.

Sample Code

```
#include <windows.h>
#include <wincon.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

typedef struct _ASTAT_
{
   ADAPTER_STATUS adapt;
   NAME_BUFFER    NameBuff [30];
}ASTAT, * PASTAT;

ASTAT Adapter;

void main (void)
{
   NCB Ncb;
   UCHAR uRetCode;
   char NetName[50];

   memset( &Ncb, 0, sizeof(Ncb) );
   Ncb.ncb_command = NCBRESET;
   Ncb.ncb_lana_num = 0;

   uRetCode = Netbios( &Ncb );
   printf( "The NCBRESET return code is: 0x%x \n", uRetCode );

   memset( &Ncb, 0, sizeof (Ncb) );
   Ncb.ncb_command = NCBASTAT;
```

```
      Ncb.ncb_lana_num = 0;

      strcpy( Ncb.ncb_callname,  "*               " );
      Ncb.ncb_buffer = (char *) &Adapter;
      Ncb.ncb_length = sizeof(Adapter);

      uRetCode = Netbios( &Ncb );
      printf( "The NCBASTAT return code is: 0x%x \n", uRetCode );
      if ( uRetCode == 0 )
      {
         printf( "The Ethernet Number is: %02x%02x%02x%02x%02x%02x\n",
                  Adapter.adapt.adapter_address[0],
                  Adapter.adapt.adapter_address[1],
                  Adapter.adapt.adapter_address[2],
                  Adapter.adapt.adapter_address[3],
                  Adapter.adapt.adapter_address[4],
                  Adapter.adapt.adapter_address[5] );
      }
   }
```

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: NtwkNetbios

Getting the Net Time on a Domain

PSS ID Number: Q98722
Authored 12-May-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

When trying to execute

```
net time /domain:egdomain /set
```

you may get a message saying the account is not known or the password is invalid. This can happen if you are logged on using an account whose name is spelled "Administrator", but the account is a different Administrator account than the one on the domain controller. For example, if you are logged on as EGMACHINE\Administrator, and attempt

```
net time /domain:egdomain /set
```

you will get an error message because EGMACHINE\Administrator is not the same account as EGDOMAIN\Administrator.

The solution is to log off EGMACHINE, log back on as EGMACHINE\PowerUsr1, then execute the command. Note that a privilege is needed to set the time on a machine. In the previous example, the account, EGMACHINE\PowerUsr1, was used to remind us that power users have the needed privilege.

MORE INFORMATION

When running Windows NT while logged on to a domain, doing a NET TIME without the /DOMAIN parameter, as mentioned above, probably will not yield the desired results. However, because you are logged on to a domain, you can do

```
net time /domain /set
```

and a domain controller from the domain you are logged on to will be used. In other words, if you are logged on to a domain, the /DOMAIN parameter is necessary, but the actual domain name can optionally be left to default to the domain you're currently participating in. If your machine is joined to the a domain, that domain will be the default domain for NET TIME /DOMAIN.

If you are trying to get the time from EGDOMAIN and have done a prior

```
net use \\egdomain\ipc$ /user:username
```

where username can be either a legitimate user name or domain name\user

name pair, or anything that will use the guest access), then the net time will use the existing connection to the IPC$ share, using the different user name.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

Getting the WinMain() lpCmdLine in Unicode

PSS ID Number: Q90912
Authored 26-Oct-1992 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

The prototype for WinMain() is as follows:

```
int PASCAL WinMain(
       HANDLE hInstance,
       HANDLE hPrevInstance,
       LPSTR lpCmdLine,
       int nCmdShow );
```

The third parameter is an LPSTR, which specifies an ANSI string. WinMain() cannot be defined to accept Unicode input because there is no way for the system to know whether or not the application wants Unicode at the time WinMain() is called; the system knows once the application has registered a window class.

To get the arguments in Unicode, use GetCommandLine().

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: UsrNls

GetWindowRect() Returns TRUE with Desktop Window Handle

PSS ID Number: Q129598
Authored 30-Apr-1995 Last modified 01-May-1995

The information in this article applies to:

- Microsoft Win32s, version 1.2

Under Windows NT and Windows 95, GetWindowRect() returns FALSE (to indicate failure) if the desktop window handle is used. Under Win32s, GetWindowRect() returns TRUE if the desktop window handle is used, however, the RECT structure is not correctly filled in.

On Win32s, GetWindowRect() returns TRUE unconditionally because the 16-bit Windows GetWindowRect() has no return value. Therefore, before calling GetWindowRect() on Win32s, you should first check that the window handle is not the desktop window handle.

Additional reference words: 1.20 GetDesktopWindow HWND_DESKTOP
KBCategory: kbprg
KBSubcategory: W32s

Global Classes in Win32

PSS ID Number: Q80382
Authored 28-Jan-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

Under 16-bit Windows, when an application wants to check whether or not a window class has been previously registered in the system, it typically checks hPrevInstance. Under 32-bit Windows and Windows NT, hPreviousInstance is always FALSE, because a class definition is not available outside the process context of the process that registers it. Thus, code that checks hPreviousInstance will always register the window class.

MORE INFORMATION

Under 32-bit Windows and Windows NT, a style of CS_GLOBALCLASS indicates that the class is available to every DLL in the process, not every application and DLL in the system, as it does in Windows 3.1.

To have a class registered for every process in the system, it is necessary to:

1. Register the class in a DLL.

2. Use a style of CS_GLOBALCLASS.

3. List the DLL in the following registry key.

   ```
   HKEY_LOCAL_MACHINE\S OFTWARE\
      Microsoft\
      Windows NT\
      CurrentVersion\
      Windows\
      AppInit_DLLs
   ```

   This will force the DLL to be loaded into every process in the system, thereby registering the class in each and every process.

For more information, please see "Window Classes in Win32," which is available on the MSDN CD, starting April 1994.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCls

Global Quota for Registry Data

PSS ID Number: Q94993
Authored 28-Jan-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SUMMARY

Window NT includes a "global quota" on the amount of memory that may be allocated in the registry. This prevents a broken or malicious application from effectively crashing the system by filling paged pool with registry data; however, it does NOT prevent such an application from using up all available registry space. Similar to the file systems, the registry does not support per process/user quotas.

MORE INFORMATION

The total amount of memory that may be consumed by registry data (the hives) is limited by the registry size limit (RSL). The RSL works as a "global quota" for registry space, both in the paged pool and on disk.

By default, the RSL is 25 percent of the size of the paged pool (the paged pool is the memory that may be swapped to disk; all user memory is part of the paged pool). Changing the size of the paged pool also affects the size of the RSL. See

```
HKEY_LOCAL_MACHINE\System\
   CurrentControlSet\
   Control\
   SessionManager\
   MemoryManagement\
   PagedPoolSize
```

in the registry.

The RSL may also be manually set. The value in bytes may be specified by setting the value entry for the key

```
HKEY_LOCAL_MACHINE\System\
   CurrentControlSet\
   Control\
   RegistrySizeLimit.
```

This key must have a type of REG_DWORD and a data length of 4 bytes, or it will be ignored.

If the value entry RegistrySizeLimit is less than 4 megabytes (MB), it will be forced up to 4 MB. If it is greater than about 80 percent of the size of

the paged pool, it will be set down to 80 percent of the size of the paged pool. (It is assumed that paged pool is always larger than 5 MB.)

The system must be rebooted for changes in the RSL to take effect.

To ensure that the user can always at least boot and edit the registry if they somehow set the RSL wrong, the quota is not enforced until after the first successful loading of a hive (that is, loading of a user profile.)

Note that the RSL sets a maximum, not an allocation (unlike some other such limits in the system). Setting a large RSL does NOT cause the system to set aside that much space unless it is actually needed by the registry. It also does NOT guarantee that that much space will be available for use in the registry.

Setting the value entry RegistrySizeLimit to 0xffffffff effectively sets RSL to be as large as paged pool allows (80 percent of paged pool size).

In the initial release of Windows NT, the paged pool defaults to 32 MB, so the default RSL is 8 MB (enough to support approximately 5000 users). The paged pool can be set to a maximum of 128 MB, so the RSL can be no larger than about 102 MB, supporting about 80,000 users (surpassing the realistic limitations of other parts of the system).

Additional reference words: 3.10
KBCategory: kbprg
KBSubcategory: BseMisc

Graphics as Jumps in Windows Help Files

PSS ID Number: Q67884
Authored 28-Dec-1990 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

One of the things that reduces the training required to learn Windows is the presence of consistent graphical help. You can create Help files that use graphic images as context jumps or as glossary pop-ups.

MORE INFORMATION

To use a graphic as a context jump, perform the following steps:

1. Create the graphic in your favorite graphics editor. The typical Windows developer has at least two tools available for this purpose:

   a. Windows Paintbrush, included with the retail release of Windows

   b. SDKPaint, provided with the Windows SDK

   A Note to Paintbrush Users

   When an image is saved from Paintbrush, by default, the entire "canvas" is saved to disk, not just the image created. To work around this potential problem, after the image is complete, use the square outline tool (at the upper right of the tool bar) to select the image. If the image is not square, select the smallest bounding rectangle, leaving a border if desired. From the Edit menu, choose Copy To, and fill in the name of the file in which to save the selection.

2. Edit the file that contains the Help text. This is often called the .RTF file.

3. Place the cursor where the graphic should occur and turn on the strikethrough character format feature of the text editor. Word for Windows users should use double underline character formatting instead.

4. Insert one of the following text strings:

   {bmc <filename>}

-or-
{bml <filename>}
-or-
{bmr <filename>}

This text should be struck through (or double underlined).

5. Turn off the strikethrough (or double underline) character format and turn on hidden text character formatting.

6. Type the context string for the jump destination.

7. Turn off hidden text character formatting.

8. Save the .RTF file.

9. Edit the .HPJ file. There must be a [BITMAPS] section in this file, and that section must include the name of the bitmap used above.

When the Help file is built, clicking the graphic with the mouse will cause Help to change to the specified context.

If the graphic has a name or other short text description, we recommend that the text also be coded as a context jump. This way, the user can click on either the graphic or the text to perform the jump. The text also provides a means for users without a mouse to perform the jump.

More information on Windows Help files is in Chapters 15 through 19 of the "Microsoft Windows Software Development Kit Tools" manual. Chapter 17 discusses context jumps, glossary pop-ups, and inserting graphics into the Help file by reference.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsHlp

Graying the Text of a Button or Static Text Control

PSS ID Number: Q39480
Authored 17-Dec-1988 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.1, 3.5, and 3.51

A control is a child window that is responsible for processing keyboard, mouse, focus, and activation messages, among others. A control paints itself and processes text strings. The color of the text in a button or static text control is automatically changed to gray when the control is disabled with the EnableWindow function. However, If an application subclasses a control to process WM_PAINT messages for the control, the application can use the GrayString function to change the text color.

Additional reference words: 3.00 3.10 3.50 4.00 95 grey
KBCategory: kbprg
KBSubcategory: UsrCtl

Handling COMMDLG File Open Network Button Under Win32s

PSS ID Number: Q117825
Authored 10-Jul-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.1, 1.15, and 1.2

When you run a Win32-based application under Win32s on top of Windows 3.1 without a network, the File Open common dialog box still has a Network button.

Under Windows NT, the File Open common dialog box that appears when you call GetOpenFileName() has a Network button only when a network is present. The API (application programming interface) will either use the Network button in the dialog box template if it exists or dynamically add the button if it is not in the template and there is a network present. If there is no network present but the template contains a Network button, the button will be hidden.

The dialog box template that is included with the Windows Software Development Kit (SDK) does not have a Network button because Windows does not include a network. When the 16-bit Windows-based application is executed under Windows for Workgroups or under Windows NT, the Network button is dynamically added, because these operating systems have built-in networking.

The template that is included with the Win32 SDK has a Network button. If the Win32-based application is run under a non-networked Windows 3.1, the Network button is shown; however it is nonfunctional because there is no network. This happens because the COMMDLG.DLL that provides Windows 3.1 with the common dialog boxes does not recognize networks. Therefore, GetOpenFileName() does not remove the Network button from the template if there is no network.

One solution is to leave the button in the template, determine when you are running under Win32s, and include OFN_NONETWORKBUTTON in the OPENFILENAME structure when Win32s is present but there is no network present. Define a hook function that during WM_INITDIALOG checks the Flags field of the OPENFILENAME struct that lParam is pointing to. If OFN_NONETWORKBUTTON is used, call ShowWindow(GetDlgItem(hWnd, psh14), SW_HIDE).

Alternatively, if your application will most likely run on networked Windows 3.1 machines, you can install the COMMDLG.DLL that ships with Windows for Workgroups 3.11 on all machines, because it is a redistributable dynamic-link library (DLL). This DLL checks to see if a network is present and removes the Network button for you if it is not.

Additional reference words: 1.10 1.15 1.20
KBCategory: kbprg
KBSubcategory: W32s

Handling WM_CANCELMODE in a Custom Control

PSS ID Number: Q74548
Authored 23-Jul-1991 Last modified 23-Jun-1995

-
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

-

SUMMARY

In the Microsoft Windows graphical environment, the WM_CANCELMODE message informs a window that it should cancel any internal state. This message is sent to the window with the focus when a dialog box or a message box is displayed, giving the window the opportunity to cancel states such as mouse capture.

When a control has the focus, it receives a WM_CANCELMODE message when the EnableWindow function disables the control or when a dialog box or a message box is displayed. When a control receives this message, it should cancel modes, such as mouse capture, and delete any timers it has created. A control must cancel these modes because an application may use a notification from the control to display a dialog box or a message box.

The DefWindowProc function processes WM_CANCELMODE by calling the ReleaseCapture function, which cancels the mouse capture for whatever window has the capture. The DefWindowProc function does not cancel any other modes.

MORE INFORMATION

For example, consider a miniature scroll bar custom control that, when it receives a mouse click, sets the mouse capture, creates a timer to provide for repeated scrolling, and sends a WM_VSCROLL message to its parent application. The timer is used to send WM_VSCROLL messages periodically to the parent when the mouse button is held down and the mouse is over the control.

If the application displays a dialog box in response to the WM_VSCROLL message, the control receives a WM_CANCELMODE message, at which time it should kill its timer and release the mouse capture. If the WM_CANCELMODE message is simply passed to the DefWindowProc function, only the mouse capture is released; the timer remains active. When the dialog box is closed, the control immediately sends the parent another WM_VSCROLL message, causing it to display the dialog box again.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg

KBSubcategory: UsrCtl

Height and Width Limitations with Windows SDK Font Editor

PSS ID Number: Q69081
Authored 06-Feb-1991 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

The Microsoft Windows Font Editor (FONTEDIT.EXE) does not allow creation of fonts in which the height or width of the font is greater than 64 pixels. This corresponds to a font-file size of approximately 115K.

This is a limitation of the Font Editor, and not of the fonts themselves. There is no limit to the size of the font file in the font format. Font files created in the Windows version 2.0 font format cannot be larger than 64K.

The Font Editor displays a message box containing one of these two errors if the entered Font Size Character Pixel Width or Character Pixel Height is larger than 64:

> Fixed/maximum width must be a number from 1-64
>
> Font height must be a number from 1-64 pixels

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsFnt

Help Fonts Must Use ANSI Character Set

PSS ID Number: Q92422
Authored 05-Nov-1992 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

When creating a help file, it is often necessary to access special characters available in certain fonts. However, when the Windows Help engine tries to display the special character, it may not appear correctly.

MORE INFORMATION

Fonts used to create RTF source files for the Windows Help engine must use the ANSI character set. Fonts that use character sets other than ANSI may not display properly in the Help engine. The only exception to this is the Symbol font, which is also supported in Windows Help version 3.1.

To determine whether a particular font uses the ANSI character set, run the FONTEST sample included with the Windows Software Development Kit (SDK). From the Font menu, choose the Choose Font option. In the Font dialog box, select the font for which you would like to determine the character set. If the font is an ANSI font, then the tmCharSet value displayed in the main window will be zero.

Additional reference words: 3.10 3.50 4.00 95 HC HCP
KBCategory: kbtool
KBSubcategory: TlsHlp

Help Universal Localization Kit (HULK) Is Available

PSS ID Number: Q129452
Authored 26-Apr-1995 Last modified 29-Apr-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 Software Development Kit (SDK) version 3.5
- Microsoft Win32s version 1.2

Microsoft HULK (Help Universal Localization Kit) is a collection of editing, testing, and building tools for Microsoft Windows-based Help, integrated into an easy-to-use environment. It is designed to make localizing Windows Help projects as smooth as possible, especially if this work is being done by external vendors.

HULK is available on MSDN (Microsoft Developer Network) Development Library, under "Unsupported Tools and Utilities." It is provided "as is" and is not supported by Microsoft.

Additional reference words: kbinf hc.exe hcp.exe
KBCategory: kbtool
KBSubcategory: wintldev

Hooking Console Applications and the Desktop

PSS ID Number: Q108232
Authored 07-Dec-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

Under Windows NT, system hooks are limited in two situations: hooking console windows and hooking the desktop.

Because of the current design of the console and the fact that its user interface runs in the Win32 server, Windows NT does not support hook calls in the context of console applications. Thus, if application A sets a system-wide input hook and text is typed in a console window, application A's input hook will not be called. The same is true for every type of Windows hook except for journal record and journal playback hooks.

Hooking a console application will be enabled in Windows NT 3.51.

MORE INFORMATION

Windows NT supports journaling by forcing the console input thread to communicate with the application that set the hook. In the case of a console, the call to the hook functions are run in the context of the application that installed the hook. This forces Windows NT to synchronously talk to this process in order for it to work; if this process is busy or blocked (as it is when it is sitting at a breakpoint), the console thread is hung.

If console applications were typical Win32-based applications, then this would not be a problem. A design change such as this would require that each console take an extra thread just to process input. This was not acceptable to the designers, and therefore console applications are not implemented in the same way that other Win32-based applications are implemented.

Similarly, if Windows NT allowed other hooks to freely hook any process, then these processes could enter a hanging state as well. The reason that journaling is allowed to hook consoles is that journaling already requires synchronization between all processes in the system, and a mechanism to disengage the journaling process (via the CTRL+ESC, ALT+ESC and CTRL+ALT+DEL keystroke combinations) is provided to prevent hanging the system message queue.

For similar reasons, 16-bit Windows-based applications cannot hook Win32-based applications under Windows NT.

The issues above apply equally well to hooking the desktop. If an application were allowed to hook the desktop, it could potentially hang it. This is completely unacceptable and violates one of the design principles of Windows NT: no application should be allowed to bring down the system or hang the user interface.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrHks

Host Name May Map to Multiple IP Addresses

PSS ID Number: Q110703
Authored 27-Jan-1994 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

The sockets function gethostname() returns a string that identifies the name of the local host. Each host has only one "official" name, regardless of how many IP addresses it has, but there may be several "aliases" for the host.

In TCP/IP, there is not a one-to-one mapping between host name and IP address. The mapping is one-to-many: one host name can have multiple IP addresses.

The sockets function getsockname() returns the sockaddr that the socket is bound to.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: NtwkWinsock

Hot Versus Warm Links in a DDEML Server Application

PSS ID Number: Q108927
Authored 20-Dec-1993 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

In message-based dynamic-data exchange (DDE), a server application can readily distinguish between a hot and a warm link as soon as it receives a request for an advise loop, via the WM_DDE_ADVISE message. This allows the server to send the appropriate value in the data handle (for example, a NULL or a valid data handle) to the client application whenever data changes. In DDEML, there is no way to distinguish between these two links when the server receives a request for an advise transaction.

MORE INFORMATION

Two applications engaged in a DDE conversation may establish one or more links (or advise loops) so that the server application sends periodic updates about the linked item/s to the client, typically when that particular data item's value changes.

In a hot advise loop, the server immediately sends a data handle to the changed data item value. In a warm advise loop, however, the server just notifies the client that the data item value has changed, but does not send the data handle until the client explicitly requests it.

In message-based DDE, a client requests the server for an update on an item by posting the WM_DDE_ADVISE message to the server application. Upon receipt of this message, the server application is able to distinguish between a request for a hot advise loop and a warm advise loop via the fDeferUpd bit of the DDEADVISE structure it received in the low-order word of lParam.

A nonzero fDeferUpd value tells the server that it is a WARM advise loop. This instructs the server to send a WM_DDE_DATA message with a NULL data handle whenever the data item changes, and wait for the client to post a WM_DDE_REQUEST before it sends the handle to the updated data.

A zero fDeferUpd value, however, indicates a HOT advise loop, which then tells the server to send a WM_DDE_DATA message with the valid data handle to the changed data item.

In DDEML, a client requests the server for a hot advise loop via the XTYP_ADVSTART transaction type in a call to the DdeClientTransaction() function. To request a warm advise loop, the client specifies an XTYP_ADVSTART transaction or'ed with the XTYPF_NODATA flag. In both cases,

the DDEML passes the same XTYP_ADVSTART to the server callback function, with no particular flags set, leaving the server with no means to distinguish between a hot or warm advise request.

Note that DDEML internally remembers the type of advise loop established.

Once an advise loop is established, the server application calls the DdePostAdvise() function whenever the value of the data item changes. In a hot advise loop, this causes the DDEML to send the server an XTYP_ADVREQ transaction to its callback function, where the server then returns a data handle to the changed data item. The DDEML then sends the XTYP_ADVDATA transaction to the client's callback function with the data handle.

In a warm advise loop, an XTYP_ADVREQ transaction is not sent to the server's callback function when the data item changes on a call to DdePostAdvise(). DDEML takes care of sending the XTYP_ADVDATA transaction directly to the client's callback function, with the data handle set to NULL. The server application does not send the handle to the changed data item until the client issues an XTYP_REQUEST transaction to obtain this data handle.

Because the type of advise loop (hot versus warm) is not known to the server application, a good rule of thumb in writing server applications that support advise loops is to return a data handle in response to both the XTYP_ADVREQ and the XTYP_REQUEST transactions. This guarantees that a data handle is returned for both hot and warm advise loops.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDde

How CREATOR_OWNER and CREATOR_GROUP Affect Security

PSS ID Number: Q126629
Authored 27-Feb-1995 Last modified 07-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 3.5

SUMMARY

This article discusses the CREATOR_OWNER and CREATOR_GROUP security identifiers (SID) and how they affect security.

MORE INFORMATION

When logged on, each user is represented by a token object. This token contains all the SIDs comprising your security context. Tokens identify one of those SIDs as a default owner for any new objects the user creates, such as files, processes, events, and so forth. Typically, this is the user's account (<domain>\<username>). For an administrator however, the default owner is set to be the local group "Administrators," rather than the individual's user account.

Each token also identifies a primary group for the user. This group does not necessarily have to be one the user is a member of (although it is by default) and it does not determine the objects a user has access to (that is, it isn't used in access validation decisions). However, by default it is assigned as the primary group of any objects the user creates. For the most part, the primary group is required simply for POSIX compatibility, but the primary group does play a role in object creation.

When a new object is created, the security system has the task of assigning protection to the new object. The system follows this process:

1. Assign the new object any protection explicitly passed in by the object creator.

2. Otherwise, assign the new object any inheritable protection from the container the object is created in.

3. Otherwise, assign the new object any protection explicitly passed in by the object creator, but marked as "default."

4. Otherwise, if the caller's token has a default DACL, that will be assigned to the new object.

5. Otherwise, no protection is assigned to the new object.

In step 2, if the parent container has inheritable access-control entries (ACE), those are used to generate protection for the new object. In this case, each ACE is evaluated to see if it should be copied to the new

object's protection. Usually, when an ACE is copied, the SID within that ACE is copied as is. The two exceptions to this rule are when CREATOR_OWNER and CREATOR_GROUP are encountered. In this case, the SID is replaced with the caller's default owner SID or primary group SID.

By default, users logging on to Windows NT are given a primary group of "Domain Users" (when logging on to a Windows NT Server) or the group called "None" (when logging onto a Windows NT Workstation system). Therefore, when you create an object in a container that has an inheritable ACE with the CREATOR_GROUP SID, you will likely end up with an ACE granting Domain Users some access. This may not be what you intended.

Additional reference words: 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

How Database WinSock APIs Are Implemented in Windows NT 3.5

PSS ID Number: Q130024
Authored 09-May-1995 Last modified 16-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.5 and 3.51

SUMMARY

This article describes the ways in which various WinSock database APIs are implemented through the Windows NT versions 3.5 and 3.51 implementation of the WinSock DLL. The article covers the following WinSock database APIs: gethostbyname(), gethostbyaddr(), getprotobyname(), getprotobynumber(), getservbyname(), and getservebynumber().

MORE INFORMATION

Following are the steps taken by each API. In a case where more than one step may be taken to resolve the requested information, the process is not carried to the next step if the information is resolved in the current step.

gethostbyname():

1. Check the HOSTS file at %SystemRoot%\System32\DRIVERS\ETC.
2. Do a DNS query if the DNS server is configured for name resolution.
3. Query one or more WINS servers.

gethostbyaddr():

1. Check the HOSTENT cache.
2. Check the HOSTS file at %SystemRoot%\System32\DRIVERS\ETC.
3. Do a DNS query if the DNS server is configured for name resolution.
4. Do an additional NetBIOS remote adapter status to an IP address for its NetBIOS name table. This step is specific only to the Windows NT version 3.51 implementation.

getprotobyname() and getprotobynumber():

1. Check the PROTOCOL file at %SystemRoot%\System32\DRIVERS\ETC.

getservbyname() and getservebynumber():

1. Check the SERVICES files at %SystemRoot%\System32\DRIVERS\ETC.

Additional reference words: 3.50 3.51
KBCategory: kbprg kbnetwork
KBSubcategory: NtwkWinsock

How HEAPSIZE/STACKSIZE Commit > Reserve Affects Execution

PSS ID Number: Q89296
Authored 16-Sep-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The syntax for the module-definition statements HEAPSIZE and STACKSIZE is as follows

```
HEAPSIZE  [reserve] [,commit]
STACKSIZE [reserve] [,commit]
```

The remarks for HEAPSIZE and STACKSIZE on page 62 of the "Tools User's Guide" manual that comes with the Win32 SDK state the following:

> When commit is less than reserve, memory demands are reduced but execution time is slower.

By default, commit is less than reserve.

The reason that execution time is slower (and it is actually only fractionally slower), is that the system sets up guard pages and could have to process guard page faults.

MORE INFORMATION

If the committed memory is less than the reserved memory, the system sets up guard page(s) around the heap or stack. When the heap or stack grows big enough, the guard pages start accessing outside the committed area. This causes a guard page fault, which tells the system to map in another page. The application continues to run as if you had originally had the new page committed.

If the committed memory is greater than the reserve, no guard pages are created and the program faults if it goes outside the committed memory area.

Experimenting with the commit versus reserve numbers may result in a combination that would produce noticeable results, but for most applications, this difference is probably not noticeable. The potential benefits do not warrant significant experimentation.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseMisc

How Keyboard Data Gets Translated

PSS ID Number: Q104316
Authored 13-Sep-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

Keyboard input is acquired by the keyboard driver, which in turn produces a scan code. This scan code is passed on to the locale-specific Win32 subsystem keyboard driver. This locale-specific driver then converts the scan code to a virtual key and a Unicode character. The Win32 subsystem then passes on this information to the application.

All messages in the Win32 application programming interface (API) that present textual information to a window procedure depend upon how the window registered its class. For example, if RegisterClassW() was called, then Unicode is presented; if RegisterClassA() was called, then ANSI is presented. The conversion of the text is handled by the Window Manager. This allows an ANSI application to send textual information to a Unicode application.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrInp

How POSIX Applications are Recognized

PSS ID Number: Q101770
Authored 21-Jul-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

The presence of the IMAGE_SUBSYSTEM_POSIX_CUI flag in the Subsystem field of the image's optional header means that the application is a 32-bit POSIX application.

Additional reference words: 3.10
KBCategory: kbprg
KBSubCategory: SubSys

How to Add an Access-Allowed ACE to a File

PSS ID Number: Q102102
Authored 28-Jul-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

This article explains the process of adding an access-allowed (or access-denied) access control entry (ACE) to a file.

Adding an access-allowed ACE to a file's access control list (ACL) provides a means of granting or denying (using an access-denied ACE) access to the file to a particular user or group. In most cases, the file's ACL will not have enough free space to add an additional ACE, and therefore it is usually necessary to create a new ACL and copy the file's existing ACEs over to it. Once the ACEs are copied over and the access-allowed ACE is also added, the new ACL can be applied to the file's security descriptor (SD). This process is explained in detail in the section below. Sample code is provided at the end of this article.

MORE INFORMATION

At the end of this article is sample code that defines a function named AddAccessRights(), which adds an access-allowed ACE to the specified file allowing the specified access. Steps 1-17 in the comments of the sample code are discussed in detail below:

1. GetUserName() is called to retrieve the name of the currently logged in user. The user name is stored in plszUserName[] array.

2. LookupAccountName() is called to obtain the SID of the user returned by GetUserName() in step 1. The resulting SID is stored in the UserSID variable and will be used later in the AddAccessAllowedACE() application programming interface (API) call. The LookupAccountName() API is also providing the user's domain in the plszDomain[] array. Please note that LookupAccountName() returns the SID of the first user or group that matches the name in plszUserName.

3. GetFileSecurity() is used here to obtain a copy of the file's security descriptor (SD). The file's SD is placed into the ucSDbuf variable, which is declared a size of 65536+SECURITY_DESCRIPTOR_MIN_LENGTH for simplicity. This value represents the maximum size of an SD, which ensures the SD will be of sufficient size.

4. Here we initialize the new security descriptor (NewSD variable) by

calling the InitializeSecurityDescriptor() API. Because the SetFileSecurity() API requires that the security item being set is contained in a SD, we create and initialize NewSD.

5. Here GetSecurityDescriptorDacl() retrieves a pointer to the discretionary access control list (DACL) in the SD. The pointer is stored in the pACL variable.

6. GetAclInformation() is called here to obtain size information on the file's DACL in the form of a ACL_SIZE_INFORMATION structure. This information is used when computing the size of the new DACL and when copying ACEs.

7. This statement computes the exact number of bytes to allocate for the new DACL. The AclBytesInUse member represents the number of bytes being used in the file's DACL. We add this number to the size of an ACCESS_ALLOWED_ACE and the size of the user's SID. Subtracting the size of a DWORD is an adjustment required to obtain the exact number of bytes necessary.

8. Here we allocate memory for the new ACL that will ultimately contain the file's existing ACEs plus the access-allowed ACE.

9. In addition to allocating the memory, it is important to initialize the ACL structure as we do here.

10. Here we check the bDaclPresent flag returned by GetSecurityDescriptorDacl() to see if a DACL was present in the file's SD. If a DACL was not present, then we skip the code that copies the file's ACEs to the new DACL.

11. After verifying that there is at least one ACE in the file's DACL (by checking the AceCount member), we begin the loop to copy the individual ACEs to the new DACL.

12. Here we get a pointer to an ACE in the file's DACL by using the GetAce() API.

13. Now we add the ACE to the new DACL. It is important to note that we pass MAXDWORD for the dwStartingAceIndex parameter of AddAce() to ensure the ACE is added to the end of the DACL. The statement ((PACE_HEADER)pTempAce)->AceSize provides the size of the ACE.

14. Now that we have copied all the file's original ACEs over to our new DACL, we add the access-allowed ACE. The dwAccessMask variable will contain the access mask being granted. GENERIC_READ is an example of an access mask.

15. Because the SetFileSecurity() API can set a variety of security information, it takes a pointer to a security descriptor. For this reason, it is necessary to attach our new DACL to a temporary SD. This is done by using the SetSecurityDescriptorDacl() API.

16. Now that we have a SD containing the new DACL for the file, we set the DACL to the file's SD by calling SetFileSecurity(). The

DACL_SECURITY_INFORMATION parameter indicates that we want the DACL in the provided SD applied to the file's SD. Please note that only the file's DACL is set, the other security information in the file's SD remains unchanged.

17. Here we free the memory that was allocated for the new DACL.

The below sample demonstrates the basic steps required to add an access-allowed ACE to a file's DACL. Please note that this same process can be used to add an access-denied ACE to a file's DACL. Because the access-denied ACE should appear before access-allowed ACEs, it is suggested that the call to AddAccessDeniedAce() precede the code that copies the existing ACEs to the new DACL.

Sample Code

```
#define SD_SIZE (65536 + SECURITY_DESCRIPTOR_MIN_LENGTH)

BOOL AddAccessRights(CHAR *pFileName, DWORD dwAcessMask)
{
   // SID variables

   UCHAR          psnuType[2048];
   UCHAR          lpszDomain[2048];
   DWORD          dwDomainLength = 250;
   UCHAR          UserSID[1024];
   DWORD          dwSIDBufSize=1024;

   // User name variables

   UCHAR          lpszUserName[250];
   DWORD          dwUserNameLength = 250;

   // File SD variables

   UCHAR          ucSDbuf[SD_SIZE];
   PSECURITY_DESCRIPTOR pFileSD=(PSECURITY_DESCRIPTOR)ucSDbuf;
   DWORD          dwSDLengthNeeded;

   // ACL variables

   PACL           pACL;
   BOOL           bDaclPresent;
   BOOL           bDaclDefaulted;
   ACL_SIZE_INFORMATION AclInfo;

   // New ACL variables

   PACL           pNewACL;
   DWORD          dwNewACLSize;

   // New SD variables

   UCHAR               NewSD[SECURITY_DESCRIPTOR_MIN_LENGTH];
```

```
PSECURITY_DESCRIPTOR psdNewSD=(PSECURITY_DESCRIPTOR)NewSD;

// Temporary ACE

PVOID          pTempAce;
UINT           CurrentAceIndex;

// STEP 1: Get the logged on user name

if(!GetUserName(lpszUserName,&dwUserNameLength))
{
   printf("Error %d:GetUserName\n",GetLastError());
   return(FALSE);
}

// STEP 2: Get SID for current user

if (!LookupAccountName((LPSTR) NULL,
    lpszUserName,
    UserSID,
    &dwSIDBufSize,
    lpszDomain,
    &dwDomainLength,
    (PSID_NAME_USE)psnuType))
{
   printf("Error %d:LookupAccountName\n",GetLastError());
   return(FALSE);
}

// STEP 3: Get security descriptor (SD) for file

if(!GetFileSecurity(pFileName,
              (SECURITY_INFORMATION)(DACL_SECURITY_INFORMATION),
              pFileSD,
              SD_SIZE,
              (LPDWORD)&dwSDLengthNeeded))
{
   printf("Error %d:GetFileSecurity\n",GetLastError());
   return(FALSE);
}

// STEP 4: Initialize new SD

if(!InitializeSecurityDescriptor(psdNewSD,SECURITY_DESCRIPTOR_REVISION))
{
   printf("Error %d:InitializeSecurityDescriptor\n",GetLastError());
   return(FALSE);
}

// STEP 5: Get DACL from SD

if (!GetSecurityDescriptorDacl(pFileSD,
                &bDaclPresent,
                &pACL,
                &bDaclDefaulted))
```

```
{
   printf("Error %d:GetSecurityDescriptorDacl\n",GetLastError());
   return(FALSE);
}

// STEP 6: Get file ACL size information

if(!GetAclInformation(pACL,&AclInfo,sizeof(ACL_SIZE_INFORMATION),
   AclSizeInformation))
{
   printf("Error %d:GetAclInformation\n",GetLastError());
   return(FALSE);
}

// STEP 7: Compute size needed for the new ACL

dwNewACLSize = AclInfo.AclBytesInUse +
               sizeof(ACCESS_ALLOWED_ACE) +
               GetLengthSid(UserSID) - sizeof(DWORD);

// STEP 8: Allocate memory for new ACL

pNewACL = (PACL)LocalAlloc(LPTR, dwNewACLSize);

// STEP 9: Initialize the new ACL

if(!InitializeAcl(pNewACL, dwNewACLSize, ACL_REVISION2))
{
   printf("Error %d:InitializeAcl\n",GetLastError());
   LocalFree((HLOCAL) pNewACL);
   return(FALSE);
}

// STEP 10: If DACL is present, copy it to a new DACL

if(bDaclPresent)  // only copy if DACL was present
{
   // STEP 11: Copy the file's ACEs to our new ACL

   if(AclInfo.AceCount)
   {
      for(CurrentAceIndex = 0; CurrentAceIndex < AclInfo.AceCount;
         CurrentAceIndex++)
      {
         // STEP 12: Get an ACE

         if(!GetAce(pACL,CurrentAceIndex,&pTempAce))
         {
           printf("Error %d: GetAce\n",GetLastError());
           LocalFree((HLOCAL) pNewACL);
           return(FALSE);
         }

          // STEP 13: Add the ACE to the new ACL
```

```
            if(!AddAce(pNewACL, ACL_REVISION, MAXDWORD, pTempAce,
               ((PACE_HEADER)pTempAce)->AceSize))
            {
               printf("Error %d:AddAce\n",GetLastError());
               LocalFree((HLOCAL) pNewACL);
               return(FALSE);
            }
         }
      }
   }

   // STEP 14: Add the access-allowed ACE to the new DACL

   if(!AddAccessAllowedAce(pNewACL,ACL_REVISION2,dwAcessMask, &UserSID))
   {
      printf("Error %d:AddAccessAllowedAce",GetLastError());
      LocalFree((HLOCAL) pNewACL);
      return(FALSE);
   }

   // STEP 15: Set our new DACL to the file SD

   if (!SetSecurityDescriptorDacl(psdNewSD,
                   TRUE,
                   pNewACL,
                   FALSE))
   {
      printf("Error %d:SetSecurityDescriptorDacl",GetLastError());
      LocalFree((HLOCAL) pNewACL);
      return(FALSE);
   }

   // STEP 16: Set the SD to the File

   if (!SetFileSecurity(pFileName, DACL_SECURITY_INFORMATION,psdNewSD))
   {
      printf("Error %d:SetFileSecurity\n",GetLastError());
      LocalFree((HLOCAL) pNewACL);
      return(FALSE);
   }

   // STEP 17: Free the memory allocated for the new ACL

   LocalFree((HLOCAL) pNewACL);
   return(TRUE);
}
```

NOTE: Security descriptors have two possible formats: self-relative and absolute. GetFileSecurity() returns an SD in self-relative format, but SetFileSecurity() expects and absolute SD. This is one reason that the code must create a new SD and copy the information, instead of simply modifying the SD from GetFileSecurity() and passing it to SetFileSecurity(). It is possible to call MakeAbsoluteSD() to do the conversion, but there may not be enough room in the current ACL anyway, as mentioned above.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

How to Add an SNMP Extension Agent to the NT Registry

PSS ID Number: Q128729
Authored 06-Apr-1995 Last modified 11-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5

SUMMARY

After developing a new extension agent DLL, you must configure the registry so that the SNMP extension agent is loaded when the SNMP service is started. This article shows you how.

MORE INFORMATION

You can use REGEDT32.EXE to configure the registry, or you can have your SNMP extension agent installation program configure the registry using the Win32 registry APIs.

To configure an SNMP extension agent in the registry, follow these steps:

1. Walk down:

   ```
   HKEY_LOCAL_MACHINE\
      SYSTEM\
      CurrentControlSet\
      Services\
      SNMP\
      Parameter\
      ExtensionAgents
   ```

   You'll notice at least one entry like this:

   ```
   1:REG_SZ:SOFTWARE\Microsoft\LANManagerMIB2Agent\CurrentVersion
   ```

   Add an entry for the new extension agent. For the SNMP Toaster sample in the SDK, the entry is:

   ```
   3:REG_SZ:SOFTWARE\CompanyName\toaster\CurrentVersion
   ```

   This entry provides a pointer to another registery entry (see step 2) that contains the physical path where the extension agent DLL can be found. Note that "CompanyName" and "toaster" strings can be any other meaningful strings that will be used in Step 2.

2. Go to:

   ```
   HKEY_LOCAL_MACHINE\SOFTWARE
   ```

   Create keys that correspond to the new entry in step 1:

      CompanyName\toaster\CurrentVersion

3. Assign the path of the extension agent DLL as the value to the CurrentVersion key in step 2. For the SNMP toaster sample agent DLL, the entry is:

      Pathname:REG_SZ:D:\mstools\samples\snmp\testdll\testdl.dll

4. Note that names and values in the NT registry are case sensitive.

5. Restart the SNMP service from the control panel. The new extension agent DLL will be loaded. Event Viewer in the administrative tools can be used to view errors encountered during the startup process of the SNMP service and extension agents.

REFERENCES

SNMP.TXT in the \BIN directory of the Win32 SDK.

Windows NT Resource Guide, Chapters 10-14.

Additional reference words: 3.50
KBCategory: kbprg
KBSubcategory: NtwkSnmp

How to Add Windows 95 Controls to Visual C++ 2.0 Dialog

PSS ID Number: Q125686
Authored 01-Feb-1995 Last modified 10-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)
- Microsoft Visual C++ version 2.0

SUMMARY

When using Visual C++ version 2.0 on Windows 95, the Windows 95 common controls do not appear by default on the control palette in the Visual C++ Dialog Editor. Many of these controls may be added, however, by adding an entry into the registry.

MORE INFORMATION

To add buttons in the Dialog Editor's control palette for TreeView, ListView, HotKey, Trackbar, Progress, and UpDown controls:

1. Run REGEDIT from the start menu.

2. Select:

   HKEY_CURRENT_USER\Software\Microsoft\Visual C++ 2.0 Dialog Editor

3. Select the New, Binary Value option from the Edit Menu.

4. Rename the new entry "ChicagoControls" without the quotation marks.

5. Select Modify from the edit menu to change the value of ChicagoControls to 01 00 00 00. The editor will add the spaces between each pair of digits.

6. Exit REGEDIT.

7. Restart Visual C++ version 2.0.

Additional reference words: 2.00 4.00
KBCategory: kbprg
KBSubcategory: UsrCtl

How to Assign Privileges to Accounts for API Calls

PSS ID Number: Q131144
Authored 04-Jun-1995 Last modified 07-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 3.51

SUMMARY

Some new security API calls were added to Win32 in Windows NT version 3.51. Two of these new calls, LogonUser() and CreateProcessAsUser(), require that the calling process have certain privileges. If the calling process is a service running in the Local System account, it will already have these privileges. Otherwise, the required privileges can be added to an account by using the "User Rights Policy" dialog box. Run the User Manager and choose User Rights from the Policies menu to see the dialog box.

NOTE: You must select the "Show Advanced User Rights" check box to see the privileges mentioned in this article.

MORE INFORMATION

The Win32 API reference documents the required privileges, but it gives their internal string names instead of the display names. The "User Rights Policy" dialog box displays the privileges using the display names.

The following table shows the display names associated with the internal string names:

Privilege	Display Name
SeTcbPrivilege	Act as part of the operating system
SeAssignPrimary	Replace a process level token
SeIncreaseQuota	Increase quotas

Additional reference words:
KBCategory: kbprg
KBSubcategory: BseSecurity

How to Back Up the Windows NT Registry

PSS ID Number: Q128731
Authored 06-Apr-1995 Last modified 23-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5

SUMMARY

This article shows by example how to back up portions of the Windows NT registry to be restored later.

MORE INFORMATION

This is normally accomplished by enabling the SeBackupPrivilege and calling RegSaveKey. The operation can fail with ERROR_ACCESS_DENIED if the caller does not have access to portions of the key, such as the registry key HKEY_LOCAL_MACHINE\SECURITY.

If you do not have access to the key, but have backup privilege, pass the REG_OPTION_BACKUP_RESTORE flag to RegCreateKeyEx in the dwOptions parameter. This has an effect similar to FILE_FLAG_BACKUP_SEMANTICS with CreateFile, allowing you to open the key for backup. The resultant key handle can be used in a subsequent call to RegSaveKey.

To back up the registry from the root, it is necessary to enumerate the subkeys from the root, opening each subkey with RegCreateKeyEx and saving the subkey with RegSaveKey.

Sample Code

The following sample source code saves the HKEY_LOCAL_MACHINE registry key, with each subkey saved to a filename matching the subkey name.

The folloing function performs the save operation:

```
LONG SaveRegistrySubKey(
    HKEY hKey,              // handle of key to save
    LPTSTR szSubKey,        // pointer to subkey name to save
    LPTSTR szSaveFileName   // pointer to save path/filename
    )

If the function succeeds, the return value is ERROR_SUCCESS.
If the function fails, the return value is an error value.

/* Save HKEY_LOCAL_MACHINE registry key, each subkey saved to a file of
 * name subkey
 *
```

```
 * this allows us to get around security restrictions which prevent
 * the use of RegSaveKey() on the root key
 *
 * the use of REG_OPTION_BACKUP_RESTORE is not documented in the Win32
 * documentation at this time.  The documentation will be changed to
 * reflect this flag.  This flag is contained in the WINNT.H header file.
 *
 * the optional target machine name is specified in argv[1]
 *
 * v1.21
 * Scott Field (sfield) 01-Apr-1995
 */

#define RTN_OK 0
#define RTN_USAGE 1
#define RTN_ERROR 13

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

LONG SaveRegistrySubKey(HKEY hKey, LPTSTR szSubKey, LPTSTR szSaveFileName);
void PERR(LPTSTR szAPI, DWORD dwLastError);

int main(int argc, char *argv[])
{
   TOKEN_PRIVILEGES tp;
   HANDLE hToken;
   LUID luid;
   LONG rc;     // contains error value returned by Regxxx()
   HKEY hKey;   // handle to key we are interested in
   LPTSTR MachineName=NULL; // pointer to machine name
   DWORD dwSubKeyIndex=0;   // index into key
   char szSubKey[_MAX_FNAME]; // this should be dynamic.
                              // _MAX_FNAME is good because this
                              // is what we happen to save the
                              // subkey as
   DWORD dwSubKeyLength=_MAX_FNAME; // length of SubKey buffer

/*
    if (argc != 2) // usage
    {
        fprintf(stderr,"Usage: %s [<MachineName>]\n", argv[0]);
        return RTN_USAGE;
    }
 */

    // set MachineName == argv[1], if appropriate
    if (argc == 2) MachineName=argv[1];

    //
    // enable backup privilege
    //
    if(!OpenProcessToken(GetCurrentProcess(),
                         TOKEN_ADJUST_PRIVILEGES,
```

```
                        &hToken ))
    {
        PERR("OpenProcessToken", GetLastError() );
        return RTN_ERROR;
    }

    if(!LookupPrivilegeValue(MachineName, SE_BACKUP_NAME, &luid))

    {
        PERR("LookupPrivilegeValue", GetLastError() );
        return RTN_ERROR;
    }

    tp.PrivilegeCount           = 1;
    tp.Privileges[0].Luid       = luid;
    tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;

    AdjustTokenPrivileges(hToken, FALSE, &tp, sizeof(TOKEN_PRIVILEGES),
                              NULL, NULL );

    if (GetLastError() != ERROR_SUCCESS)
    {
        PERR("AdjustTokenPrivileges", GetLastError() );
        return RTN_ERROR;
    }

    // only connect if a machine name specified
    if (MachineName != NULL)
    {
        if((rc=RegConnectRegistry(MachineName,
                                  HKEY_LOCAL_MACHINE,
                                  &hKey)) != ERROR_SUCCESS)
        {
            PERR("RegConnectRegistry", rc);
            return RTN_ERROR;
        }
    }
    else hKey=HKEY_LOCAL_MACHINE;

    while((rc=RegEnumKeyEx(
                        hKey,
                        dwSubKeyIndex,
                        szSubKey,
                        &dwSubKeyLength,
                        NULL,
                        NULL,
                        NULL,
                        NULL)
                        ) != ERROR_NO_MORE_ITEMS) { // are we done?

    if(rc == ERROR_SUCCESS)
   {
        LONG lRetVal; // return value from SaveRegistrySubKey

#ifdef DEBUG
```

```
        fprintf(stdout,"Saving %s\n", szSubKey);
#endif

        // save registry subkey szSubKey to filename szSubKey
        if( (lRetVal=SaveRegistrySubKey(hKey, szSubKey, szSubKey)
            ) != ERROR_SUCCESS)
        {
            PERR("SaveRegistrySubKey", lRetVal);
        }

        // increment index into the key
       dwSubKeyIndex++;

       // reset buffer size
        dwSubKeyLength=_MAX_FNAME;

        // Continue the festivities
        continue;
    }
    else
   {
       //
       // note: we need to watch for ERROR_MORE_DATA
       // this indicates we need a bigger szSubKey buffer
       //
        PERR("RegEnumKeyEx", rc);
        return RTN_ERROR;
   }

    } // RegEnumKeyEx

    // close registry key we have been working with
    RegCloseKey(hKey);

    // Revoke all privileges this process holds (including backup)
    AdjustTokenPrivileges( hToken, TRUE, NULL, 0, NULL, NULL);

    // close handle to process token
    CloseHandle(hToken);

    return RTN_OK;
}

LONG SaveRegistrySubKey(
    HKEY hKey,              // handle of key to save
    LPTSTR szSubKey,        // pointer to subkey name to save
    LPTSTR szSaveFileName   // pointer to save path/filename
    )
{
    HKEY hKeyToSave;    // Handle of subkey to save
    LONG rc;            // result code from RegXxx
    DWORD dwDisposition;

    if((rc=RegCreateKeyEx(hKey,
                          szSubKey, // Name of subkey to open
```

```
                            0,
                            NULL,
                            REG_OPTION_BACKUP_RESTORE, // in winnt.h
                            KEY_QUERY_VALUE, // minimal access
                            NULL,
                            &hKeyToSave,
                            &dwDisposition)
                            ) == ERROR_SUCCESS)
    {
        // Save registry subkey.  If the registry is remote, files will
        // be saved on the remote machine
        rc=RegSaveKey(hKeyToSave, szSaveFileName, NULL);

        // close registry key we just tried to save
        RegCloseKey(hKeyToSave);
    }

    // return the last registry result code
    return rc;
}

void PERR(
    LPTSTR szAPI,       // pointer to failed API name
    DWORD dwLastError   // last error value associated with API
    )
{
    LPTSTR MessageBuffer;
    DWORD dwBufferLength;

    //
    // TODO get this fprintf out of here!
    //
    fprintf(stderr,"%s error! (rc=%lu)\n", szAPI, dwLastError);

    if(dwBufferLength=FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
                                    FORMAT_MESSAGE_FROM_SYSTEM,
                                    NULL,
                                    dwLastError,
                                    LANG_NEUTRAL,
                                    (LPTSTR) &MessageBuffer,
                                    0,
                                    NULL))
    {


        DWORD dwBytesWritten;

        //
        // Output message string on stderr
        //
        WriteFile(GetStdHandle(STD_ERROR_HANDLE),
                  MessageBuffer,
                  dwBufferLength,
                  &dwBytesWritten,
                  NULL);
```

```
        //
        // free the buffer allocated by the system
        //
        LocalFree(MessageBuffer);
    }
}
```

Additional reference words: 3.50
KBCategory: kbprg kbcode
KBSubcategory: BseMisc BseSecurity CodeSam

How to Broadcast Messages Using NetMessageBufferSend()

PSS ID Number: Q131458
Authored 12-Jun-1995 Last modified 15-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.5, 3.51

The NetMessageBufferSend() API can be used to broadcast a message. To broadcast a copy a of a particular message to all workstations running messanger service in a particular domain, the LPWSTR msgname parameter needs to be specified as "DOMAINNAME*" - where DOMAINNAME is the name of the domain to which a message is to be sent. In this case, you can use the following piece of code to call this API:

```
#define UNICODE
#define MESGLEN 50
WCHAR awcToName[] = TEXT("DomainName*");
WCHAR awcFromName[] = Text("MyComputer");
WCHAR awcMesgBuffer[MESGLEN] = Text("This ia Test Message");
NET_API_STATUS nasStatus;

nasStatus = NetMessageBufferSend(NULL,
                                 awcToName,
                                 awcFromName,
                                 awcMesgBuffer,
                                 MESGLEN);
```

Additional reference words: 3.50 3.51 LanMan
KBCategory: kbprg kbnetwork
KBSubcategory: NtwkMisc

How to Build a Japanese NT 3.50 Application on US NT 3.50

PSS ID Number: Q126744
Authored 01-Mar-1995 Last modified 24-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 3.5
- Microsoft Visual C++, 32-bit Edition, version 2.0
- Microsoft Windows NT version 3.5

Use the following steps to build a Japanese application on U.S. Windows NT version 3.5 using the English 32-bit Edition of Visual C++ version 2.0:

1. Copy c_932.nls from your Japanese Windows NT version 3.5 CD or disks.

2. Add c_932.nls to your English Windows NT version 3.5 system directory (for example, WINNT35\SYSTEM32.

3. Using REGEDT32.EXE, search for "codepage" in the HKEY_LOCAL_MACHINE registry and add this value:

   932 : REG_SZ : c_932.nls

4. Add the "/C932" switch to the RC compiler option in the Project setting in the Microsoft Visual C++ IDE (Intergrated Development Environment).

Take similar steps if you want to use U.S. Windows NT version 3.5 as the development environment and build an application that requires a codepage that is not present on U.S. Windows NT version 3.5.

Additional reference words: 3.50 msvcj setlocale language
KBCategory: kbprg
KBSubcategory: WIntlDev

How to Calculate Dialog Base Units with Non-System-Based Font

PSS ID Number: Q125681
Authored 01-Feb-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

This article shows how to calculate the dialog base unit in Windows for the dialog box using a font other than System Font. You can use this calculation to build dialog templates in memory or calculate dialog dimensions.

MORE INFORMATION

Each dialog box template contains measurements that specify the position, width, and height of the dialog box and the controls it contains. These measurements are device independent, so an application can use a single template to create the same dialog box for all types of display devices. This ensures that a dialog box will have the same proportions and appearance on all screens despite differing resolutions and aspect ratios between screens.

Further, dialog box measurements are given in dialog base units. One horizontal base unit is equal to one-fourth of the average character width for the system font. One vertical base unit is equal to one-eighth of the average character height for the system font. An application can retrieve the number of pixels per base unit for the current display by using the GetDialogBaseUnits function. The low-order word of the return value, from the GetDialogBaseUnits function, contains the horizontal base units and the high-order word of the return value, from the GetDialogBaseUnits function, contains the vertical base units.

Using this information, you can compute the dialog base units for a dialog using font other than system font:

```
horz pixels == (horz dialog units * average char width  of font) / 4
vert pixels == (vert dialog units * average char height of font) / 8
```

As the font of a dialog changes, the actual size and position of a control also changes.

- Four pixels is half the average character width of the system font.
- Eight pixels is half the average character height of the system font.

Here's another version of the same formulas:

```
horz pixels == 2 * horz dialog units * (average char width  of dialog font
                       / average char width of system font)
```

```
vert pixels == 2 * vert dialog units * (average char height of dialog font
                        / average char height of system font)

One dialog base unit is equivalent to the number of pixels per dialog unit
which gives:

1 horz dialog base unit == (2 * average char width  dialog font /
                              average char width  system font) pixels
1 vert dialog base unit == (2 * average char height dialog font /
                              average char height system font) pixels

Average character width and height of a font can be computed as follows:

hFontOld = SelectObject(hdc,hFont);
GetTextMetrics(hdc,&tm);
GetTextExtentPoint32(hdc,"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrst"
           "uvwxyz",52,&size);
avgWidth = (size.cx/26+1)/2;
avgHeight = (WORD)tm.tmHeight;

Additional reference words: 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDlg
```

How to Calculate the Height of Edit Control to Resize It

PSS ID Number: Q124315
Authored 28-Dec-1994 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), version 3.1
- Microsoft Win32 Software Development Kit (SDK), versions 3.5 and 3.51

SUMMARY

When a program changes the font of an edit control, it must calculate the new height of the control so that text is displayed correctly. When an edit control contains a border, the control automatically adds white space around the text so the text won't interfere with the border.

This article shows by example how a program can modify the height of an edit control so that text displayed in the control looks right after the program changes the font.

MORE INFORMATION

The height of the control on creation is calculated as the height of the control's font plus half of the smaller of the height of the control's font or the height of the system font. You can use a function similar to the one below to calculate the new height of an edit control when the font in the control is changed.

Sample Code

```
void ResizeEdit(HWND hwndEdit, HFONT hNewFont)
{
   HFONT       hSysFont,
               hOldFont;
   HDC         hdc;
   TEXTMETRIC  tmNew,
               tmSys;
   RECT        rc;
   int         nTemp;

   //get the DC for the edit control
   hdc = GetDC(hwndEdit);

   //get the metrics for the system font
   hSysFont = GetStockObject(SYSTEM_FONT);
   hOldFont = SelectObject(hdc, hSysFont);
   GetTextMetrics(hdc, &tmSys);

   //get the metrics for the new font
   SelectObject(hdc, hNewFont);
```

```
   GetTextMetrics(hdc, &tmNew);

   //select the original font back into the DC and release the DC
   SelectObject(hdc, hOldFont);
   DeleteObject(hSysFont);
   ReleaseDC(hwndEdit, hdc);

   //calculate the new height for the edit control
   nTemp = tmNew.tmHeight + (min(tmNew.tmHeight, tmSys.tmHeight)/2);

   //re-size the edit control
   GetWindowRect(hwndEdit, &rc);
   MapWindowPoints(HWND_DESKTOP, GetParent(hwndEdit), (LPPOINT)&rc, 2);
   MoveWindow( hwndEdit,
               rc.left,
               rc.top,
               rc.right - rc.left,
               nTemp,
               TRUE);
}
```

Additional reference words: 3.10 3.50 win16sdk
KBCategory: kbprg kbcode
KBSubcategory: UsrCtl

How to Change Hard Error Popup Handling in Windows NT

PSS ID Number: Q128642
Authored 05-Apr-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5
- Microsoft Windows NT version 3.5

SUMMARY

In an unattended environment, you may want to automatically dispatch hard error popups that require user intervention. This article gives you the code you need to change the hard error popup mode.

MORE INFORMATION

Windows NT allows the user to change the handling of hard error popups that result from application and system errors. Such errors include no disk in the drive and general protection (GP) faults.

Normally, these events cause a hard error popup to be displayed, which requires user intervention to dispatch. This behavior can be modified so that such errors are logged to the Windows NT event log. When the error is logged to the event log, no user intervention is necessary, and the system provides a default handler for the hard error. The user can examine the event log to determine the cause of the hard error.

Registry Entry

The following registry entry controls the hard error popup handling in Windows NT:

```
HKEY_LOCAL_MACHINE\
 SYSTEM\
 CurrentControlSet\
 Control\
 Windows\
 ErrorMode
```

Valid Modes

The following are valid values for ErrorMode:

- Mode 0

  This is the default operating mode that serializes the errors and waits for a response.

- Mode 1

  If the error does not come from the system, this is the normal operating mode. If the error comes from the system, this logs the error to the event log and returns OK to the hard error. No intervention is required and the popup is not seen.

- Mode 2

  This always logs the error to the event log and returns OK to the hard error. Popups are not seen.

In all modes, system-originated hard errors are logged to the system log. To run an unattended server, use mode 2.

Sample Code to Change Hard Error Popup Mode

The following function changes the hard error popup mode. If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

```
BOOL SetGlobalErrorMode(
    DWORD dwErrorMode   // specifies new ErrorMode value
    )
{
    HKEY hKey;
    LONG lRetCode;

    // make sure the value passed isn't out-of-bounds
    if (dwErrorMode > 2) return FALSE;

    if(RegOpenKeyEx(HKEY_LOCAL_MACHINE,
                    "SYSTEM\\CurrentControlSet\\Control\\Windows",
                    0,
                    KEY_SET_VALUE,
                    &hKey) != ERROR_SUCCESS) return FALSE;

    lRetCode=RegSetValueEx(hKey,
                           "ErrorMode",
                           0,
                           REG_DWORD,
                           (CONST BYTE *) &dwErrorMode,
                           sizeof(DWORD) );

    RegCloseKey(hKey);

    if (lRetCode != ERROR_SUCCESS) return FALSE;

    return TRUE;
}
```

Sample Code to Obtain Hard Error Popup Mode

The following function obtains the hard error popup mode. If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. If the function succeeds, dwErrorMode contains the error popup mode. Otherwise, dwErrorMode is undefined.

```
BOOL GetGlobalErrorMode(
    LPDWORD dwErrorMode // Pointer to a DWORD to place popup mode
    )
{
    HKEY hKey;
    LONG lRetCode;
    DWORD cbData=sizeof(DWORD);

    if(RegOpenKeyEx(HKEY_LOCAL_MACHINE,
                    "SYSTEM\\CurrentControlSet\\Control\\Windows",
                    0,
                    KEY_QUERY_VALUE,
                    &hKey) != ERROR_SUCCESS) return FALSE;

    lRetCode=RegQueryValueEx(hKey,
                             "ErrorMode",
                             0,
                             NULL,
                             (LPBYTE) dwErrorMode,
                             &cbData );

    RegCloseKey(hKey);

    if (lRetCode != ERROR_SUCCESS) return FALSE;

    return TRUE;
}
```

Additional reference words: 3.50
KBCategory: kbprg kbcode
KBSubcategory: BseErrdebug BseMisc CodeSam

How to Change Small Icon for FileOpen and Other Common

PSS ID Number: Q130758
Authored 25-May-1995 Last modified 22-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SUMMARY

Applications that need to display icons on the caption bars of File Open and other Common Dialogs can do so by installing a hook function and sending the WM_SETICON message from within the hook function to the common dialog to change its small icon. Note that OFN_ENABLEHOOK flag (or relavant flags for other common dialogs) has to be set for your hook function to be called.

MORE INFORMATION

Under Windows 95, every popup or overlapped window can have two icons associated with it, a large icon used when the window is minimized and a small icon used for displaying the system menu icon.

Common Dialogs under Windows 95 do not display a small icon on their caption bars by default. If you want the application to display its own icon for the system menu, have the application install a hook funtion for that common dialog and send the WM_SETICON message when the hook callback function is called with the WM_INITDIALOG message.

The WM_SETICON message is sent to change or set the small and large icons of a window. In this case, because you are setting the small icon, wParam must be set to FALSE.

Code Sample

The following code shows how to do this for a File Open Common Dialog:

```
// Fill in the OPENFILENAME structure to support
// a hook and  a template (optional).

OpenFileName.lStructSize       = sizeof(OPENFILENAME);
OpenFileName.hwndOwner      = hWnd;
OpenFileName.hInstance         = g_hInst;
  ...
  ...
  ...
  ...

OpenFileName.lpfnHook          = ComDlg32HkProc;
```

```
 OpenFileName.lpTemplateName  = NULL;
 OpenFileName.Flags              = OFN_SHOWHELP |
                OFN_EXPLORER | OFN_ENABLE_HOOK;
```

Note that the lpTemplateName parameter is set to NULL. To just install a hook, one does not need a custom template. The hook function will get called if it is sepcified in the structure.

Below is the Comdlg32HkgProc hook callback funtion that chages the small icon. This code below is for the open or save as dialog boxes only.

```
BOOL CALLBACK ComDlg32HkProc(HWND hDlg,
                             UINT uMsg,
                             WPARAM wParam,
                             LPARAM lPar
{

   HWND hWndParent;
   HICON  hIcon;


   switch (uMsg)
   {
      case WM_INITDIALOG:

         hWndParent = GetParent(hDlg);

         hIcon = LoadIcon(g_hInst, "CustomIcon");

         SendMessage(hWndParent,
                     WM_SETICON,
                    (WPARAM)(BOOL)FALSE,
                    (LPARAM)(HICON)hIcon);

        return TRUE;

        break;


      default:
        break;

    }
```

NOTE: This code calls GetParent() to get the actual window handle of the common dialog box. This is done for the FileOpen and SaveAs dialog boxes only. These dialogs, when created with the OFN_EXPLORER look with a hook and a template (optional), create a seperate dialog to hold all the controls. This is the dialog handle that is passed in the hook function. The parent of this dialog is the main common dialog window, whose caption icon must be modified. The FileOpen and SaveAs dialog boxes with the old style (no OFN_EXPLORER) need not call GetParent().

All other common dialogs, such as ChooseColor and ChooseFont, behave as the the Windows version 3.1 common dialogs behaved, so the code listed in this

article does not need to call GetParent(). It can just send the WM_SETICON message to the hDlg that is passed to the hook function.

Additional reference words: 4.00 user common dialog
KBCategory: kbprg kbcode
KBSubcategory: UsrCmnDlg

How to Change the International Settings Programmatically

PSS ID Number: Q126625
Authored 27-Feb-1995 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.5, 3.51, 4.0

The International control panel icon allows you to choose international settings for the time, date, keyboard, and so forth. You can change these settings from your application code by calling SetLocaleInfo(). Be sure to broadcast a WM_WININICHANGE message using PostMessage() so that all applications are notified of the change.

The Win32 API allows you to display the time and date in the correct international format. Use GetDateFormat() and GetTimeFormat() to get the date and time in the format specified by the user.

Additional reference words: 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrLoc WintlDev

How to Compile Large Chinese or Korean Help Files

PSS ID Number: Q123331
Authored 29-Nov-1994 Last modified 10-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK version 3.5

The Extended Help Compiler is not available in Traditional/Simplified Chinese Windows version 3.1 SDK extension or in the Korean Windows version 3.1 SDK extension. Therefore, large help files in Chinese or Korean cannot be compiled under Windows version 3.1 using the extended help compiler.

However, large Chinese or Korean help files can be compiled under Windows NT, by using Chinese or Korean Help Compiler (HC31.EXE). Make sure the directory containing the appropriate HC31.EXE is in the path.

If the .HPJ file contains Compress=YES, change it to compress=MEDIUM or to compress=FALSE.

The resulting .HLP files cannot be displayed correctly under Windows NT. If you try it, you'll see random symbols in place of the Chinese or Korean characters. To examine the result of the compilation, exit from Windows NT, and start Chinese or Korean Windows. Then display the help files.

Additional reference words: fesdk hcp
KBCategory: kbprg kbtool
KBSubcategory: WIntlDev

How to Convert a Binary SID to Textual Form

PSS ID Number: Q131320
Authored 07-Jun-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT

SUMMARY

It may be useful to convert a binary SID (security identifier) to a readable, textual form, for display or manipulation purposes.

One example of an application that makes use of SIDs in textual form is the Windows NT Event Viewer. If the Event Viewer cannot look up the name associated with the SID of a logged event, the Event Viewer displays a textual representation of the SID.

Windows NT also makes use of textual SIDs when loading user configuration hives into the HKEY_USERS registry key.

Applications that obtain domain and user names can display the textual SID representation when the Win32 API LookupAccountSid fails to obtain domain and user information. Such a failure can occur if the network is down, or the target machine is unavailable.

Sample Code

The following sample code displays the textual representation of the SID associated with the current user. This source code converts a SID using the same algorithm that the Windows NT operating system components use.

```
/*++

A standardized shorthand notation for SIDs makes it simpler to
visualize their components:

   S-R-I-S-S...

In the notation shown above:

   S identifies the series of digits as an SID.
   R is the revision level.
   I is the identifier-authority value.
   S is subauthority value(s).

A SID could be written in this notation as follows:

   S-1-5-32-544

In this example, the SID has a revision level of 1, an identifier-authority
```

value of 5, first subauthority value of 32, second subauthority value of
544. (Note that the above SID represents the local Administrators group.)

The GetTextualSid function converts a binary SID to a textual string.

The resulting string will take one of two forms. If the IdentifierAuthority
value is not greater than 2^32, then the SID will be in the form:

```
S-1-5-21-2127521184-1604012920-1887927527-19009
  ^ ^ ^^ ^^^^^^^^^^ ^^^^^^^^^^ ^^^^^^^^^^ ^^^^^
  | | |      |          |          |        |
  +-+-+------+----------+----------+--------+--- Decimal
```

Otherwise, the SID will take the form:

```
S-1-0x206C277C6666-21-2127521184-1604012920-1887927527-19009
  ^ ^^^^^^^^^^^^^^ ^^ ^^^^^^^^^^ ^^^^^^^^^^ ^^^^^^^^^^ ^^^^^
  |       |        |      |          |          |        |
  |   Hexadecimal  |      |          |          |        |
  +----------------+------+----------+----------+--------+--- Decimal
```

If the function succeeds, the return value is TRUE. If the function fails,
the return value is FALSE. To get extended error information, call the
Win32 API GetLastError().

--*/

```
#define RTN_OK 0
#define RTN_ERROR 13

#include <windows.h>
#include <stdio.h>

BOOL GetTextualSid(
    PSID pSid,          // binary SID
    LPSTR TextualSID,   // buffer for textual representation of SID
    LPDWORD dwBufferLen // required/provided TextualSid buffersize
    );

int main(void)
{
    #define MY_BUFSIZE 256 // all allocations should be dynamic
    HANDLE hToken;
    TOKEN_USER ptgUser[MY_BUFSIZE];
    DWORD cbBuffer=MY_BUFSIZE;
    char szTextualSid[MY_BUFSIZE];
    DWORD cbSid=MY_BUFSIZE;
    BOOL bSuccess;

    //
    // obtain current process token
    //
    if(!OpenProcessToken(
                GetCurrentProcess(), // target current process
```

```
                TOKEN_QUERY,         // TOKEN_QUERY access
                &hToken              // resultant hToken
                ))
    {
        fprintf(stderr, "OpenProcessToken error! (rc=%lu)\n",
            GetLastError() );
        return RTN_ERROR;
    }

    //
    // obtain user identified by current process's access token
    //
    bSuccess=GetTokenInformation(
                hToken,    // identifies access token
                TokenUser, // TokenUser info type
                ptgUser,   // retrieved info buffer
                cbBuffer,  // size of buffer passed-in
                &cbBuffer  // required buffer size
                );

    // close token handle. Do this even if error above
    CloseHandle(hToken);

    if(!bSuccess)
    {
        fprintf(stderr, "GetTokenInformation error! (rc=%lu)\n",
            GetLastError() );
        return RTN_ERROR;
    }

    //
    // obtain the textual representaion of the SID
    //
    if(!GetTextualSid(
                ptgUser->User.Sid, // user binary SID
                szTextualSid,      // buffer for TextualSid
                &cbSid             // size/required buffer
                ))
    {
        fprintf(stderr, "GetTextualSid error! (rc=%lu)\n",
            GetLastError() );
        return RTN_ERROR;
    }

    // display the TextualSid representation
    fprintf(stdout,"%s\n", szTextualSid);

    return RTN_OK;
}

BOOL GetTextualSid(
    PSID pSid,          // binary SID
    LPSTR TextualSID,   // buffer for textual representaion of SID
    LPDWORD dwBufferLen // required/provided TextualSid buffersize
    )
```

```
{
    PSID_IDENTIFIER_AUTHORITY psia;
    DWORD dwSubAuthorities;
    DWORD dwSidRev=SID_REVISION;
    DWORD dwCounter;
    DWORD dwSidSize;

    //
    // test if SID passed in is valid
    //
    if(!IsValidSid(pSid)) return FALSE;

    // obtain SidIdentifierAuthority
    psia=GetSidIdentifierAuthority(pSid);

    // obtain sidsubauthority count
    dwSubAuthorities=*GetSidSubAuthorityCount(pSid);

    //
    // compute buffer length
    // S-SID_REVISION- + identifierauthority- + subauthorities- + NULL
    //
    dwSidSize = 15 + 12 + (12 * dwSubAuthorities) + 1;

    //
    // check provided buffer length.
    // If not large enough, indicate proper size and setlasterror
    //
    if (*dwBufferLen < dwSidSize)
    {
        *dwBufferLen = dwSidSize;
        SetLastError(ERROR_INSUFFICIENT_BUFFER);
        return FALSE;
    }

    //
    // prepare S-SID_REVISION-
    //
    wsprintf(TextualSID, "S-%lu-", dwSidRev );

    //
    // prepare SidIdentifierAuthority
    //
    if ( (psia->Value[0] != 0) || (psia->Value[1] != 0) )
    {
        wsprintf(TextualSID + lstrlen(TextualSID),
                    "0x%02hx%02hx%02hx%02hx%02hx%02hx",
                    (USHORT)psia->Value[0],
                    (USHORT)psia->Value[1],
                    (USHORT)psia->Value[2],
                    (USHORT)psia->Value[3],
                    (USHORT)psia->Value[4],
                    (USHORT)psia->Value[5]);
    }
    else
```

```
    {
        wsprintf(TextualSID + lstrlen(TextualSID), "%lu",
                    (ULONG)(psia->Value[5]      )   +
                    (ULONG)(psia->Value[4] <<  8)   +
                    (ULONG)(psia->Value[3] << 16)   +
                    (ULONG)(psia->Value[2] << 24)   );
    }

    //
    // loop through SidSubAuthorities
    //
    for (dwCounter=0 ; dwCounter < dwSubAuthorities ; dwCounter++)
    {
        wsprintf(TextualSID + lstrlen(TextualSID), "-%lu",
                    *GetSidSubAuthority(pSid, dwCounter) );
    }

    return TRUE;
}
```

Additional reference words: Convert LookupAccountSid SID String
KBCategory: kbprg
KBSubcategory: BseSecurity BseMisc CodeSam

How to Create 3D Controls in Client Area of Non-Dialog Window

PSS ID Number: Q130763
Authored 25-May-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SUMMARY

An application that uses standard Windows controls (edit boxes, list boxes, and so on) as part of a dialog box with the DS_3DLOOK style set, have the 3D look by default. But if an application creates child controls as part of the main window's client area, these controls do not have the 3D look by default. Applications should add the WS_EX_CLIENTEDGE style to the list of styles while creating the child controls to get the new 3D look.

MORE INFORMATION

WS_EX_CLIENTEDGE is the new extended style that gives controls (or any window for that matter) the new 3D look. When controls are created as part of a dialog box by using a dialog template based in the resource file, Windows adds the WS_EX_CLIENTEDGE style to the list of styles.

Some applications use controls as child windows in the client area of the main window. If the WS_EX_CLIENTEDGE style is not specified, these controls have the Windows version 3.11 user interface (2D look).

Use CreateWindowEx() to create controls in the client area of a non-dialog window, and make sure you OR in the WS_EX_CLIENTEDGE style. If your application is using Microsoft Foundation Classes (MFC) version 3.x, you can override the CreateEx() member function of the CEdit, CList, or any other standard control class.

Additional reference words: 4.00 95 user
KBCategory: kbprg kbui
KBSubcategory: UsrCtl

How to Create a Topmost Window

PSS ID Number: Q81137
Authored 26-Feb-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

Windows 3.1 introduces the concept of a topmost window that stays above all the non-topmost windows even when the window is not the active window.

There are two ways to add the topmost attribute to a window:

1. Use CreateWindowEx to create a new window. Specify WS_EX_TOPMOST as the value for the dwExStyle parameter.

2. Call SetWindowPos, specifying an existing non-topmost window and HWND_TOPMOST as the value for the hwndInsertAfter parameter.

SetWindowPos can also be used to remove the topmost attribute from a window. To do so, specify HWND_NOTOPMOST or HWND_BOTTOM as the value for the hwndInsertAfter parameter.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrWndw

How to Create Inheritable Win32 Handles in Windows 95

PSS ID Number: Q118605
Authored 25-Jul-1994 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK)

SUMMARY

Sometimes it is convenient for you to create an object such as a semaphore or file and then allow a child process to inherit the object's handle. This provides a means for two or more related processes to easily share an object.

Although Windows 95 does not have a security system such as the one in Microsoft Windows NT, Win32 API functions that create objects still use the SECURITY_ATTRIBUTES structure to determine whether the handle to the newly created object can be inherited. This article shows how to initialize a SECURITY_ATTRIBUTES structure to control whether an object handle can be inherited.

MORE INFORMATION

Win32 API functions that create objects require a SECURITY_ATTRIBUTES parameter to give a newly created object access-control information and to determine whether the handle to the object can be inherited.

The SECURITY_ATTRIBUTES structure contains the following members:

```
Type      Name
----      ----
DWORD     nLength;
LPVOID    lpSecurityDescriptor;
BOOL      bInheritHandle;
```

Secure Win32 operating systems such as Microsoft Windows NT use the lpSecurityDescriptor member to enforce how and by which processes an object is accessed. Because Windows 95 does not have a security system, it ignores lpSecurityDescriptor. Like Microsoft Windows NT, Windows 95 uses the bInheritHandle member to determine whether an object's handle can be inherited by child processes.

To initialize a SECURITY_ATTRIBUTES structure so that a handle can be inherited, set bInheritHandle to TRUE. The following code snippet shows how to create a mutex with an inheritable handle:

```
      // Set the length of the structure, allow the handle to be
      // inherited, and use the default security descriptor (which
      // Windows 95 will ignore, but Windows NT will use.) Then create
      // a named, initially unowned mutex whose handle can be
```

```
      // inherited.

   SECURITY_ATTRIBUTES sa;
   HANDLE              hMutex1;

   sa.nLength              = sizeof(sa);
   sa.bInheritHandle       = TRUE;
   sa.lpSecurityDescriptor = NULL;

   hMutex1 = CreateMutex(&sa, FALSE, "MUTEX1");
```

To prevent the handle from being inherited, set bInheritHandle to FALSE. The following code example demonstrates creating a mutex with a noninheritable handle:

```
      // Set the length of the structure, do not allow the handle
      // to be inherited, and use the default security descriptor
      // (which Windows 95 will ignore, but Windows NT will use).
      // Create a named, initially unowned mutex whose handle cannot
      // be inherited.

   SECURITY_ATTRIBUTES sa;
   HANDLE              hMutex1;

   sa.nLength              = sizeof(sa);
   sa.bInheritHandle       = FALSE;
   sa.lpSecurityDescriptor = NULL;

   hMutex1 = CreateMutex(&sa, FALSE, "MUTEX1");
```

You can also prevent a handle to an object from being inherited by specifying NULL in the call to Win32 object creation API function instead of specifying a pointer to a SECURITY_ATTRIBUTES structure. This is equivalent to setting bInheritHandle to FALSE and lpSecurityDescriptor to NULL. For example:

```
      // Use NULL instead of pointer to SECURITY_ATTRIBUTES
      // structure to create a named, initially unowned
      // mutex whose handle cannot be inherited. A NULL security
      // descriptor will be used by Windows NT, but ignored by
      // Windows 95.

   HANDLE hMutex1;
   hMutex1 = CreateMutex(NULL, FALSE, "MUTEX1");
```

Cross-Platform Compatibility Information

Keep in mind that while Windows 95 does not have a security system, Windows NT does. Windows 95 ignores the lpSecurityDescriptor member of the SECURITY_ATTRIBUTES, but Windows NT uses it. If access to the object needs to be controlled in a specific way on Windows NT, then the lpSecurityDescriptor should be initialized by calling the Win32 security API functions.

Additional reference words: 4.00
KBCategory: kbprg
KBSubcategory: BseSecurity

How to Create Non-rectangular Windows

PSS ID Number: Q125669
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

Windows 95 and Windows NT version 3.51 provide a new API called SetWindowRgn() that makes it easy for applications to create irregularly shaped windows.

MORE INFORMATION

In previous versions of Windows and Windows NT, it was possible to create only rectangular windows. To simulate a non-rectangular window required a lot of work on the application developer's part. Besides handling all drawing for the window, the application was required to perform hit-testing and force underlying windows to repaint as necessary to refresh the "transparent" portions of the window.

Windows 95 and Windows NT version 3.51 greatly simplify this by providing the SetWindowRgn function. An application can now create a region with any desired shape and use SetWindowRgn to set this as the clipping region for the window. Subsequent painting and mouse messages are limited to this region, and Windows automatically updates underlying windows that show through the non-rectangular window. The application need only paint the window as desired.

For more information on using SetWindowRgn, see the Win32 API documentation.

Additional reference words: 4.00
KBCategory: kbprg
KBSubcategory: UsrMisc

How to Delete Keys from the Windows NT Registry

PSS ID Number: Q127990
Authored 22-Mar-1995 Last modified 23-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 3.5

There are two ways to delete registry keys: use REGEDT32.EXE or call RegDeleteKey() from your application.

The documentation for RegDeleteKey() points out that the specified key to be deleted must not have subkeys. If the key to be deleted does have subkeys, RegDeleteKey() will fail with access denied. This happens despite the fact that the machine account has delete privileges and the registry handle passed to RegDeleteKey() was opened with delete access. The additional requirement is that the key must have no subkeys.

This limitation does not exist in 16-bit Windows. The difference exists in Windows NT because of atomicity and security considerations that 16-bit Windows does not have.

You can select a key with subkeys and delete it with REGEDT32. This is because REGEDT32 recursively deletes the subkeys for you, making multiple call to RegDeleteKey(). You should use recursive subkey deletion in your application as well, if you need to delete keys that have subkeys.

Additional reference words: 3.50
KBCategory: kbprg
KBSubcategory: BseMisc

How to Design Multithreaded Applications to Avoid Deadlock

PSS ID Number: Q126768
Authored 01-Mar-1995 Last modified 25-May-1995

--
The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.5, 3.51, and 4.0
--

SUMMARY

Debugging a multithreaded application that deadlocks is challenging because the debugger cannot identify for you which thread owns which resource. You would have to track this information in your code. Because it is difficult to debug a deadlock situation, it is important to design your application to avoid deadlock.

This article is a brief introduction to a very complex topic. There are references at the bottom of this article for additional information.

The key point to keep in mind when designing a multithreaded program is that resources must always be requested in the same order.

MORE INFORMATION

The Win32 API provides WaitForSingleObject() and WaitForMultipleObjects() for requesting resources with handles. You would use a different method to request other resources, depending on the resource type.

Many deadlocks occur because resources are not requested in the same order by the application threads. For example:

- Thread 1 holds resource A and wants resource B.

- Thread 2 holds resoruce B and wants resource A.

Both threads block forever, resulting in deadlock. There are many other possible scenarios.

To avoid this problem, identify all of your application's critical resources and order them from least precious to most precious. Design your code such that if a thread needs several resources, it requests them in order, starting with the least precious resource. Resources should be freed in the reverse order and as soon as it is possible. This is not a requirement to avoid deadlock, but it is good practice.

In the example given above, suppose that resource B is more precious than resource A. Here's how the code would resolve the situation:

- Thread 2 already holds B, but because it wants A, it releases B and waits for A.

- Thread 1 grabs B, then begins the task. It releases A when possible.

- Thread 2 grabs A and waits for B.

- Thread 1 finishes the task, then releases B.

- Thread 2 grabs B, finishes the task, then releases A, then releases B.

The reason you should request the least precious resource first is that it doesn't matter as much if you hold it longer while waiting to acquire all the resources that you need. If the resource is precious, you want to hold it for the smallest amount of time possible, so other threads can use it.

REFERENCES

MSDN Development Library, "Detecting Deadlocks in Multithreaded Win32 Applications", by Ruediger Asche.

For more information, refer to a good book on operating system design.

Additional reference words: 3.50 4.00 95 race condition
KBCategory: kbprg
KBSubcategory: BseProcThrd

How to Detect All Program Terminations

PSS ID Number: Q125689
Authored 01-Feb-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

The processes for detecting program terminations in fall into two categories:

- Win32 processes use WaitForSingleObject() or WaitForMultipleObjects() to wait for other Win32 processes, Windows 16-bit processes, and MS-DOS-based applications to terminate.

- Windows 16-bit processes, on the other hand, must use the TOOLHELP NotifyRegister() function. The 16-bit processes can be notified when other 16-bit processes and MS-DOS-based applications exit, but have the limitation of not being notified of Win32 process activity.

MORE INFORMATION

Win32 processes can use WaitForSingleObject() or WaitForMultipleObjects() to wait for a spawned process. By using CreateProcess() to launch a Win32 process, 16-bit process, or MS-DOS-based application, you can fill in a PROCESS_INFORMATION structure. The hProcess field of this structure can be used to wait until the spawned process terminates. For example, the following code spawns a process and waits for its termination:

```
STARTUPINFO StartupInfo = {0};
StartupInfo.cb = sizeof(STARTUPINFO);
if (CreateProcess(NULL, szCommandLine, NULL, NULL, FALSE,
                  0, NULL, NULL, &StartupInfo, &ProcessInfo))
{
    WaitForSingleObject(ProcessInfo.hProcess, INFINITE);
    /* Process has terminated */
    ...
}
else
{
    /* Process could not be started */
    ...
}
```

If necessary, you can put this code into a separate thread to allow the initial thread to continue to execute.

This synchronization method is not available to 16-bit processes. Instead, they must use the TOOLHELP NotifyRegister function to register a callback

function to be called when a program terminates. This method will detect the termination of 16-bit processes and MS-DOS-based applications, but not Win32 processes.

The following code shows how to register a callback function with NotifyRegister():

```
FARPROC lpfnCallback;

lpfnCallback = MakeProcInstance(NotifyRegisterCallback, ghInst);
if (!NotifyRegister(NULL, (LPFNNOTIFYCALLBACK)lpfnCallback,
                    NF_NORMAL))
{
   MessageBox(NULL, "NotifyRegister Failed", "Error", MB_OK);
   FreeProcInstance(lpfnCallback);
}
```

The next section of code demonstrates the implementation of the callback function:

```
BOOL FAR PASCAL __export NotifyRegisterCallback (WORD wID,
                                                 DWORD dwData)
{
   HTASK hTask;  // task that called the notification callback
   TASKENTRY te;

   // Check for task exiting
   switch (wID)
   {
      case NFY_EXITTASK:
         // Obtain info about the task that is terminating
         hTask = GetCurrentTask();
         te.dwSize = sizeof(TASKENTRY);
         TaskFindHandle(&te, hTask);

         // Check if the task that is terminating is our child task.
         // Also check if the hInstance of the task that is
         // terminating is the same as the hInstance of the task
         // that was WinExec'd by us earlier in the program.

         if (te.hTaskParent == ghtaskParent &&
             te.hInst == ghInstChild)
             PostMessage(ghwnd, WM_USER+509, (WORD)te.hInst, dwData);
         break;

      default:
         break;
   }
   // Pass notification to other callback functions
   return FALSE;
}
```

The NotifyRegisterCallback() API is called by the 16-bit TOOLHELP DLL in the context of the process that is causing the event. Problems arising because of reentrancy and notification chaining makes the callback function

subject to certain restrictions. For example, operations that cause TOOLHELP events cannot be done in the callback function. (See the TOOLHELP NotifyRegister function documentation in your Software Development Kit for events that cause TOOLHELP callbacks.)

There is no way a 16-bit process can be notified when a Win32 process exits. However, a 16-bit process can use TaskFirst() and TaskNext() to periodically walk the task list to determine if a Win32 process is still executing. This technique also works for 16-bit processes and MS-DOS-based applications. For example, the following code shows how to check for the existence of a process:

```
BOOL StillExecuting(HINSTANCE hAppInstance)
{
   TASKENTRY  te = {0};

   te.dwSize = sizeof(te);
   if (TaskFirst(&te))
       do
       {
          if (te.hInstance == hAppInstance)
             return TRUE;      // process found
       } while (TaskNext(&te));

   // process not found
   return FALSE;
}
```

Refer to the TermWait sample for complete details on how to use NotifyRegister and implement a callback function. For additional information, please search in the Microsoft Knowledge Base using this word:

```
TERMWAIT
```

Additional reference words: 4.00 95 end exit notification notify spawn terminate termination
KBCategory: kbprg kbcode
KBSubcategory: BseProcThrd

How to Detect Slow CPU & Unaccelerated Video Under Windows 95

PSS ID Number: Q131259
Authored 07-Jun-1995 Last modified 22-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

Under Windows 95, use GetSystemMetrics(SM_SLOWMACHINE) to check for low-end computers. It returns a nonzero value if the computer has a 386 CPU, is low on memory, or has a slow display card.

The return values (bit flags) are:

0x0001	- CPU is a 386
0x0002	- low memory machine (less than 5 megabytes)

The following is notable for video:

0x0004	- slow (nonaccerated) display card

Additional reference words: 4.00 win95 system display slow machine
KBCategory: kbprg
KBSubcategory: UsrSys

How to Determine If a Device Is Palette Capable

PSS ID Number: Q72387
Authored 23-May-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

On a device that supports the Windows palette management function, an application can create a logical palette, select the palette into a device context (DC), and realize the palette, which maps its colors into the system (hardware) palette. The GetDeviceCaps() API informs an application whether a given device is capable of performing palette manipulation and, if so, the size of the palette. This article discusses the GetDeviceCaps() API and how it is used.

MORE INFORMATION

To determine whether a device can perform palette operations, call the GetDeviceCaps() API with the RASTERCAPS parameter. If the RC_PALETTE bit of the return is set, the device supports the palette management functions.

To determine how many colors in the system palette are available for the application to use, the following parameters can be used in a GetDeviceCaps() call:

Parameter	Description
SIZEPALETTE	Total number of entries in the system palette
NUMRESERVED	Number of reserved (static) colors in the system palette

This functionality is demonstrated in the MyPal sample code that is included on the Windows version 3.x Software Development Kit (SDK) Source Code 2 disk. For a demonstration, start MyPal and click the right mouse button.

The reserved colors are entries in the system palette that are used by Windows and cannot be changed by an application [except by using SetSystemPaletteUse(), which is not recommended]. The reserved colors are used for the following purposes:

- Active border
- Active caption
- Background color MDI applications
- Desktop background color

Push button faces
Push button edges
Push button text
Caption text
Disabled (gray) text
Highlight color in controls (to highlight items in the control)
Highlight text color
Inactive border
Inactive caption
Inactive caption text (new to Windows version 3.1)
Menu background
Menu text
Scroll-bar gray area
Window background
Window frame
Window text

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiPal

How to Determine the Japanese OEM Windows Version

PSS ID Number: Q130054
Authored 10-May-1995 Last modified 16-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 Software Development Kit (SDK) version 3.5
- Microsoft Win32s version 1.2

SUMMARY

Because of the variety of computer manufacturers (NEC, Fujitsu, IBMJ, and so on) in Japan, sometimes Windows-based applications need to know which OEM (original equipment manufacturer) manufactured the computer that is running the application. This article explains how.

MORE INFORMATION

There is no documented way to detect the manufacturer of the computer that is currently running an application. However, a Windows-based application can detect the type of OEM Windows by using the return value of the GetKeyboardType() function.

If an application uses the GetKeyboardType API, it can get OEM ID by specifying "1" (keyboard subtype) as argument of the function. Each OEM ID is listed here:

OEM Windows	OEM ID
Microsoft	00H (DOS/V)
all AX	01H
EPSON	04H
Fujitsu	05H
IBMJ	07H
Matsushita	0AH
NEC	0DH
Toshiba	12H

Application programs can use these OEM IDs to distinguish the type of OEM Windows. Note, however, that this method is not documented, so Microsoft may not support it in the future version of Windows.

As a rule, application developers should write hardware-independent code, especially when making Windows-based applications. If they need to make a hardware-dependent application, they must prepare the separated program file for each different hardware architecture.

Additional reference words: 3.10 1.20 3.50 1.20 kbinf
KBCategory: kbhw

KBSubcategory: wintldev

How to Determine the Type of Handle Retrieved from

PSS ID Number: Q126258
Authored 16-Feb-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

OpenPrinter returns a valid handle when a printer name or a server name is passed to it. Sometimes it may be necessary to determine if the returned handle is a handle to a printer because some Win32 spooler functions only accept printer handles and will fail on server handles. The following code determines if a handle is a printer handle:

```
BOOL IsPrinterHandle( HANDLE hPrinter)
{
    DWORD       cbNeeded;
    DWORD       Error;
    BOOL        bRet = FALSE;
    LPPRINTER_INFO_2 pPrinter;
    DWORD       cbBuf;
    HANDLE      hMem = NULL;

    if( !GetPrinter(hPrinter, 2, (LPBYTE)NULL, cbBuf, &cbNeeded ))
    {
        Error = GetLastError( );

        if( Error == ERROR_INSUFFICIENT_BUFFER )
        {
            hMem = GlobalAlloc(GHND, cbNeeded);
            if (!hMem) return bRet;
            pPrinter = (LPPRINTER_INFO_2)GlobalLock(hMem);
            cbBuf = cbNeeded;
            if(GetPrinter(hPrinter, 2, (LPBYTE)pPrinter, cbBuf, &cbNeeded))
            {
                bRet = TRUE;
                GlobalUnlock(hMem);
                GlobalFree(hMem);
            }
            else SetLastError( ERROR_INVALID_PRINTER_NAME );
        }
        else if( Error == ERROR_INVALID_HANDLE )
        {
            SetLastError( ERROR_INVALID_PRINTER_NAME );
        }
    }
    return bRet;
}
```

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg kbcode
KBSubcategory: GdiPrn

How to Determine Which Version of Win32s Is Installed

PSS ID Number: Q121091
Authored 26-Sep-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s, versions 1.1, 1.15, and 1.2

SUMMARY

The "Win32s Programmer's Reference" describes how to use the GetWin32sInfo() function in a program to determine which version of Win32s is installed. This article explains how to determine which version of Win32s is installed interactively.

MORE INFORMATION

Use one of the following two methods to determine interactively which version of Win32s is installed:

1. Check the WIN32S.INI file in your Windows system directory. The Version entry contains the major version and the build number (m.mm.bbb.b). This entry should be modified by any Win32-based application which installs a later version of Win32s on your Windows machine.

   NOTE: Because it is up to the application vendor to set this value when installing Win32s, the value may not be accurate. Microsoft strongly urges all independent software vendors (ISVs) to modify the WIN32S.INI file so that this information is available to customers.

   -or-

2. If Win32s is installed on top of Windows for Workgroups, select the WIN32S16.DLL file from the Windows system directory in File Manager. Then from the File menu, choose Properties. The Version line contains the major version and the build number.

Version Information

Win32s version 1.1.88 was distributed as Win32s version 1.1 and Win32s version 1.1.89 was distributed as Win32s version 1.1a.

Win32s version 1.15.103 was distributed as Win32s version 1.15 and Win32s version 1.15.111 was distributed as Win32s version 1.15a.

Win32s version 1.2.123 was distributed as Win32s version 1.2.

Additional reference words: 1.10 1.15 1.20
KBCategory: kbenv
KBSubCategory: W32s

How to Disable the Screen Saver Programmatically

PSS ID Number: Q126627
Authored 27-Feb-1995 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.5 and 3.51

SUMMARY

Under Windows NT, you can disable the screen saver from your application code. To detect if the screen saver is enabled, use this:

```
SystemParametersInfo( SPI_GETSCREENSAVEACTIVE,
                      0,
                      pvParam,
                      0
                    );
```

On return, the parameter pvParam will point to TRUE if the screen saver setting is enabled in the System control panel applet and FALSE if the screen saver setting is not enabled.

To disable the screen saver setting, call SystemParametersInfo() with this:

```
SystemParametersInfo( SPI_SETSCREENSAVEACTIVE,
                      FALSE,
                      0,
                      SPIF_SENDWININICHANGE
                    );
```

MORE INFORMATION

When the screen saver is activated by the system, it is run on a desktop other than the user's desktop (similar to the login desktop displayed when no one is logged in). Therefore, you cannot use FindWindow() to determine if the screen saver is currently active.

Here are two methods that you can use to detect if the screen saver is currently running:

1. Get the name of the current screen saver from the registry, parse the PE header of the screen saver binary to get the process name, then check for an active process with that name in the performance registry.

   -or-

2. Write a screen saver that would be spawned by the system and would in turn spawn the "real" screen saver. The first screen saver could notify your application when the screen saver has been activated or deactivated.

Additional reference words: 3.50
KBCategory: kbprg
KBSubcategory: GdiScrsav

How to Display Debugging Messages in Windows 95

PSS ID Number: Q125868
Authored 07-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

In Windows 95, 16-bit and 32-bit Windows-based applications may use OutputDebugString() to display debug messages. Furthermore, the 16-bit and 32-bit system DLLs may also display debug messages. This article describes how to view these messages during application developement.

MORE INFORMATION

It is possible to use the 16-bit DBWIN application to display debug messages from 16-bit Windows-based applications and from the debugging versions of system DLLs (such as GDI.EXE, USER.EXE, and KRNL386.EXE). To receive debug messages via DBWIN, you must install the Windows 95 SDK debug components.

To receive messages from 32-bit Windows-based applications under Windows 95, you must debug the application with a Win32 debugger such as WinDbg, or install WDEB386 as a .VxD or in the AUTOEXEC.BAT file. To receive messages from the debugging versions of the 32-bit system DLLs (KERNEL32.DLL, USER32.DLL, and GDI32.DLL), you must install the Windows 95 SDK debugging components, in conjunction with WDEB386.

You can use WDEB386 to display debug messages from both 16-bit and 32-bit Windows-based applications and from the debugging versions of system components. Because WDEB386 works over a serial communications port, it is necessary to use a serial terminal or second computer to operate it. For more information about configuring and using WDEB386, please search for articles in the Microsoft Knowledge Base by using this word:

WDEB386

Alternative system level debuggers, which provide functionality similiar to WDEB386, may in the future be provided by third-party vendors.

Also, you can write 32-bit application-level debuggers that display debug messages from the debuggee by handling the DEBUG_EVENT structure member OUTPUT_DEBUG_STRING_EVENT.

Additional reference words: 3.95 4.00
KBCategory: kbprg
KBSubcategory: BseErrdebug

How to Display Old-Style FileOpen Common Dialog in Windows 95

PSS ID Number: Q131282
Authored 07-Jun-1995 Last modified 10-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SUMMARY

If you want an application to revert back to the old-style FileOpen or SaveAs common dialog box, you must either provide a dialog template or a hook function in addition to not specifying the OFN_EXPLORER flag.

MORE INFORMATION

Windows 95 provides a new flag for the File Open or Save As common dialog box called OFN_EXPLORER. When set, this flag ensures that the File Open dialog box displays a user interface that is similar to the Windows Explorer (or so-called Explorer-style dialog box).

You may want your application to revert to the old Windows version 3.1 style dialog box. For example, you might want to maintain a user interface consistent with the Windows NT user interface. Windows NT version 3.51 currently does not support the new Explorer-style File Open common dialog box. The next version of Windows NT, however, should implement this new feature.)

To display the old-style common dialog box:

- Don't specify the OFN_EXPLORER value in the OPENFILENAME structure's Flags member.

- Provide a dialog template or a hook. If the application does not have either one, a simple hook that always returns FALSE should suffice.

NOTE: When you specify a hook function for the old-style common dialog box in Windows 95, the hDlg received in the hookProc is the actual handle to the dialog containing the standard controls. This is not true, however, for the new Explorer-style common dialog hookProc where the hDlg received is the handle to a child of the dialog box containing the standard controls. Therefore, to get a handle to the actual File Open or Save As dialog box, an application should call GetParent() on the hDlg passed to the hook procedure.

Additional reference words: 4.00 subdialog
KBCategory: kbprg kbui
KBSubcategory: UsrCmnDlg

How to Draw a Custom Window Caption

PSS ID Number: Q99046
Authored 20-May-1993 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

Microsoft Windows draws captions in the caption bar (or title bar) for all eligible windows in the system. Applications need to specify only the WS_CAPTION style to take advantage of this facility. The current version of Microsoft Windows, however, imposes three significant restrictions on the captions. An application that does not want to be tied by any of these restrictions may want to draw its own caption. This article lists the restrictions and the steps required to draw a window caption.

These restrictions also apply to Windows NT, but there are a few differences for Windows 95.

It is important to note that an application should not draw its own caption unless it has very good reasons to do so. A window caption is a user interface object, and rendering it in ways different from other windows in the system may obstruct the user's conceptual grasp of the Microsoft Windows user interface.

MORE INFORMATION

Windows and Windows NT

The three important restrictions imposed by Microsoft Windows version 3.1 and Microsoft Windows NT on the caption for a window are:

- It consists of text only; graphics are not allowed.
- All text is centered and drawn with the system font.
- The length of the displayed caption is limited to 78 characters even when there is space on the caption bar to accommodate extra characters.

An application can essentially render its own caption consisting of any graphic and text with the standard graphics and text primitives by painting on the nonclient area of the window. The application should draw in response to the WM_NCPAINT, WM_NCACTIVATE, WM_SETTEXT, and WM_SYSCOMMAND messages. When processing these messages, an application should first pass on the message to DefWindowProc() for default processing, then render its caption before returning from the message. This ensures that Microsoft Windows can properly draw the nonclient area. Because drawing the caption

is part of DefWindowProc()'s nonclient area processing, an application should specify an empty window title to avoid any Windows-initiated drawing in the caption bar. The following steps indicate the computations needed to determine the caption drawing area:

1. Get the current window's rectangle using GetWindowRect(). This includes client plus nonclient areas and is in screen coordinates.

2. Get a device context (DC) to the window using GetWindowDC().

3. Compute the origin and dimensions of the caption bar. One needs to account for the window decorations (frame, border) and window bitmaps (min/max/system boxes). Remember that different window styles will result in different decorations and a different number of min/max/system boxes. Use GetSystemMetrics() to compute the dimensions of the frame, border, and the system bitmap dimensions.

4. Render the caption within the boundaries of the rectangle computed in step 3. Remember that the user can change the caption bar color any time by using the Control Panel. Some components of the caption, particularly text backgrounds, may need to be changed based on the current caption bar color. Use GetSysColor to determine the current color.

The following code sample draws a left-justified caption for a window (the code sample applies only to the case where the window is active):

Sample Code

```
   case WM_NCACTIVATE:
      if ((BOOL)wParam == FALSE)
      {
         DefWindowProc( hWnd, message, wParam, lParam );
      // Add code here to draw caption when window is inactive.

         return TRUE;
      }
      // Fall through if wParam == TRUE, i.e., window is active.

      case WM_NCPAINT:
      // Let Windows do what it usually does. Let the window caption
      // be empty to avoid any Windows-initiated caption bar drawing

         DefWindowProc( hWnd, message, wParam, lParam );
         hDC = GetWindowDC( hWnd );
         GetWindowRect( hWnd, (LPRECT)&rc2 );

      // Compute the caption bar's origin. This window has a system box
      // a minimize box, a maximize box, and has a resizeable frame

         x = GetSystemMetrics( SM_CXSIZE ) +
             GetSystemMetrics( SM_CXBORDER ) +
             GetSystemMetrics( SM_CXFRAME );
         y = GetSystemMetrics( SM_CYFRAME );
```

```
            rc1.left = x;
            rc1.top = y;

         // 2*x gives twice the bitmap+border+frame size. Since there are
         // only two bitmaps, two borders, and one frame at the end of the
         // caption bar, subtract a frame to account for this.

            rc1.right = rc2.right - rc2.left - 2*x -
                        GetSystemMetrics( SM_CXFRAME );
            rc1.bottom = GetSystemMetrics( SM_CYSIZE );

         // Render the caption. Use the active caption color as the text
         // background.

            SetBkColor( hDC, GetSysColor(COLOR_ACTIVECAPTION) );
            DrawText( hDC, (LPSTR)"Left Justified Caption", -1,
                    (LPRECT)&rc1, DT_LEFT );
          ReleaseDC( hWnd, hDC );
          break;
```

Windows 95

On Windows 95, the text is not centered and the user can choose the Font. In addition, your application might want to monitor the WM_WININICHANGED message, because the user can change titlebar widths, and so forth, dynamically. When this happens, the application should take the new system metrics into account, and force a window redraw.

Additional reference words: 3.00 3.10 3.50 4.00 minimum maximum
KbCategory: kbprg kbcode
KbSubCategory: UsrPnt

How to Draw a Gradient Background

PSS ID Number: Q128637
Authored 05-Apr-1995 Last modified 23-May-1995

-
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

-

SUMMARY

This article provides source code for drawing a gradient background pattern similar to the one used in Microsoft Setup applications. The code will compile and run on Windows version 3.1, Win32s, and Windows 95.

MORE INFORMATION


```
/************************************************************
*                                                           *
*  DrawBackgroundPattern()                                  *
*                                                           *
*  Purpose: This function draws a gradient pattern that     *
*           transitions between blue and black. This is     *
*           similar to the background used in Microsoft     *
*           setup programs.                                 *
*                                                           *
************************************************************/
void DrawBackgroundPattern(HWND hWnd)
{
  HDC hDC = GetDC(hWnd);  // Get the DC for the window
  RECT rectFill;          // Rectangle for filling band
  RECT rectClient;        // Rectangle for entire client area
  float fStep;            // How large is each band?
  HBRUSH hBrush;
  int iOnBand;  // Loop index

  // How large is the area you need to fill?
  GetClientRect(hWnd, &rectClient);

  // Determine how large each band should be in order to cover the
  // client with 256 bands (one for every color intensity level)
  fStep = (float)rectClient.bottom / 256.0f;
```

```
  // Start filling bands
  for (iOnBand = 0; iOnBand < 256; iOnBand++) {

    // Set the location of the current band
    SetRect(&rectFill,
            0,                          // Upper left X
            (int)(iOnBand * fStep),     // Upper left Y
            rectClient.right+1,         // Lower right X
            (int)((iOnBand+1) * fStep)); // Lower right Y

    // Create a brush with the appropriate color for this band
    hBrush = CreateSolidBrush(RGB(0, 0, (255 - iOnBand)));


    // Fill the rectangle
    FillRect(hDC, &rectFill, hBrush);

    // Get rid of the brush you created
    DeleteObject(hBrush);
  };

  // Give back the DC
  ReleaseDC(hWnd, hDC);
}
```

Additional reference words: 3.00 3.10 3.50 4.00 GRADIENT BACKGROUND DITHER
KBCategory: kbprg kbcode
KBSubcategory: GdiMisc

How to Examine the Use of Process Memory Under Win32s

PSS ID Number: Q129599
Authored 30-Apr-1995 Last modified 01-May-1995

The information in this article applies to:

- Microsoft Win32s, version 1.2

SUMMARY

Under Windows, tools like HeapWalk and PWalk can be used to examine memory use of 16-bit code. However, these tools cannot be used to look at memory use of 32-bit code. This article discusses how to look at process memory use under Win32s.

MORE INFORMATION

If you run the debug version of Win32s and kernel debugger WDEB386, you can break into the debugger at any point by pressing CTRL+C and using debug information from the Win32s VxD. Use the command .w32s to get the list of information types available.

```
#.w32s
W32S debug routines:

A - General Info
B - Print Free LS ranges
C - Print RRD & Section lists
D - Print Modules list
E - Toggle SwapOut trace
F - Toggle PageFault trace
G - count present alias pages
H - List RRD Commit List
I - Toggle Virtual Alloc/Free trace
J - Toggle Mapped Section trace
K - List Locked Pages
[ESC] Exit W32S Debug Routines
```

Option C gives you information about the sparse memory usage.

The memory for .EXE and .DLL files is allocated in the sparse memory. Here's an example printout using option C:

```
RRD List:
Index      Start      Size       Owner      #Commits   CommSize   #PresPg
00000000   87AA0000   0000E000   00000000   - VIEW -   - VIEW -   - VIEW -
00000001   87A90000   00001000   00001F37   00000001   00001000   00000001
00000002   87A50000   00040000   00000000   00000000   00000000   00000000
00000003   87A40000   00002000   00000000   - VIEW -   - VIEW -   - VIEW -
00000004   87A30000   00009000   00000000   - VIEW -   - VIEW -   - VIEW -
00000005   87A20000   00002000   00000000   - VIEW -   - VIEW -   - VIEW -
```

```
00000006   87A10000   00002000   00000000   - VIEW -   - VIEW -   - VIEW -
00000007   87910000   00100000   00001F37   00000001   00001000   00000001
00000008   878F0000   00020000   00001F37   00000001   00020000   00000002
00000009   878C0000   00021000   00001F37   00000001   00021000   00000001
0000000A   878B0000   00005000   00000000   00000001   00005000   00000005
0000000B   87860000   00043000   00000000   00000001   00043000   00000031
0000000C   87830000   0002D000   00000000   00000001   0002D000   0000000C
0000000D   87810000   00011000   00000000   00000001   00011000   0000000E
0000000E   80869000   00001000   00001F37   00000001   00001000   00000000
0000000F   87800000   00002000   00001F37   00000001   00002000   00000001
00000010   80635000   00001000   00001F37   00000001   00001000   00000000
                      ========                         ========   ========
Total                 00229000                         000CD000   00000056

Sections List:
SecIndex   hFile      SecSize    #Ref       #Views     CommSize   #PresPg
00000001   00000004   00002000   00000000   00000001   00002000   00000001
00000002   00000005   00002000   00000000   00000001   00002000   00000001
00000003   00000006   00009000   00000000   00000001   00009000   00000003
00000004   00000007   00002000   00000000   00000001   00002000   00000001
00000005   00000008   0000E000   00000000   00000001   0000E000   00000001
                      ========                         ========   ========
Total                 0001D000                         0001D000   00000007

G. Total                                               000EA000   0000005D
```

The Size column contains the reserved size and the CommSize column contains the committed size. The addresses are zero-based (ring 0), not based on 0xffff0000 (ring 3). Therefore, you must add 0x10000 to the addresses you see in the list in order to get the ring 3 addresses.

Option D gives you the list of modules and where they reside in memory. These addresses are zero-based addresses as well, as is any information that you get from the VxD.

Another way to get information indicating where things are placed in memory is to set the verbose loader flag (0x20) in the Win32sDebug variable in the [386Enh] section of the SYSTEM.INI file.

NOTE: Do not add the 0x, just write Win32sDebug=20. The loader then will print in the debug terminal information about each loaded module. For example:

```
Open file D:\WIN32APP\FREECELL\FREECELL.EXE in mode 0xa0
LELDR: allocating 0x11000
LELDR: Module D:\WIN32APP\FREECELL\FREECELL.EXE [1] loaded at 0x87820000
LELDR: obj  1 loaded @ 0x87821000, 0x   5c00 bytes   .text,flags=0x60000020
LELDR: obj  2 loaded @ 0x87827000, 0x      0 bytes    .bss,flags=0xc0000080
LELDR: obj  3 loaded @ 0x87828000, 0x    200 bytes  .rdata,flags=0x40000040
LELDR: obj  4 loaded @ 0x87829000, 0x    a00 bytes   .data,flags=0xc0000040
LELDR: obj  5 loaded @ 0x8782a000, 0x   2400 bytes   .rsrc,flags=0x40000040
LELDR: obj  6 loaded @ 0x8782d000, 0x    200 bytes    .CRT,flags=0xc0000040
LELDR: obj  7 loaded @ 0x8782e000, 0x    a00 bytes  .idata,flags=0x40000040
LELDR: obj  8 loaded @ 0x8782f000, 0x   1e00 bytes  .reloc,flags=0x42000040
File D:\WIN32APP\FREECELL\FREECELL.EXE is closed
```

The addresses here are ring 3 addresses.

REFERENCES

Please see the "Win32s Programmer's Reference" included in the Win32 SDK for more information about the debugging features. This information is not included in the version of the Win32s documentation distributed with Visual C++.

Additional reference words: 1.20
KBCategory: kbprg
KBSubcategory: W32s

How to Find Out Which Listview Column Was Right-Clicked

PSS ID Number: Q125694
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

You can use the technique described in this article to find out which column was clicked after right-clicking the listview column header.

MORE INFORMATION

LVN_COLUMNCLICK notifies a listview's parent window when a column is clicked using the left mouse button, but no such notification occurs when a column is clicked with the right mouse button.

Windows 95 sends an NM_RCLICK notification to the listview's parent window when a column is clicked with the right mouse button, but the message sent does not contain any information as to which column was clicked, especially if the window is sized so that the listview is scrolled to the right.

The correct way to determine which column was clicked with the right mouse button, regardless of whether the listview is scrolled, is to send the header control an HDM_HITTEST message, which returns the index of the column that was clicked in the iItem member of the HD_HITTESTINFO struct. In sending this message, make sure the point passed in the HD_HITTESTINFO structure is relative to the header control's client coordinates. Do not pass it a point relative to the listview's client coordinates; if you do, it will return an incorrect column index value.

The header control in this case turns out to be a child of the listview control of LVS_REPORT style.

The following code demonstrates this method. Note that while the code processes the NM_RCLICK notification on a WM_NOTIFY message, you also process the WM_CONTEXTMENU message, which is also received as a notification when the user clicks the right mouse button.

```
case WM_NOTIFY:
{
    if ((((LPNMHDR)lparam)->code == NM_RCLICK))
    {
       HWND hChildWnd;
       POINT pointScreen, pointLVClient, pointHeader;
       DWORD dwpos;

       dwPos = GetMessagePos();
```

```
        pointScreen.x = LOWORD (dwPos);
        pointScreen.y = HIWORD (dwPos);

        pointLVClient = pointScreen;

        // Convert the point from screen to client coordinates,
        // relative to the listview
        ScreenToClient (ghwndLV, &pointLVClient);

        // Because the header turns out to be a child of the
        // listview control, we obtain its handle here.
        hChildWnd = ChildWindowFromPoint (ghwndLV, pointLVClient);

        // NULL hChildWnd means R-CLICKED outside the listview.
        // hChildWnd == ghwndLV means listview got clicked: NOT the
        // header.
        if ((hChildWnd) && (hChildWnd != ghwndLV))
        {
           char szClass [50];

           // Verify that this window handle is indeed the header
           // control's by checking its classname.
           GetClassName (hChildWnd, szClass, 50);
           if (!lstrcmp (szClass, "SysHeader32"))
           {
              HD_HITTESTINFO hdhti;
         char szBuffer [80];

              // Transform to client coordinates
              // relative to HEADER control, NOT the listview!
              // Otherwise, incorrect column number is returned.

              pointHeader = pointScreen;
              ScreenToClient (hChildWnd, &pointHeader);

              hdhti.pt = pointHeader;
              SendMessage (hChildWnd,
                           HDM_HITTEST,
                           (WPARAM)0,
                           (LPARAM) (HD_HITTESTINFO FAR *)&hdhti);
             wsprintf (szBuffer, "Column %d got clicked.\r\n", hdhti.iItem);

             MessageBox (NULL, szBuffer, "Test", MB_OK);
           }
        }
     }
     return 0L;
}

Additional reference words: 4.00
KBCategory: kbprg kbcode
KBSubcategory: UsrCtl
```

How to Find the Version Number of Win32s

PSS ID Number: Q125014
Authored 19-Jan-1995 Last modified 20-Jan-1995

--
The information in this articles applies to:

- Microsoft Win32s, versions 1.0, 1.1, and 1.2

--

SUMMARY

This article describes how to obtain the version number information for Win32s installed on a Windows 3.1 machine from one of the following places:

- From an end user perspective, on either Windows for Workgroups 3.11 or Windows NT 3.5
- A 16-bit application running on Windows 3.1
- A 32-bit application running on Windows 3.1

The following section describes in more details how to obtain this information in all the above situations.

MORE INFORMATION

From an End User Perspective

Because Win32s does not have a user interface, there is no obvious way to get the version number information for Win32s that is installed on Windows 3.1. However, end users have the following two options:

- Read the <windir>\SYSTEM\WIN32S.INI file, which has an entry for version information. Because this .INI file can be updated by the Setup program of any Win32s application, this information is not completely reliable.

- From File Manager on Windows for Workgroups 3.11 or Windows NT 3.5, select the WIN32S16.DLL and choose Properties from the File menu. This method yields a dialog box with version information on Win32s. Remember that WIN32S16.DLL is a 16-bit DLL; however, File Manager on Windows NT 3.5 can still read this version resource information.

From a 16-Bit Application

To get version number information for Win32s from a 16-bit application, use the Win32s specific function, GetWin32sInfo(), which is documented in the Win32s Programmer's Reference. This function is exported by the 16-bit W32SYS.DLL file in Win32s 1.1 and later. The GetWin32sInfo() function fills a specified structure with the information from Win32s VxD. Usually a 16-bit Windows setup program should use this function to determine if Win32s

is already installed before continuing installation. Note that a 16-bit program must use LoadLibrary and GetProcAddress to call the function because the function did not exist in Win32s version 1.0.

The following example on using GetWin32sInfo() is extracted from the Win32s Programmer's Reference:

```
// Example of a 16-bit application that indicates whether Win32s is
// installed, and the version number if Win32s is loaded and VxD is
// functional.

BOOL FAR PASCAL IsWin32sLoaded(LPSTR szVersion)
{
    BOOL              fWin32sLoaded = FALSE;
    FARPROC           pfnInfo;
    HANDLE            hWin32sys;
    WIN32SINFO        Info;

    hWin32sys = LoadLibrary("W32SYS.DLL");

    if (hWin32sys > HINSTANCE_ERROR) {
        pfnInfo = GetProcAddress(hWin32sys, "GETWIN32SINFO");
        if (pfnInfo) {
            // Win32s version 1.1 is installed
            if (!(*pfnInfo)((LPWIN32SINFO) &Info)) {

                fWin32sLoaded = TRUE;
                wsprintf(szVersion, "%d.%d.%d.0",
                         Info.bMajor, Info.bMinor, Info.wBuildNumber);
            } else
                fWin32sLoaded = FALSE;    // Win32s VxD not loaded.
        } else {
            // Win32s version 1.0 is installed.
            fWin32sLoaded = TRUE;
            lstrcpy( szVersion, "1.0.0.0" );
        }
        FreeLibrary( hWin32sys );
    } else                                // Win32s not installed.
        fWin32sLoaded = FALSE;


    return fWin32sLoaded;
}
```

From a 32-Bit Application

To determine if Win32s is installed, use the function GetVersion(); to then get the version of Win32s use the function, GetVersionEx(). This function fills a specified structure with version information of Win32s on Windows 3.1. The following is an example illustrating the use of this function:

```
// Example of a 32-bit code that determines the operating system installed
// and the version number on all platforms: Windows NT, Windows 95, Win32s.
```

```
typedef BOOL (*LPFNGETVERSIONEX) (LPOSVERSIONINFO);

BOOL IsWin32sLoaded(char *szVersion)
{
  BOOL              fWin32sLoaded = FALSE;
  DWORD             dwGetVer;
  HMODULE           hKernel32;
  OSVERSIONINFO     ver;
  LPFNGETVERSIONEX    lpfnGetVersionEx;

  // First, check if Win32s is installed
  dwGetVer = GetVersion();
  if (!(dwGetVer & 0x80000000))
  {
     //  Windows NT is loaded
     // Note, GetVersion will also return version number on Windows NT

     return;
  }
  else if (LOBYTE(LOWORD(dwVersion))< 4)
  {
    // Win32s is loaded
    fWin32sLoaded = TRUE;
  }
  else {
     // Windows 95 is loaded
     // Note, GetVersion will also return version number on Windows 95

     return;
  }

  // Now, let's find the version number of Win32s
    hKernel32 = GetModuleHandle("Kernel32");
    if (hKernel32)
    {
        lpfnGetVersionEx = (LPFNGETVERSIONEX)GetProcAddress(hKernel32,
"GetVersionExA");
        if (lpfnGetVersionEx)
   {
           // Win32s version 1.15 or later is installed
           ver.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);
           if (!(*lpfnGetVersionEx)((LPOSVERSIONINFO) &ver))
             DisplayError("GetVersionEx");
           else
             wsprintf(szVersion, "%d.%d.%d - %s", ver.dwMajorVersion,
ver.dwMinorVersion,
                      ver.dwBuildNumber, PlatformName(ver.dwPlatformId));
   }
   else
   {
// This failure could mean several things
// 1. On an NT system, it indicates NT version 3.1 because GetVersionEx()
//    is only implemented on NT 3.5.
// 2. On Windows 3.1 system, it means either Win32s version 1.1 or 1.0 is
//    installed. You can distinguish between 1.1 and 1.0 in two ways:
```

```
// a. Get version info from WIN32S16.DLL like File Manager on NT does.
// b. Thunk to 16-bit side and call GetWin32sInfo.


   }
    }

 return (fWin32sLoaded);
}
```

NOTE: In general, 32-bit applications that use Win32s should always ship with the latest version of Win32s. Therefore the detection code above can be greatly simplified if determination of previous versions of Win32s is not needed.

Additional reference words: 3.10 win32s w32s win32 wfw
KBCategory: kbenv
KBSubcategory: W32s

How to Gracefully Fail at Service Start

PSS ID Number: Q115829
Authored 05-Jun-1994 Last modified 03-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1, 3.5, and 3.51

SUMMARY

If an error occurs while your service is running or initializing (SERVICE_START_PENDING) and you need to stop the service process, do the following:

1. Clean up any resources that are being used (threads, memory, and so forth). You should start sending a SERVICE_STOP_PENDING status if the clean up process is lengthy. Be sure to update the Service Control Manager as demonstrated in the Win32 SDK SERVICE sample.

2. Send out a SERVICE_STOPPED status from the last thread to terminate before it calls ExitThread().

3. Set SERVICE_STATUS.dwWin32ExitCode and/or SERVICE_STATUS.dwServiceSpecificExitCode to values that indicate why the service is stopping. If you return a value for the dwServiceSpecificErrorCode field, then the dwWin32ExitCode field should be set to ERROR_SERVICE_SPECIFIC_ERROR.

   The reason for setting these values is that if a service fails its operation, but returns an exit code of 0, the following error message is returned by default:

      Error 2140: An internal Windows NT error occurred

MORE INFORMATION

When the last service in the process has terminated (you may have multiple services in the service process), the StartServiceCtrlDispatcher() call in the main thread returns. The main routine should call ExitProcess() because all of the services have terminated.

REFERENCES

There is a termination sample in the "Win32 Programmer's Reference," in the "Services" overview section (58.2.2), "Writing a ServiceMain Function." This is a simple situation where the service process only consists of one thread. This thread returns when it is ready to terminate, instead of calling ExitThread().

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseService

How to Handle FNERR_BUFFERTOOSMALL in Windows 95

PSS ID Number: Q131462
Authored 12-Jun-1995 Last modified 15-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SUMMARY

When an application uses the Open File common dialog with the OFN_ALLOWMULTISELECT flag, there is a danger that the buffer passed to the common dialog in the OPENFILENAME.lpstrFile field will be too small. In this situation, GetOpenFileName() will return an error value and CommDlgExtendedError() will return FNERR_BUFFERTOOSMALL.

To work around this problem, watch for the Open or OK button to be pressed in the dialog hook; then reallocate the buffer if necessary.

This technique works on Windows version 3.1, Windows NT, and Windows 95, but the implementation details are different when dealing with Windows 95 Explorer-type dialog boxes versus traditional Open and Save common dialog boxes. This article explains how to do it in Windows 95.

MORE INFORMATION

With the introduction of the new common dialogs for Windows 95, a new way of handling the FNERR_BUFFERTOOSMALL error was developed. It is still necessary to watch for the Open button to be pressed and reallocate the buffer if needed, but the way to watch for the OK is much different.

When you install a hook on the Open File common dialog in Windows 95 using the OPENFILENAME.lpfnHook member, the dialog you are hooking is a child of the main Open File dialog. Therefore, to intercept the OK button, you need to subclass the parent dialog. To do this, you can install the hook procedure and watch for the CDN_INITDONE notification. The Open File dialog will send this as part of a WM_NOTIFY message when the initialization for the dialog is complete. For example:

```
LRESULT CALLBACK DialogHook(HWND hwnd, UINT uMsg, WPARAM wParam,
                            LPARAM lParam)
{
    static HWND hwndParentDialog;
    LPOFNOTIFY lpofn;

    switch (uMsg)
    {
        case WM_INITDIALOG:
            // You need to use a copy of the OPENFILENAME struct used to
            // create this dialog. You can store a pointer to the
            // OPENFILENAME struct in the ofn.lCustData so you can retrieve
```

```
            // it here in the lParam. Once you have it, you need to hang on
            // to it. Using window properties provides a good thread safe
            // solution to using a global variable.

            SetProp(hwnd, "OFN", lParam);
            return (0);

        case WM_NOTIFY:
            // The OFNOTIFY struct is passed in the lParam of this message.

            lpofn = (LPOFNOTIFY) lParam;

            switch (lpofn->hdr.code)
            {
                CDN_INITDONE:
                    // Subclass the parent dialog to watch for the OK
                    // button.

                    hwndParentDialog = GetParent(hwnd);
                    g_lpfnDialogProc =
                            (FARPROC) SetWindowLong(hwndParentDialog,
                                                    DWL_DLGPROC,
                                                    OpenFileSubclassProc);
                    break;

            }
            return (0);

        case WM_DESTROY:
            // Need to clean up the subclassing we did on the dialog.
            SetWindowLong(hwndParentDialog, DWL_DLGPROC, g_lpfnDialogProc);

            // Also need to free the property with the OPENFILENAME struct
            RemoveProp(hwnd, "OFN");
            return (0);
    }
    return (0);
}
```

Once the parent dialog is subclassed, the program can watch for the actual Open button. When the program gets the Open button command, it needs to check to see if the buffer originally allocated is large enough to handle all the files selected. The CommDlg_OpenSave_GetFilePath() API will return the length needed. Here is an example of the subclass procedure:

```
LRESULT CALLBACK OpenFileSubclassProc(HWND hwnd, UINT uMsg, WPARAM wParam,
                                      LPARAM lParam)
{
    LPTSTR lpsz;
    WORD   cbLength;

    switch (uMsg)
    {
        case WM_COMMAND:
            switch (LOWORD(wParam))
```

```
            {
                case IDOK:
                    // Need to verify the original buffer size is large
                    // enough to handle the files selected. The
                    // CommDlg_OpenSave_GetFilePath() API will return the
                    // length needed for this buffer.

                    cbLength = CommDlg_OpenSave_GetFilePath(hwnd, NULL, 0);

                    // OFN_BUFFER_SIZE is the size of the buffer originally
                    // used in the OPENFILENAME.lpszFile member.

                    if (OFN_BUFFER_SIZE < cbLength)
                    {
                        // The buffer is too small, so allocate a
                        // new buffer.
                        lpsz = (LPTSTR) HeapAlloc(GetProcessHeap(),
                                                  HEAP_ZERO_MEMORY,
                                                  cbLength);
                        if (lpsz)
                        {
                            // The OFN struct is stored in a property of
                            // the dialog window.

                            lpofn = (LPOPENFILENAME) GetProp(hwnd, "OFN");

                            lpofn->lpstrFile = lpsz;
                            lpofn->nMaxFile  = cbLength;
                        }
                    }

                    // Now let the dialog handle the message normally.
                    break;
            }
            break;
    }

    return (CallWindowProc(g_lpfnDialogProc, hwnd, uMsg, wParam, lParam));
}
```

The dialog should now return without error. Be aware that the buffer allocated in the subclass procedure needs to be freed once the dialog returns.

Finally, this technique only works for 32-bit applications that are using the Explorer-type common dialogs. For 32-bit applications that don't use the OFN_EXPLORER flag, Windows 95 thunks to the 16-bit common dialog and the hook function only gets a copy of the OPENFILENAME structure.

Additional reference words: 4.00
KBCategory: kbui kbprg kbcode
KBSubcategory: UsrCmnDlg

How to Ignore WM_MOUSEACTIVATE Message for an MDI Window

PSS ID Number: Q62068
Authored 18-May-1990 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

In order to make an MDI window to become active and have the caret be in the same position as when the window was last active you need to process the WM_MOUSEACTIVATE message and return MA_ACTIVATEANDEAT for the first time. Therefore, you need to set a Boolean flag in the WM_MDIACTIVATE message so that the return is set only once. The sample code below can be used to modify the MULTIPAD sample application. Also, the following is documentation on MA_ACTIVATE* messages, taken from the Windows 3.0 final SDK README.WRI file:

WM_MOUSEACTIVATE

Return Value — The return value specifies whether the window should be activated and whether the mouse event should be discarded. It must be one of the following values:

Value	Meaning
MA_ACTIVATE	Activate the window.
MA_NOACTIVATE	Do not activate the window.
MA_ACTIVATEANDEAT	Activate the window and discard the mouse event.

SAMPLE CODE

```
/* --- multipad.c  MPMDIWndProc section --- */

 case WM_MOUSEACTIVATE:   // added
     if (bEatMessage) {
        bEatMessage = FALSE;
        return (LONG)MA_ACTIVATEANDEAT ;
     }
     /* else break */
     break;

 case WM_MDIACTIVATE:
     /* If we're activating this child, remember it */
     if (wParam){
       hwndActive     = hwnd;
       hwndActiveEdit = (HWND)GetWindowWord (hwnd, GWW_HWNDEDIT);
       bEatMessage = TRUE;      // added
     }
```

```
      else{
            hwndActive     = NULL;
            hwndActiveEdit = NULL;
      }
      break;
```

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMdi

How to Implement Context-Sensitive Help in Windows 95 Dialogs

PSS ID Number: Q125670
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

In Windows versions 3.x, applications implement context-sensitive help for dialog boxes by installing either a message filter hook, or a task-specific keyboard hook that monitors the WM_KEYDOWN message and responds to F1 key presses.

Windows 95 makes it easier because it provides a new WM_HELP message that gets sent each time the user presses the F1 key, giving the application a chance to bring up help information on the control that has the keyboard focus or on the dialog box itself. This new WM_HELP message is not limited to dialog boxes alone, as it gets sent to any window that has keyboard focus or to the currently active window.

MORE INFORMATION

Windows 95 also provides a new dialog style, DS_CONTEXTHELP that adds a question mark button to the dialog box's caption bar. This button, when clicked, changes the cursor to a question mark with a pointer. When the user clicks any control in the dialog box, Windows 95 sends a WM_HELP message for that control. The dialog procedure should process the WM_HELP message as follows:

```
   // Define an array of dword pairs,
   // where the first of each pair is the control ID,
   // and the second is the context ID for a help topic,
   // which is used in the help file.
   static const DWORD aMenuHelpIDs[] =
   {
      edt1, IDH_EDT1 ,
      lst1, IDH_LST1 ,
      lst2, IDH_LST2 ,
      0,    0
   };

   case WM_HELP:
   {
      LPHELPINFO lphi;

      lphi = (LPHELPINFO)lparam;
      if (lphi->iContextType == HELPINFO_WINDOW)   // must be for a control
      {
         WinHelp (lphi->hItemHandle,
```

```
                  "GEN32.HLP",
                  HELP_WM_HELP,
                  (DWORD)(LPVOID)aMenuHelpIDs);
      }
      return TRUE;
   }
```

Calling WinHelp() with the HELP_WM_HELP parameter as demonstrated above displays the help topic in a pop-up window.

In addition to the WM_HELP message, Windows 95 provides a new WM_CONTEXTMENU message that gets sent each time the user right-clicks a window. Typically this message is processed by displaying a context menu using the TrackPopupMenu() function. However, this message can be processed to bring up help information by calling the WinHelp() function to display the help topic in a pop-up window, as in this example:

```
   case WM_CONTEXTMENU:
   {
      WinHelp ((HWND)wparam,
               "GEN32.HLP",
                HELP_CONTEXTMENU,
               (DWORD)(LPVOID)aSampleMenuHelpIDs);
      return TRUE;
   }
```

NOTE: Look at the third parameter (HELP_CONTEXTMENU) passed to WinHelp() this time. This causes a pop-up menu to come up that displays "What's This?" text. It then displays the help topic in a pop-up window when the menu item is selected.

Additional reference words: 4.00
KBCategory: kbprg kbcode
KBSubcategory: UsrDlgs

How to Increase Windows NT System and Desktop Heap Sizes

PSS ID Number: Q125752
Authored 05-Feb-1995 Last modified 06-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 3.5

SUMMARY

Sometimes, it may be necessary to increase the amount of memory that Windows NT will make available for the system and desktop heaps. This can be accomplished by editing an entry in the registration database. System heap items are things like desktops and one-time-allocated items like system metrics. The items that come out of the desktop heap are items such as windows, menus, hook structures, queues, and some thread information.

MORE INFORMATION

The entry to be edited is under:

```
HKEY_LOCAL_MACHINE\
   System\
   CurrentControlSet\
   Control\
   Session Manager\
   SubSystems\
   Windows
```

Under this entry, you will find a string similar to the following (the slash (/) is a line continuation charater):

```
%SystemRoot%\system32\csrss.exe /
   ObjectDirectory=\Windows /
   SharedSection=1024,512 /
   Windows=On /
   SubSystemType=Windows /
   ServerDll=basesrv,1 /
   ServerDll=winsrv:GdiServerDllInitialization,4 /
   ServerDll=winsrv:UserServerDllInitialization,3 /
   ServerDll=winsrv:ConServerDllInitialization,2 /
   ProfileControl=Off /
   MaxRequestThreads=16
```

By changing the SharedSection values, you can affect the heap sizes. The first number (1024 as shown above) is the maximum size of the system wide heap in kilobytes. The second number (512 as shown above) is the maximum size of the per desktop heap in kilobytes. A desktop value of 512K can support approximately 2,500 windows.

The memory you allocate needs to be backed up by paging space. It should

not have much effect on performance if you create the same number of items with different heap sizes. The main effect is overhead in heap management and initialization.

Additional reference words: 3.50
KBCategory: kbprg
KBSubcategory: UsrWndw

How to Keep a Window Iconic

PSS ID Number: Q66244
Authored 16-Oct-1990 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

Normally, when an application's main window is being represented by an icon ("iconic"), you can restore it to an open window by double-clicking the icon or by choosing the Restore option from the System menu.

Opening the window can be prevented by placing code into the application that processes the WM_QUERYOPEN message by returning FALSE.

If it is necessary to perform processing before the iconic window is opened, the processing should be done in response to the WM_QUERYOPEN message. After processing is complete the program can return TRUE and the window will be opened.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrWndw

How to Keep an MDI Window Always on Top

PSS ID Number: Q108315
Authored 08-Dec-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

When creating a multiple document interface (MDI) window, there are no styles available to have the new window stay on top of the other MDI windows. Alternatively, two methods are available to achieve this functionality:

- Process the WM_WINDOWPOSCHANGED message and call SetWindowPos() to change the Z-order of the window.

- Install a timer for the MDI windows and reset the Z-order of the window when processing the WM_TIMER message.

MORE INFORMATION

MDICREATESTRUCT has the field "style", which can be set with the styles for the new MDI window. Extended styles, such as WS_EX_TOPMOST, are not available in MDI windows. This field of MDICREATESTRUCT is passed to CreateWindowEx() in the dwStyle parameter. The dwExStyle field is set to 0L. The two methods shown below cannot be used at the same time in the same application.

Method 1: Process the WM_WINDOWPOSCHANGED message and call SetWindowPos() to change the Z-order of the window.

Sample Code

```
LRESULT CALLBACK MdiWndProc (HWND hWnd, UINT message, WPARAM wParam,
                    LPARAM lParam)
{
    static HWND hWndAlwaysOnTop = 0;
    switch (message)
     {
     case WM_CREATE :
         if (!hWndAlwaysOnTop)
          {
          SetWindowText (hWnd, "Always On Top Window");
          hWndAlwaysOnTop = hWnd;
          }
            break;
     case WM_WINDOWPOSCHANGED :
```

```
        if (hWndAlwaysOnTop)
         {
         WINDOWPOS FAR* pWP = (WINDOWPOS FAR*)lParam;
         if (pWP->hwnd != hWndAlwaysOnTop)
             SetWindowPos (hWndAlwaysOnTop, HWND_TOP, 0, 0, 0, 0,
                     SWP_NOACTIVATE | SWP_NOMOVE | SWP_NOSIZE);
         }
        break;
     //
     // Other Messages to process here.
     //
     case WM_CLOSE :
        if (hWndAlwaysOnTop == hWnd)
         hWndAlwaysOnTop = NULL;
     default :
        return DefMDIChildProc (hWnd, message, wParam, lParam);
     }
    return 0L;
}
```

Method 2: Install a timer for the MDI windows and reset the Z-order of the window when processing the WM_TIMER message.

Sample Code

```
LRESULT CALLBACK MdiWndProc (HWND hWnd, UINT message, WPARAM wParam,
                    LPARAM lParam)
{
    static HWND hWndAlwaysOnTop = 0;
    switch (message)
     {
     case WM_CREATE :
        SetTimer (hWnd, 1, 200, NULL);
        if (!hWndAlwaysOnTop)
         {
         SetWindowText (hWnd, "Always On Top Window");
         hWndAlwaysOnTop = hWnd;
         }
           break;
     case WM_TIMER :
        if (hWndAlwaysOnTop)
         {
         SetWindowPos (hWndAlwaysOnTop, HWND_TOP, 0, 0, 0, 0,
                    SWP_NOACTIVATE | SWP_NOMOVE | SWP_NOSIZE);
         }
        break;
     case WM_DESTROY:
        KillTimer (hWnd, 1) ;
        break;
     //
     // Other Messages to process here.
     //
     case WM_CLOSE :
        if (hWndAlwaysOnTop == hWnd)
```

```
          hWndAlwaysOnTop = NULL;
     default :
         return DefMDIChildProc (hWnd, message, wParam, lParam);
     }
    return 0L;
}
```

For additional information on changing the Z-order of child pop-up windows, please see the following article(s) in the Microsoft Knowledge Base:

```
   ARTICLE-ID: Q66943
   TITLE     : Determining the Topmost Pop-Up Window
```

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrWndw

How to Look Up a User's Full Name

PSS ID Number: Q119670
Authored 20-Aug-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

Windows NT workstations can be organized into a domain, which is a collection of computers on a Windows NT Advanced Server network. The domain administrator maintains centralized user and group account information.

MORE INFORMATION

To find the full name of a user if you have the user name and domain name:

1. Convert the user name and domain name to Unicode, if they are not already Unicode strings. This is a requirement of the ported LAN Manager APIs that are used in the following steps.

2. Look up the name of the domain controller (DC) for the domain name by calling NetServerEnum().

3. Look up the user name by calling NetUserGetInfo()

4. Convert the full user name to ANSI, unless the program is expecting to work with Unicode strings.

The sample code below is a function that takes a user name and a domain name as the first two arguments and returns the user's full name in the third argument.

For information on how to get the current user and domain, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID: Q111544
TITLE : Looking Up the Current User and Domain

Sample Code

```
#include <windows.h>
#include <lm.h>
#include <stdio.h>

/*****************************************************************\
* Function: GetFullName( char *UserName, char *Domain, char *dest ); *
*                                                                 *
```

```
   * Parameters:                                                        *
   *    UserName: the user name                                        *
   *    Domain  : the domain to which the user belongs                 *
   *    dest    : receives the user's full name                        *
   *                                                                   *

   BOOL GetFullName(char *UserName, char *Domain, char *dest)
   {
      WCHAR  wszUserName[256];           // Unicode user name
      WCHAR  wszDomain[256];
      LPBYTE ComputerName;

      struct _SERVER_INFO_100 *si100;   // Server structure
      struct _USER_INFO_2 *ui;          // User structure

      // Convert ASCII user name and domain to Unicode.

      MultiByteToWideChar( CP_ACP, 0, UserName,
         strlen(UserName)+1, wszUserName, sizeof(wszUserName) );
      MultiByteToWideChar( CP_ACP, 0, Domain,
         strlen(Domain)+1, wszDomain, sizeof(wszDomain) );

      // Get the computer name of a DC for the specified domain.

      NetGetDCName( NULL, wszDomain, &ComputerName );

      // Look up the user on the DC.

      if(NetUserGetInfo( (LPWSTR) ComputerName,
         (LPWSTR) &wszUserName, 2, (LPBYTE *) &ui))
      {
         printf( "Error getting user information.\n" );
         return( FALSE );
      }

      // Convert the Unicode full name to ASCII.

      WideCharToMultiByte( CP_ACP, 0, ui->usri2_full_name,
         -1, dest, 256, NULL, NULL );

      return( TRUE );
   }
```

Additional reference words: 3.10 3.50
KBCategory: kbnetwork kbprg
KBSubcategory: NtwkLmapi

How to Make an Application Display Real Units of Measurement

PSS ID Number: Q127152
Authored 13-Mar-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

Sometimes you need an application to display things in terms of a real unit of measurement such as an inch or millimeter. When dealing with a printer, resolution is usually given in dots per inch (DPI), which makes it easy to convert pixels to real inches. However, on a video display, resolution is given only in pixels. A given video mode will be some X pixels wide with no information as to the real dimensions of the display area.

Because there is no way to programmatically determine the real dimensions of the viewable area on a video display, it is impossible for a program to determine real output dimensions. Two manual methods for determining real output dimensions are given in this article.

MORE INFORMATION

When output is destined for a printer, the application can call GetDeviceCaps() using LOGPIXELSX and LOGPIXELSY to determine dots per real inch. However, for a video display, LOGPIXELSX and LOGPIXELSY are defined by the video driver and may vary wildly. These numbers define a logical inch, which is almost never equal to a real inch.

Applications that need to output real sizes to the video display can use one of the following two methods for determining output size:

1. The application can ask the user what size monitor is attached. Using this value, an application can approximate the actual viewable area, and given the resolution of the output (GetDeviceCaps, HORZRES, VERTRES), the application can approximate real inches. This solution gives only an approximation of a real inch. Several factors can introduce errors into this approximation including the size adjustments on digital monitors.

2. The application can ask the user to hold a measuring device to the screen and measure a given line. This is the only way to guarantee that output on a video display is exactly the expected size, and recalibration would be necessary after any adjustment to the monitor.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbgraphic
KBSubcategory: GdiDisplay

How to Minimize Memory Allocations for New TreeView Control

PSS ID Number: Q130697
Authored 25-May-1995 Last modified 30-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.51 and 4.0

SUMMARY

You can use the new TreeView Common Control to display a hierarchical list of items. This new control is available for Win32-based applications running under Windows NT or Windows 95. Applications typically use this control to display a list of items like directories or files on a given drive. Each node in a TreeView control allocates about 40 bytes. If the TreeView control displays a lot of items, applications can easily consume large amounts of memory, which slows down other applications.

Below are some techniques applications can use to minimize memory allocations for TreeView controls.

MORE INFORMATION

TreeView controls maintain internal data structures for every node added to the control. This data structure along with image lists associated with items and text strings for items can drain the physical memory available on the system.

Applications that need to display thousands of items or nodes in the TreeView control can be more proficient about memory allocations by doing the following:

1. When inserting an item into a TreeView control, ensure that the pszText member of the TV_ITEM is not the actual string that needs to be displayed, but is the value LPSTR_TEXTCALLBACK. If a string pointer is passed, the control stores that string internally by allocating memory for it. When this flag is specified, the parent window of the control is responsible for storing the name (string). The string in most cases can be generated dynamically. In this case, the TreeView control sends the parent window a TVN_GETDISPINFO notification message when it needs the item text for displaying, sorting, or editing and sends a TVN_SETDISPINFO notification message when the item text changes.

2. Fill the TreeView nodes on demand. One way to really minimize memory usage in a TreeView control is to fill in only the visible nodes. The TV_ITEM struct's cChildren member can be put to good use for this purpose. This is used as a flag to indicate whether the item has associated child items. It is 1 if the item has one or more child items; otherwise, it is 0 (zero). When inserting visible items into the TreeView control, set this cChildren member to 1 if that node will have

child items under it. Do not insert the child items. When the user clicks the node, the application receives a TVN_ITEMEXPANDING with NM_TREEVIEW.action set to TVE_EXPAND. Insert the child items at that point. Then when the user clicks the same node again (to collapse the node), the applciation receives a TVN_ITEMEXPANDED with NM_TREEVIEW.action set to TVE_COLLAPSE. At that time, collapse the node and all its child items by calling TreeView_Expand (hWndTrevview, hItem, TVE_COLLAPSE|TVE_COLLAPSERESET). This frees up the memory used up by all the children or child items.

3. If the application uses different icons for each child item in the TreeView control, specify the I_IMAGECALLBACK value for the iImage and iSeletedImage members of the TV_ITEM Structure. This way, the control doesn't have to store these images for every child item - thereby reducing the memory requirements for the control as a whole.

Additional reference words: 4.00 usage user styles
KBCategory: kbprg
KBSubcategory: TreeView

How to Modify Executable Code in Memory

PSS ID Number: Q127904
Authored 21-Mar-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 3.5, 3.51, and 4.0

Follow the steps in this article to create self-modifying code; that is, to modify code pages while they are in memory and execute them there.

NOTE: Self-modifying code is not advised, but there are cases where you may wish to use it.

Step-by-Step Example
---

1. Call VirtualProtect() on the code pages you want to modify, with the PAGE_WRITECOPY protection.

2. Modify the code pages.

3. Call VirtualProtect() on the modified code pages, with the PAGE_EXECUTE protection.

4. Call FlushInstructionCache().

All four steps are required. The reason for calling FlushInstructionCache() is to make sure that your changes are executed. As processors get faster, the instruction caches on the chips get larger. This allows more out of order prefetching to be done. If you modify your code, but do not call FlushInstructionCache(), the previous instructions may already be in the cache and your changes will not be executed.

Additional reference words: 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseMm

How to Modify the Width of the Drop Down List in a Combo Box

PSS ID Number: Q131845
Authored 21-Jun-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32s version 1.2
- Microsoft Win32 Software Development KIT (SDK) versions 3.5 and 4.0

SUMMARY

The Windows combo box contains a list box (of the ComboLBox class) within it. In the standard combo box, this list box has exactly the same width as the combo box. However, you can make the width of the list box wider or narrower than the width of the combo box. You may have seen combo box lists like this in Microsoft Word and Microsoft Excel. This article shows by example how to subclass a standard combo box class to achieve this functionality.

MORE INFORMATION

The combo box in Windows is actually a combination of two or more controls; that's why it's called a "combo" box. For more information about the parts of a combo box and how they relate to each other, please see the following article in the Microsoft Knowledge Base:

```
ARTICLE-ID: Q65881
TITLE     : The Parts of a Windows Combo Box and How They Relate
```

To make the combo box list wider or narrower, you need the handle of the list box control within the combo box. This task is difficult because the list box is actually a child of the desktop window (for CBS_DROPDOWN and CBS_DROPDOWNLIST styles). If it were a child of the ComboBox control, dropping down the list box would clip it to the parent, and it wouldn't display.

A combo box receives WM_CTLCOLOR messages for its component controls when they need to be painted. This allows the combo box to specify a color for these controls. The HIWORD of the lParam in this message is the type of the control. In case of the combo box, Windows sends it a WM_CTLCOLOR message with the HIWORD set to CTLCOLOR_LISTBOX when the list box control needs to be painted. The LOWORD of the lParam contains the handle of the list box control.

In 32-bit Windows, the WM_CTLCOLOR message has been replaced with multiple messages, one for each type of control (WM_CTLCOLORBTN). For more information about this, please see the following article in the Microsoft Knowledge Base:

```
ARTICLE-ID: Q81707
```

TITLE : WM_CTLCOLOR Processing for Combo Boxes of All Styles

Once you obtain the handle to the list box control window, you can resize the control by using the MoveWindow API.

The following code sample demonstrates how to do this. This sample assumes that you have placed the combo box control in a dialog box.

Sample Code

```
// Global declarations.

LRESULT CALLBACK NewComboProc (HWND hWnd,   UINT message,   WPARAM
   wParam, LPARAM lParam ); // prototype for the combo box subclass proc

HANDLE hInst;                       // Current app instance
BOOL bFirst;                        // a flag

// Dialog procedure for the dialog containing the combo box.

BOOL __export CALLBACK DialogProc(HWND hDlg, UINT message, WPARAM wParam,
            LPARAM lParam)
{
    FARPROC lpfnNewComboProc;

    switch (message)
    {
    case WM_INITDIALOG:

   bFirst = TRUE;       // set flag here - see below for usage

   // subclass the combo box

   lpfnOldComboProc = (FARPROC ) SetWindowLong (
               GetDlgItem ( hDlg, IDC_COMBO1 ),
               GWL_WNDPROC,
               (LONG)NewComboProc );
   break;

     case WM_DESTROY:
   (FARPROC ) SetWindowLong (    GetDlgItem ( hDlg, IDC_COMBO1 ),
               GWL_WNDPROC,
               (LONG)lpfnOldComboProc );
   break;
     default:
        break;
     }

     return FALSE;

} // end dialog proc
```

```
// Combobox subclass proc.

LRESULT CALLBACK NewComboProc (HWND hWnd,   UINT message,   WPARAM
            wParam, LPARAM lParam );

{
   static HWND hwndList;
   static RECT rectList;

#ifdef WIN16
   if (  WM_CTLCOLOR  == message)  // combo controls are to be painted.
#else
   if ( WM_CTLCOLORLISTBOX == message ) // 32 bits has new message.
#endif
   {
      // is this message for the list box control in the combo?
#ifdef WIN16
      if ( CTLCOLOR_LISTBOX==HIWORD (lParam )   ) // need only for 16 bits
      {
#endif
         // Do only the very first time, get the list
         // box handle and the list box rectangle.
         // Note the use of GetWindowRect, as the parent
         // of the list box is the desktop window

         if ( bFirst )
         {
#ifdef WIN16
            hwndList = LOWORD (lParam );
#else
            hwndList = (HWND) lParam ;     // HWND is 32 bits
#endif
            GetWindowRect ( hwndList, &rectList );
                        bFirst = FALSE;
         }

         // Resize listbox window cx by 50 ( use your size here )

         MoveWindow ( hwndList, rectList.left, rectList.top,
         ( rectList.right - rectList.left + 50 ),
         rectList.bottom - rectList.top, TRUE );
#ifdef WIN16
      }
#endif
   }

   // Call original combo box procedure to handle other combo messages.

   return CallWindowProc ( lpfnOldComboProc, hWnd, message,
wParam, lParam  );
}
```

Additional reference words: 1.20 3.10 3.50 4.00 95
KBCategory: kbprg kbui kbcode
KBSubcategory: usrctl

How to Obtain a Handle to Any Process with SeDebugPrivilege

PSS ID Number: Q131065
Authored 02-Jun-1995 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1, 3.5, and 3.51

SUMMARY

In Windows NT, you can retrieve a handle to any process in the system by enabling the SeDebugPrivilege in the calling process. The calling process can then call the OpenProcess() Win32 API to obtain a handle with PROCESS_ALL_ACCESS.

MORE INFORMATION

This functionality is provided for system-level debugging purposes. For debugging non-system processes, it is not necessary to grant or enable this privilege.

This privilege allows the caller all access to the process, including the ability to call TerminateProcess(), CreateRemoteThread(), and other potentially dangerous Win32 APIs on the target process.

Take great care when granting SeDebugPrivilege to users or groups.

Sample Code

The following source code illustrates how to obtain SeDebugPrivilege in order to get a handle to a process with PROCESS_ALL_ACCESS. The sample code then calls TerminateProcess on the resultant process handle.

```
/*++

The SeDebugPrivilege allows you to open any process for debugging purposes.
After enabling the privilege, you can open a target process by using
OpenProcess() with PROCESS_ALL_ACCESS.

By default, this privilege is granted only to SYSTEM and the local
Administrators group.

User Manager | Policies | User Rights | Show Advanced User Rights | Debug
Programs can be used to grant or revoke this privilege to arbitrary users
or groups.

WARNING: This privilege allows all access to a process. A malevolent user
could open a system process, create a remote thread in the system process,
```

```
and execute code in the system security context. Great care must be used
when giving out this privilege

--*/

#define RTN_OK 0
#define RTN_USAGE 1
#define RTN_ERROR 13

#include <windows.h>
#include <stdio.h>

BOOL SetPrivilege(
    HANDLE hToken,          // token handle
    LPCTSTR Privilege,      // Privilege to enable/disable
    BOOL bEnablePrivilege   // TRUE to enable.  FALSE to disable
    );

void DisplayError(LPTSTR szAPI);

int main(int argc, char *argv[])
{
    HANDLE hProcess;
    HANDLE hToken;
    int dwRetVal=RTN_OK; // assume success from main()

    // show correct usage for kill
    if (argc != 2)
    {
        fprintf(stderr,"Usage: %s [ProcessId]\n", argv[0]);
        return RTN_USAGE;
    }

    if(!OpenProcessToken(
            GetCurrentProcess(),
            TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY,
            &hToken
            )) return RTN_ERROR;

    // enable SeDebugPrivilege
    if(!SetPrivilege(hToken, SE_DEBUG_NAME, TRUE))
    {
        DisplayError("SetPrivilege");

        // close token handle
        CloseHandle(hToken);

        // indicate failure
        return RTN_ERROR;
    }

   // open the process
    if((hProcess = OpenProcess(
            PROCESS_ALL_ACCESS,
            FALSE,
```

```
            atoi(argv[1]) // PID from commandline
            )) == NULL)
    {
        DisplayError("OpenProcess");
        return RTN_ERROR;
    }

    // disable SeDebugPrivilege
    SetPrivilege(hToken, SE_DEBUG_NAME, FALSE);

    if(!TerminateProcess(hProcess, 0xffffffff))
    {
        DisplayError("TerminateProcess");
        dwRetVal=RTN_ERROR;
    }

    // close handles
    CloseHandle(hToken);
    CloseHandle(hProcess);

    return dwRetVal;
}

BOOL SetPrivilege(
    HANDLE hToken,          // token handle
    LPCTSTR Privilege,      // Privilege to enable/disable
    BOOL bEnablePrivilege   // TRUE to enable.  FALSE to disable
    )
{
    TOKEN_PRIVILEGES tp;
    LUID luid;
    TOKEN_PRIVILEGES tpPrevious;
    DWORD cbPrevious=sizeof(TOKEN_PRIVILEGES);

    if(!LookupPrivilegeValue( NULL, Privilege, &luid )) return FALSE;

    //
    // first pass.  get current privilege setting
    //
    tp.PrivilegeCount           = 1;
    tp.Privileges[0].Luid       = luid;
    tp.Privileges[0].Attributes = 0;

    AdjustTokenPrivileges(
            hToken,
            FALSE,
            &tp,
            sizeof(TOKEN_PRIVILEGES),
            &tpPrevious,
            &cbPrevious
            );

    if (GetLastError() != ERROR_SUCCESS) return FALSE;

    //
```

```
    // second pass.  set privilege based on previous setting
    //
    tpPrevious.PrivilegeCount       = 1;
    tpPrevious.Privileges[0].Luid   = luid;

    if(bEnablePrivilege) {
        tpPrevious.Privileges[0].Attributes |= (SE_PRIVILEGE_ENABLED);
    }
    else {
        tpPrevious.Privileges[0].Attributes ^= (SE_PRIVILEGE_ENABLED &
            tpPrevious.Privileges[0].Attributes);
    }

    AdjustTokenPrivileges(
            hToken,
            FALSE,
            &tpPrevious,
            cbPrevious,
            NULL,
            NULL
            );

    if (GetLastError() != ERROR_SUCCESS) return FALSE;

    return TRUE;
}

void DisplayError(
    LPTSTR szAPI    // pointer to failed API name
    )
{
    LPTSTR MessageBuffer;
    DWORD dwBufferLength;

    fprintf(stderr,"%s() error!\n", szAPI);

    if(dwBufferLength=FormatMessage(
                FORMAT_MESSAGE_ALLOCATE_BUFFER |
                FORMAT_MESSAGE_FROM_SYSTEM,
                NULL,
                GetLastError(),
                GetSystemDefaultLangID(),
                (LPTSTR) &MessageBuffer,
                0,
                NULL
                ))
    {
        DWORD dwBytesWritten;

        //
        // Output message string on stderr
        //
        WriteFile(
                GetStdHandle(STD_ERROR_HANDLE),
                MessageBuffer,
```

```
                dwBufferLength,
                &dwBytesWritten,
                NULL
                );

        //
        // free the buffer allocated by the system
        //
        LocalFree(MessageBuffer);
    }
}
```

Additional reference words: 3.10 3.50 3.51 OpenProcess TerminateProcess
KBCategory: kbprg kbcode
KBSubcategory: BseSecurity BseMisc CodeSam

How to Obtain Filename and Path from a Shell Link or Shortcut

PSS ID Number: Q130698
Authored 25-May-1995 Last modified 30-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.51 and 4.0

SUMMARY

The new shell link under Windows 95 (also available for Windows NT after Windows 95 ships) provides applications and users a way to create shortcuts or links to objects in the shells namespace. Some applications need to get the filename and path, given a link or shortcut. The IShellLink OLE Interface implemented by the shell can be used to obtain this information, among other things.

MORE INFORMATION

A shell link allows the user or an application to access an oject from anywhere in the namespace. Links or shortcuts to objects are stored as binary files. These files contain information such as the path to the object, working directory, the path of the icon used to display the object, the description string, and so on.

Given a link or shortcut, applications can use the IShellLink interface and its functions to obtain all the pertinent information about that object. The IShellLink interface supports fucntions such as GetPath(), GetDescription(), Resolve(), GetWorkingDirectory(), and so on.

Code Sample

The following code shows how to obtain the filename or path and description of a given link file.

```
// GetLinkInfo() fills the filename and path buffer
// with relevant information
// hWnd         - calling app's window handle.
//
// lpszLinkName - name of the link file passed into the function.
//
// lpszPath     - the buffer that will receive the filepath name.
//

HRESULT GetLinkInfo( HWND hWnd,
           LPCTSTR lpszLinkName,
           LPSTR lpszPath,
                  LPSTR szDescription)
{
```

```
   HRESULT hres;
   IShellLink *psl;
   WIN32_FIND_DATA wfd;

// Assume Failure to start with:
   *lpszPath = 0;
   *lpszDescription = 0;

// Call CoCreateInstance to obtain the IShellLink
// Interface pointer. This call fails if
// CoInitialize is not called, so it is assumed that
// CoInitialize has been called.

   hres = CoCreateInstance( CLSID_ShellLink,
               NULL,
               CLSCTX_INPROC_SERVER,
                           IID_IShellLink,
                                 (LPVOID *)&psl );
   if ( SUCCEEDED( hres ) )
   {
      IPersistFile *ppf;

      bRetVal = TRUE;

// The IShellLink Interface supports the IPersistFile
// interface. Get an interface pointer to it.
      hres = psl->lpVtbl->QueryInterface(psl,
                  IID_IPersistFile,
                  (LPVOID *)&ppf );
      if ( SUCCEEDED( hres ) )
      {
         WORD wsz[MAX_PATH];

//Convert the given link name string to wide character string.
         MultiByteToWideChar( CP_ACP, 0,
                  lpszLinkName,
                  -1, wsz, MAX_PATH );
//Load the file.
         hres = ppf->lpVtbl->Load(ppf, wsz, STGM_READ );
         if ( SUCCEEDED( hres ) )
         {
// Resolve the link by calling the Resolve() interface function.
            hres = psl->lpVtbl->Resolve(psl,  hWnd,
                    SLR_ANY_MATCH |
                    SLR_NO_UI);
            if ( SUCCEEDED( hres ) )
            {
               hres = psl->lpVtbl->GetPath( psl,    lpszPath,
                  MAX_PATH,
                       (WIN32_FIND_DATA*)&wfd,
                                             SLGP_SHORTPATH );
        if(!SUCCEEDED(hres))
      return FALSE;
```

```
                hres = psl->lpVtbl->Get(Description(psl,
                     lpszDescription,
                        MAX_PATH );

          if(!SUCCEEDED(hres))
        return FALSE;

              }
           }
           ppf->lpVtbl->Release();
        }
        psl->lpVtbl->Release();
     }
     return hres;
}
```

Ensure that the interface pointers are released when the application is finished using them. For a list of other functions that the IShellLink Interface supports, please see the documentation on the IShellLink interface.

NOTE: If applications use C++ instead of C, the interface pointer (psl, ppf) and the lpVtbl variables are implicit.

Additional reference words: 4.00
KBCategory: kbprg kbcode kbole
KBSubcategory: IShellLink

How to Obtain Fonts, ToolTips, and Other Non-Client Metrics

PSS ID Number: Q130764
Authored 25-May-1995 Last modified 22-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SUMMARY

The SystemParametersInfo() API under Windows 95 has been expanded to include a new set of FLAGS to set or get system-wide parameters. One such flag is the SPI_GET/SETNONCLIENTMETRICS flag and the NONCLIENTMETRICS structure. This flag and the related structure when used with the SystemsParametersInfo() API can provide applications with a plethora of information on the non-client metrics system wide.

MORE INFORMATION

Windows 95 provides users with the ability to change the fonts used (displayed) by menus and message boxes and change the height of caption bars of windows by simply changing the settings in the Appearance property sheet. They need only right click the desktop to bring up the Display properties dialog box; then they can choose the Appearance page. This was not possible under Windows version 3.1.

Windows 95 applications can programatically change these features with the help of the SystemParametersInfo() function and the NONCLIENTMETRICS structure. Windows 95 applications should not randomly change these system settings unless absolutely necessary because system-wide changes occur.

To obtain the fonts used by message boxes, menus, status bars, and captions, applications can call SystemParametersInfo() with the SPI_GETNONCLIENTMETRICS flag, passing the address of the NONCLIENTMETRICS structure as the third parameter. The system then fills this structure with all sorts of information. The lfMessageFont, lfMenuFont, lfStatusFont members of the NONCLIENTMETRICS structure have the font information.

Similarly, applications can change the font used by menus, message boxes, status bars, and captions by calling SystemParametersInfo() with the SPI_SETNONCLIENTMETRICS flag. When this flag is spcified, a NONCLIENTMETRICS structure is filled with the appropriate values if its address is passed in as the third parameter. Once again, the changes made this way are reflected system wide, so application designers should use this flag sparingly.

Additional reference words: 4.00 user controls styles
KBCategory: kbprg
KBSubcategory: UsrSys

How to Obtain Icon Information from an .EXE in Windows 95

PSS ID Number: Q131500
Authored 12-Jun-1995 Last modified 15-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SUMMARY

SHGetFileInfo is a new API that Windows 95 provides to allow an application to extract icon information from a particular file. With the introduction of large (32x32) and small (16x16) icons in Windows 95, SHGetFileInfo provides this information as well by filling in the appropriate members of the SHFILEINFO structure.

MORE INFORMATION

In Windows version 3.1, an application can use the ExtractIcon() to retrieve the handle of an icon associated with a specified executable file, dynamic link library, or icon file.

Windows 95 provides a new function, SHGetFileInfo(), which, among other things, provides this icon information when using the SHGFI_ICON flag. When the function returns, the handle to the icon is returned in the hIcon member of the SHFILEINFO structure, and the index of the icon within the system image list is returned in the iIcon member of the same structure.

The code below demonstrates how to retrieve the Gen32 sample's icon and associates that icon with a dialog box's button:

```
   HICON  hGen32Icon;
   SHFILEINFO  shfi;

   if (SHGetFileInfo ((LPCSTR)"C:\\MySamplesDir\\Gen32\\Gen32.Exe",
                      0,
                      &shfi,
                      sizeof (SHFILEINFO),
                      SHGFI_ICON))
   {
      hGen32Icon = shfi.hIcon;

      // Note that this button has been defined in the .RC file
      // to be of BS_ICON style
      SendDlgItemMessage (hDlg,
                          IDC_BUTTON,
                          BM_SETIMAGE,
                          (WPARAM)IMAGE_ICON,
                          (LPARAM)(HICON)hGen32Icon);
   }
   else
```

```
   {
   // SHGetFileInfo failed...
   }
```

Windows 95 also introduces the concept of large and small icons associated with applications where the large icon is displayed when the application is minimized and the small icon is displayed in the upper-left corner of the application. This small icon, when clicked, drops down the application's system menu.

SHGetFileInfo() provides the file's large and small icon information as well, using the SHGFI_LARGEICON and SHGFI_SMALLICON flags respectively. SHGFI_LARGEICON returns the handle of the system image list containing the large icon images, whereas SHGFI_SMALLICON returns that of the system image list containing the small icon images. These flags, when OR'ed with the SHGFI_SYSICONINDEX flag, return the icon index within the appropriate system image list in the iIcon member of the SHFILEINFO struct.

The code sample below demonstrates how to retrieve the small icon associated with the same GEN32 sample.

```
   HICON  hGen32Icon;
   HIMAGELIST hSysImageList;
   SHFILEINFO  shfi;

   hSysImageList = SHGetFileInfo
((LPCSTR)"C:\\MySamplesDir\\Gen32\\Gen32.Exe",
                                0,
                                &shfi,
                                sizeof (SHFILEINFO),
                                SHGFI_SYSICONINDEX | SHGFI_SMALLICON);
   if (hSysImageList)
   {
   hGen32Icon = ImageList_GetIcon (hSysImageList,
                                     shfi.iIcon,
                                     ILD_NORMAL);
   }
   else
   {
   // SHGetFileInfo failed...
   }
```

Before closing, the application must call DestroyIcon() to free system resources associated with the icon returned by ImageList_GetIcon().

NOTE: The SHGetFileInfo() API will be supported in the next release of Windows NT that supports the new shell interface very similar to Windows 95. This API is not supported in Windows NT version 3.51.

Additional reference words: DLL EXE ICO 32 x 32 16 x 16
KBCategory: kbprg kbcode
KBSubcategory: UsrRsc

How To Open Volumes Under Windows 95

PSS ID Number: Q125712
Authored 02-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

Windows 95 does not support opening disk drives or disk partitions with CreateFile(), as Windows NT does. Windows 95 also does not support the DeviceIoControl() IOCTL APIs, as Windows NT does. Instead, low-level disk access in Windows 95 can be achieved through DeviceIoControl() calls to the VWIN32 VxD.

MORE INFORMATION

Windows NT supports obtaining a handle to a disk drive or disk partition by using CreateFile() and specifying the name of the drive or partition as the filename (e.g. "\\.\PHYSICALDRIVE0" or "\\.\C:"). This handle can then be used in the DeviceIoControl() Win32 API.

Windows 95 differs in the following ways:

1. Obtaining a disk drive or disk partition handle is not supported. The call to CreateFile() will fail, and GetLastError() will return error code 2, ERROR_FILE_NOT_FOUND.

2. The DeviceIoControl IOCTL functions (such as IOCTL_DISK_FORMAT_TRACKS) are not supported. These IOCTLs require the handle to a disk drive or disk partition and thus can't be used.

3. DeviceIoControl() is called using a handle to a VxD rather than a handle to a disk drive or disk partition. Obtain a handle to VWIN32.VXD by using CreateFile("\\\\.\\VWIN32", ...). Use this handle in calls to DeviceIoControl() to perform volume locking (Int 21h Function 440Dh, Subfunctions 4Ah and 4Bh), and then to perform BIOS calls (Int 13h), Absolute Disk Reads/Writes (Int 25h and 26h), or MS-DOS IOCTL functions (Int 21h Function 440Dh).

REFERENCES

The chapter "Device I/O Control in Windows 95" describes the procedures for using DeviceIoControl() in Windows 95, and contains a description of the MS-DOS IOCTL functions supported by the VWIN32 VxD.

For information on using CreateFile() to obtain disk drive or disk partition handles under Windows NT, see the description for CreateFile() in the Microsoft Windows Programmer's Reference, Volume 3.

For a complete list of IOCTLs, see the description of the DeviceIoControl() function in the Microsoft Windows Programmer's Reference, Volume 3.

Additional reference words: 4.00
KBCategory: kbprg
KBSubcategory: BseFileio

How to Overlay Images Using Image List Controls

PSS ID Number: Q125629
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

One of the more interesting controls Windows 95 provides as part of its new common controls is the image list. Image lists provide an easy way to manage a group of bitmaps and draw them on the screen, without having to worry about calling CreateCompatibleDC(), SelectObject(), BitBlt(), StretchBlt(), and so on.

One interesting feature that image lists provide through the ImageList_Draw() API is the ability to overlay images -- that is, to draw an image transparently over another image. Calling ImageList_Draw() with the last parameter set to an index to an overlay mask instructs the image list to draw an image, and draw the overlay mask on top of it.

MORE INFORMATION

To overlay images correctly using image lists, follow these steps:

1. Create a bitmap that will have the images you want to draw as well as the overlay images you want drawn on top of these images.

   For example, say you have a bitmap of four 16x16 images:

   - a green circle.
   - a red circle.
   - a panda.
   - a frog.

2. Create an image list out of the bitmap you've created in step 1 by using ImageList_LoadImage() as shown here:

```
hImageList = ImageList_LoadImage (hInst,
                                  "MyBitmap",
                                  16,
                                  4,
                                  RGB (255,0,0),
                                  IMAGE_BITMAP,
                                  0);
```

3. Decide which images you want to specify as overlay masks, and tag them as such by using the ImageList_SetOverlayImage() function. The following code specifies the panda and the frog (with 0-based index, this comes out to image 2 and 3) as overlay masks 1 and 2.

NOTE: You can only specify up to four overlay masks per image list.

```
ImageList_SetOverlayImage (hImageList,
                           2,         // 0-based index to image list
                           1);        // 1-based index to overlay mask.


ImageList_SetOverlayImage (hImageList,
                           3,       // 0-based index to image list
                           2);      // 1-based index to overlay mask.
```

4. Draw the image. The following code draws the green circle (or image 0 in the example image list). Then it draws the panda (overlay image 1 in the example) on top of it.

```
ImageList_Draw (hImageList,
                0,     // 0-based index to imageList of image to draw
                hDC,   // handle to a DC
                16, 16 // (x,y) location to draw
                INDEXTOOVERLAYMASK (1));   // Overlay image #1
```

Additional reference words: 4.00
KBCategory: kbprg
KBSubcategory: UsrCtl

How to Override Full Drag

PSS ID Number: Q121541
Authored 09-Oct-1994 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.5, 3.51, and 4.0

Windows NT version 3.5 introduces full drag, which allows you to see the entire window moving or resizing instead of seeing just an outline of the window moving or resizing. You can enable full drag by running the Desktop Control Panel applet and selecting the Full Drag checkbox.

When you resize a window with full drag enabled, the application will receive numerous messages indicating that the window is resizing. (You can verify this with Spy.) If this has undesirable effects on your application, you will need to override the full drag feature in your application.

When the moving or resizing starts, the application receives this message:

```
WM_ENTERSIZEMOVE (0231)
```

When the moving or resizing finishes, the application receives this message:

```
WM_EXITSIZEMOVE (0232)
```

The above messages act as a notification that the window is entering and exiting a sizing or moving operation. If you want, you can use these notifications to set a flag to prevent the program from handling a WM_PAINT message during the move or size operation to override full drag.

Additional reference words: 3.50 3.51 4.00 95
KBCategory: kbprg kbtool
KBSubcategory: UsrWndw

How to Pass Large Memory Block Through Win32s Universal Thunk

PSS ID Number: Q126708
Authored 28-Feb-1995 Last modified 03-Mar-1995

The information in this article applies to:

- Microsoft Win32s version 1.2

SUMMARY

You can pass a memory address to a thunk routine. The pointer address is translated via the universal thunk (UT). However, the translated pointer is only guaranteed for 32K. This article describes ways to pass a larger memory block through the universal thunk.

MORE INFORMATION

GlobalAlloc()

You can call GlobalAlloc() to allocate a larger memory block on the 32-bit side of the thunk, copy the data into this memory block, send the handle to the 16-bit side, and lock the handle on the 16-bit side with GlobalLock().

VirtualAlloc()

If you allocate the memory using VirtualAlloc(), it will be aligned on a 64K boundary, so that you can address the entire memory block. HeapAlloc() allocates large memory blocks using VirtualAlloc() as well. NOTE: You are still limited to 64K of memory, due to the selector tiling.

Allocate a selector

To use this method, get the 32-bit offset used by the Win32-based application and the selector base for the data selector returned by GetThreadSelectorEntry(), then calculate the linear address of the memory block. With this linear address, you can use AllocSelector(), SetSelectorBase(), and SetSelectorLimit() to access the memory block from the 16-bit side of the thunk.

NOTE: Sparse memory will cause problems in the general case. Make sure that the memory range has been not only reserved, but also committed.

Additional reference words: 1.20
KBCategory: kbprg
KBSubcategory: W32s

How To Pass Numbers to a Named Range in Excel through DDE

PSS ID Number: Q45714
Authored 14-Jun-1989 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

To send an array to Excel via DDE, you must send the array in TEXT, CSV, or BIFF clipboard format.

For example, if you want to send three numbers for one ROW in CSV format, use 1,2,3 (where each number is separated by a comma). If you want to place three numbers in one COLUMN in CSV format, place the CR/LF (Od 0a) (carriage return/line feed) characters after each number in the set.

If you would like to send the numbers via the TEXT format separated into columns, place the CR/LF characters between each number in the set. To organize the numbers into one row, place a TAB (09) character between each number.

In order to send an array of data into a named range, use the following steps:

1. Highlight the appropriate cells in Excel
2. Set up your typical hot link from Excel, for example:

   =Service|Topic!Item

3. Instead of hitting Enter after typing the above, hit Ctrl+Shift+Enter. This will cause your data to come in as an array, rather than as a single item.

   NOTE: To do the reverse of this, that is, for a client application to POKE data to Excel, the client will have to specify an item name of say, "R1C1:R1C2" to poke an array of data to the range R1C1..R1C2.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDde

How to Perform Auto Repeat as Media Player Does

PSS ID Number: Q124185
Authored 21-Dec-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0
- Microsoft Video for Windows Development Kit (DK) version 1.1

The Media Player (MPLAYER.EXE) included with Microsoft Windows and Microsoft Windows NT (MPLAY32.EXE) provides an auto-repeat option that automatically repeats the playback of a multimedia file. You can incorporate this functionality into your application on Digital Video devices by using an extension to the standard Media Control Interface (MCI) commands as follows:

- When calling the mciSendString() function, add the word "repeat" to the play command, as in this example:

```
mciSendString("play mov notify repeat", NULL, 0, hWnd);
```

- When calling the mciSendCommand() function, set the play flag MCI_DGV_PLAY_REPEAT. For example, to add auto repeat functionality to the MOVPLAY sample included with the Video for Windows DK, add the following line to the playMovie() function in MOVPLAY1.C, right before the mciSendCommand() function call:

```
dwFlags |= MCI_DGV_PLAY_REPEAT;
```

"Digital Video Command Set for the Media Control Interface" documents the Digital Video MCI extensions. It is available on the Microsoft Developer Network (MSDN) CD. Look for it in the Specifications section of the CD contents, under "Digital Video MCI Specification." You can also search the CD using the word MCI_DGV_PLAY_REPEAT for more information about that flag.

Additional reference words: 3.10 3.50 4.00 95 Video for Windows MCIAVI MCI_PLAY AVI loop continuous play
KBCategory: kbmm kbprg kbdocerr
KBSubcategory: MMVideo

How to Port a 16-bit DLL to a Win32 DLL

PSS ID Number: Q125688
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

There are several significant differences between Win16 DLLs and Win32 DLLs. These differences require more than just a simple recompilation to turn your Win16 DLL into a Win32 DLL. In this article we will show you how to port your Win16 DLL to a Win32 DLL.

MORE INFORMATION

The first major difference is that the Win32 DLL entry point is called with every process attach and detach. Secondly, you must account for the fact that processes can be multithreaded and as such your DLL entry point will be called with thread attach and detach messages. You need to ensure that your DLL is "thread-safe" by using multithreaded libraries and mutual exclusion for functions in your DLL that would otherwise cause data corruption when preempted and reentered. This requires that you use Win32 synchronization methods to guard critical resources. Finally, each Win32 process gets its own copy of the Win32 DLL's data.

Step One for Porting Your DLL

The first step in porting a DLL from 16-bit Windows to 32-bit Windows is moving code from your LibMain (or LibEntry) and _WEP (or WEP) to the new DLL initialization function. The new DLL initialization function is called DllMain. You might code your DllMain like this:

```
BOOL WINAPI DllMain (HANDLE hModule, DWORD fdwReason, LPVOID lpReserved)
{
   switch (fdwReason)
   {
      case DLL_PROCESS_ATTACH:
         /* Code from LibMain inserted here.  Return TRUE to keep the
            DLL loaded or return FALSE to fail loading the DLL.

            You may have to modify the code in your original LibMain to
            account for the fact that it may be called more than once.
            You will get one DLL_PROCESS_ATTACH for each process that
            loads the DLL. This is different from LibMain which gets
            called only once when the DLL is loaded. The only time this
            is critical is when you are using shared data sections.
            If you are using shared data sections for statically
            allocated data, you will need to be careful to initialize it
```

```
            only once. Check your code carefully.

            Certain one-time initializations may now need to be done for
            each process that attaches. You may also not need code from
            your original LibMain because the operating system may now
            be doing it for you.
         */
         break;

      case DLL_THREAD_ATTACH:
         /* Called each time a thread is created in a process that has
            already loaded (attached to) this DLL. Does not get called
            for each thread that exists in the process before it loaded
            the DLL.

            Do thread-specific initialization here.
         */
         break;

      case DLL_THREAD_DETACH:
         /* Same as above, but called when a thread in the process
            exits.

            Do thread-specific cleanup here.
         */
         break;

      case DLL_PROCESS_DETACH:
         /* Code from _WEP inserted here.  This code may (like the
            LibMain) not be necessary.  Check to make certain that the
            operating system is not doing it for you.
         */
         break;
   }

   /* The return value is only used for DLL_PROCESS_ATTACH; all other
      conditions are ignored.  */
   return TRUE;   // successful DLL_PROCESS_ATTACH
}
```

DllMain Called with Flags

There are several conditions where DllMain is called with the DLL_PROCESS_ATTACH, DLL_PROCESS_DETACH, DLL_THREAD_ATTACH, or DLL_THREAD_DETACH flags.

The DLL_PROCESS_ATTACH flag is sent when a DLL is loaded into the address space of a process. This occurs in both situations where the DLL is loaded with LoadLibrary, or implicitly during application load. When the DLL is implicitly loaded, DllMain is executed with DLL_PROCESS_ATTACH before the processes enter WinMain. When the DLL is explicitly loaded, DllMain is executed with DLL_PROCESS_ATTACH before LoadLibrary returns.

The DLL_PROCESS_DETACH flag is sent when a process cleanly unloads the DLL

from its address space. This occurs during a call to FreeLibrary, or if the DLL is implicitly loaded, a clean process exit. When a DLL is detaching from a process, the individual threads of the process do not call the DLL_THREAD_DETACH flag.

The DLL_THREAD_ATTACH flag is sent when a new thread is being created in a process already attached to the DLL. Threads in existence before the process attached to a DLL will not send the DLL_THREAD_ATTACH flag. The first thread to attach to the DLL does not send the DLL_THREAD_ATTACH flag; it sends the DLL_PROCESS_ATTACH flag instead.

The DLL_THREAD_DETACH flag is sent when a thread is exiting cleanly. There is a situation when DllMain may be called when the thread did not first send the DLL_THREAD_ATTACH flag. This can happen if there are other threads still running and the original thread exits cleanly. The thread originally called DllMain with the DLL_PROCESS_ATTACH flag and later calls DllMain with the DLL_THREAD_DETACH flag. You may also have DllMain being called with DLL_THREAD_DETACH if a thread exits but was running in the process before the call to LoadLibrary.

Situations Where DllMain is Not Called or Is Bypassed

DllMain may not be called at all in dire situations where a thread or process was killed by a call to TerminateThread or TerminateProcess. These functions bypass calling DllMain. They are recommended only as a last resort. Data owned by the thread or process is at risk of loss because the process or thread could not shut itself down properly.

DllMain may be bypassed intentionally by a process if it calls DisableThreadLibraryCalls. This function (available with Windows 95 and Windows NT versions 3.5 and later) disables all DLL_THREAD_ATTACH and DLL_THREAD_DETACH notifications for a DLL. This enables a process to reduce its code size and working set. For more information on this function, see the SDK documentation on DisableThreadLibraryCalls.

Step Two for Porting Your DLL

The second step of porting your DLL involves changing functions that were declared with __export and included in the module definition (.DEF) file EXPORTS section. The proper export declaration for the Microsoft Visual C++ 32-bit compiler and linker is __declspec(dllexport). This declaration should be used when prototyping and declaring functions. You do not have to explicitly declare an exported function in the DEF file for the proper LIB and EXP files to be created; __declspec(dllexport) will do this for you. Here's an example of an exported function:

```
// prototype in DLL is necessary
__declspec(dllexport) DWORD WINAPI DLLFunc1(LPSTR);

// function
__declspec(dllexport) DWORD WINAPI DLLFunc1(LPSTR lpszIn)
{
   DWORD dwRes;
```

```
   /* DLL function logic */

   return dwRes;
}
```

To include the function in an application, prototype the above function in the application with the __declspec(dllimport) modifier. __declspec(dllimport) is not necessary, but does improve the speed of your code that implements the function call. Here's an example:

```
   __declspec(dllimport) DWORD WINAPI DLLFunc1(LPSTR);
```

Then, link the DLL's import library (.LIB) file with the application makefile or project.

Some sections of your DEF file will be ignored by the 32-bit linker because of architectural differences between Win16 and Win32. You may still use the EXPORTS section of your DEF if you wish to include ordinals for exported functions, or to rename exported functions. See your linker documentation for more information about what is acceptable in a DEF file for a Win32 DLL. Users of other 32-bit compilers and linkers will have to refer to their documentation for more information on exporting functions.

Applications that link to your DLL may be multithreaded. This possibility means that you should always build your Win32 DLL as multithreaded to support preemption and reentrancy. If you use runtime library functions in your DLL, they may be preempted and reentered. That would cause problems for a normal runtime library. If you use C runtime or some other runtime library, you should use a multithreaded version of the runtime library. Microsoft Visual C++ users should link with the /MT option and include LIBCMT.LIB. This will include the multithreaded C runtime library. Optionally, you can select the Multithreaded Run-time Library option in the project settings in the Visual Workbench. If you are using another runtime library and cannot get a multithreaded version of the library, you must protect calls to the library from reentrancy using a mutex or critical section synchronization object. Information later in this article discusses this issue.

Other Design Issues You Should Consider

One of the most significant changes to DLLs in 32-bit Windows operating systems is that each process executes in its own private address space. This means that a DLL cannot directly share dynamically-allocated memory between processes. Addresses are 32-bit offsets in a process's address space. Passing them between processes is possible, but won't work as in Win16 because each process has its own address space. In another process, this pointer may address unknown data, or invalid memory space.

"DS != SS" issues common to 16-bit DLLs no longer apply. A Win32 process executes in its own private address space and there is no segmentation of this address space. In a Win32 DLL, all functions are called using the calling thread's stack and all pointers are 32-bit linear addresses. In a Win32 process or DLL, a FAR pointer is the same as a near pointer. In other

words, a pointer is just a pointer.

If you must share data, you can specify certain global variables to be shared among processes by using a shared data section in the DLL. For more information on shared data sections, please search the Microsoft Knowledge Base using the following words:

   #pragma data_seg

or:

   sections share dll

You can also use a file-mapping object to share memory by sharing the system page file. This will allow two different processes to share dynamic memory. For more information on file mapping objects, please search the Microsoft Knowledge Base using these words:

   shared memory

Another significant change in the behavior of Win32 DLLs from Win16 DLLs is the inclusion of synchronization. The Win32 API provides synchronization objects that allow the programmer to implement correct synchronization. You should be aware that your 32-bit DLL may be preempted and called again from a different thread in the process. For example, if a thread executing a DLL function accesses global data and is preempted and another function modifies the same data, the original thread will resume but will be using modified data. You'll need to use synchronization objects to resolve this situation.

Your Win32 DLL may also be preempted by a thread in a different process. This situation becomes important if the functions use shared data sections or file mappings. You will need to examine the functions in a DLL to determine if this will cause problems. If so, you will have to control access to data or sections of code that are sensitive to this problem. Mutexes and critical sections are well suited for DLL synchronization. For more information on synchronization, please search the Microsoft Knowledge Base using these words:

   synchronization objects

For additional information, please search the Microsoft Knowledge Base using these words:

   Win32 DLL

Additional reference words: 4.00 LibEntry LibMain Port WEP _WEP Win32
KBCategory: kbprg
KBSubcategory: BseDll

How to Program Keyboard Interface for Owner-Draw Menus

PSS ID Number: Q121623
Authored 12-Oct-1994 Last modified 15-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

You can implement the keyboard interface for owner-draw menus, which allow a user to access a menu by typing a menu mnemonic, by processing the WM_MENUCHAR message.

MORE INFORMATION

Menus other than owner-draw menus can specify a menu mnemonic by inserting an underscore next to a character in the menu string so that the user can select the menu by typing ALT+<menu mnemonic character>. But in owner-draw menus, you cannot specify a menu mnemonic in this manner. Instead, you must process the WM_MENUCHAR message to provide owner-draw menus with menu mnemnoics.

WM_MENUCHAR is sent when the user types a menu mnemonic that does not match any of the predefined mnemonics of the current menu. wParam specifies the ASCII character that corresponds to the key the user pressed together with the ALT key. The low-order word of lParam specifies the type of the selected menu and contains:

- MF_POPUP if the cuurent menu is a popup menu.
- MF_SYSMENU if the menu is the system menu.

The high-order word of lParam contains the menu handle of the current menu. The window with the owner-draw menus can process WM_MENUCHAR as follows:

```
   case WM_MENUCHAR:
      nIndex = Determine index of menu item to be selected from
               character that was typed and handle of the current
               menu.
      return MAKELRESULT(nIndex, 2);
```

The 2 in the high-order word of the return value informs Windows that the low-order word of the return value contains the zero-based index of the menu item to be selected by Windows.

Windows 95 defines four new constants that correspond to the possible return values from the WM_MENUCHAR message:

```
Constant    Value   Meaning
```

MNC_IGNORE	0	Informs Windows that it should discard the character the user pressed and create a short beep on the system speaker.
MNC_CLOSE	1	Informs Windows that it should close the active menu.
MNC_EXECUTE	2	Informs Windows that it should choose the item specified in the low-order word of the return value. The owner window receives a WM_COMMAND message.
MNC_SELECT	3	Informs Windows that it should select the item specified in the low-order word of the return value.

Additional reference words: 3.10 3.50 4.00
KBCategory: kbprg kbui
KBSubcategory: UsrMen

How to Remove Win32s

PSS ID Number: Q120486
Authored 12-Sep-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.0, 1.1, 1.15, and 1.2

SUMMARY

To remove Win32s:

1. Remove the following line from the [386Enh] section in the SYSTEM.INI file:

   device=<WINDOWS>\<SYSTEM>\win32s\w32s.386

   where <WINDOWS> and <SYSTEM> are where the Windows and System directories are, respectively.

2. Modify the following line from the [BOOT] section in the SYSTEM.INI file:

   drivers=mmsystem.dll winmm16.dll

   to the following (remove winmm16.dll):

   drivers=mmsystem.dll

3. Delete the following files from the <WINDOWS>\<SYSTEM> subdirectory or from the SYSTEM directory in network installations:

   W32SYS.DLL
   WIN32S16.DLL
   WIN32S.INI

4. Delete all the files in the <WINDOWS>\<SYSTEM>\WIN32S subdirectory or the <SYSTEM>\WIN32S subdirectory in network installations. Then delete subdirectory itself.

5. Restart Windows.

NOTE: <WINDOWS> refers to the windows installation directory. On a networked Windows installation, the system directory may be located on a remote share. If you are only removing Win32s from your machine, then you do not need to remove the shared files (in <SYSTEM> and <SYSTEM>\WIN32S). Before removing these files from the network share, make sure that all users that use Win32s have removed the references to Win32s from the SYSTEM.INI file.

MORE INFORMATION

This information can be found in the Win32s Programmer's Reference help file.

Additional reference words: 1.00 1.10 1.15 1.20
KBCategory: kbsetup kbprg
KBSubcategory: W32s

How to Right-Justify Menu Items in Windows 95

PSS ID Number: Q125675
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

In Windows 95, right-justify (right-align) a menu item by using the MFT_RIGHTJUSTIFY type in MENUITEMINFOSTRUCTURE.

MORE INFORMATION

There is a new menu type in Windows 95, MFT_RIGHTJUSTIFY type, which you can use to right justify a menu item. The Windows version 3.1 method of prefixing the string with "\a" or "\b" will no longer work.

To right justify a menu item in Windows 95:

1. Get the menu handle of the original menu.

2. Get the original menu item information stored in the MENUITEMINFO structure.

3. Change the menu item type to include MFT_RIGHTJUSTIFY by or'ing the original value with MFT_RIGHTJUSTIFY.

4. Set the new menu item information.

For example, to create a right-justified menu item, add the following code to WM_CREATE:

```
HMENU hMenu;
MENUITEMINFO  mii;
char szBuffer [80];

hMenu = GetMenu (hwnd);

// Get the original value of mii.fType first
// and OR that with MFT_RIGHTJUSTIFY
mii.cbSize = sizeof (MENUITEMINFO);
mii.fMask = MIIM_TYPE;
mii.dwTypeData= szBuffer;
mii.cch   = sizeof (szBuffer);

GetMenuItemInfo(hMenu, 1, TRUE, &mii);

// OR in MFT_RIGHTJUSTIFY type
mii.fMask = MIIM_TYPE;
```

```
   mii.fType  = mii.fType | MFT_RIGHTJUSTIFY;

   // Right justify the specified item and all those following it
   SetMenuItemInfo(hMenu, 1, TRUE,   &mii);

   return 0;
```

REFERENCES

For additional information on right justifying menus in Windows 3.1, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID: Q67063
TITLE   : Inserting Right Justified Text in a Menu in Windows

Additional reference words: 4.00 alignment align
KBCategory: kbprg
KBSubcategory: UsrMen

How to Search for Win32 Articles by KB Subcategory

PSS ID Number: Q115696
Authored 01-Jun-1994 Last modified 21-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

The Microsoft Knowledge Base (KB) is categorized by using keywords. This article lists the category and subcategory keywords specific to articles in the Microsoft Win32 Software Development Kit (SDK) for Windows NT Collection.

MORE INFORMATION

Microsoft Win32 SDK KBSubcategory Major and Minor Keywords

Each KB article in the Win32 SDK collection may contain one or more product-specific subcategory keywords (called KBSubcategory keywords) that places the article in an appropriate Win32 SDK category. Each KBSubcategory word is composed of the concatenation of a major topic keyword and a minor topic keyword. For example, you can find all the BASE DLL articles by using BseDll as the keyword when you query the Microsoft Knowledge Base.

Use the asterisk (*) wildcard to find articles that fall into the general categories or into an intermediate subcategory. For example, to find all the articles that apply to GDI issues, query on Gdi*. An article usually has only one subcategory keyword, but it may have more.

Here are the topics and the corresponding KBSubcategory keywords. The minor topics are in the indented list under each major topic.

Major Topic / Minor Topic	KBSubcategory Keyword
BASE (Bse)	
Comm APIs	BseCommapi
Console	BseCon
DLLs	BseDll
Error Debug	BseErrdebug
Exception Handling	BseExcept
File I/O	BseFileio
Floating Point	BseFltpt
IPC	BseIpc
Base Misc	BseMisc
Memory Management	BseMm

Procedure Threads	BseProcThrd
Security	BseSecurity
Service	BseService
Syncronization	BseSync
GDI (Gdi)	
Bitmaps	GdiBmp
Cursors/Icons	GdiCurico
DCs	GdiDc
Display	GdiDisplay
Drawing	GdiDrw
Fonts	GdiFnt
Metafiles	GdiMeta
GDI Misc	GdiMisc
OpenGL	GdiOpenGL
Palettes/Colors	GdiPal
Brushes/Pens	GdiPnbr
Printing	GdiPrn
Screen Saver	GdiScrsav
TrueType	GdiTt
MULTIMEDIA	
CD-ROM	MMCDROM
MM File I/O Services	MMIO
Joystick	MMJoy
MediaView	MMMediaView
Midi	MMMidi
Sound Card Mixer Control	MMMixer
Multimedia Timers	MMTimer
Video for Windows (VFW)	MMVideo
Waveform Audio	MMWave
Miscellaneous	MMMisc
MISC	
International Issues	WIntlDev
Setup/Install	Setins
Subsystems	SubSys
NETWORKING (Ntwk)	
Networking Misc	NtwkMisc
LAN Manager APIs	NtwkLmapi
NetBIOS	NtwkNetbios
RAS APIs	NtwkRAS
RPC	NtwkRpc
SNMP	NtwkSnmp
TC/PIP	NtwkTcpip
WNet APIs	NtwkWinnet
Windows Sockets	NtwkWinsock
PEN	Wpen*
Video Drivers	WpenVideoDrv
Pen Drivers	WpenTbltDrv
OAK	WpenOemOak
Recognizer	WpenRcgnzr
Dictionary	WpenDict

Visual Basic	WpenVB
Training	WpenTrain
RC	WpenRc
RCRESULT	WpenRcResult
SYG	WpenSyg
SYV	WpenSyv
SYC	WpenSyc
SYE	WpenSye
Ink	WpenInk
Setup	WpenSetup
Pen applets	WpenApps
control panel apps	WpenCpl
PEN UI	WpenPenUI
Gestures	WpenGestures
Keyboard	WpenKeyboard
Pen Misc	WpenMisc

TAPI

Telephony API	Tapi

TOOLS (Tls)

Dialog Editor	TlsDlg
Font Editor	TlsFnt
Headers	TlsHdr
Help Compiler	TlsHlp
Image Editor	TlsImg
MEP	TlsMep
Misc	TlsMisc
MS Setup	TlsMss
Porting Tool	TlsPort
Profiler	TlsPrf
Resource Compiler	TlsRc
Spy	TlsSpy
WinDbg	TlsWindbg

USER (Usr)

Clipboard	UsrClp
Classes	UsrCls
Common Dialogs	UsrCmnDlg
Controls	UsrCtl
DDE	UsrDde
Dialog/Message Boxes	UsrDlgs
Drag & Drop	UsrDnd
Extension Libraries	UsrExt
Hooks	UsrHks
Input-Mouse/Keyboard	UsrInp
Localization/Intntl.	UsrLoc
MDI	UsrMdi
Menus/Accelerators	UsrMen
User Misc	UsrMisc
Network DDE	UsrNetDde
NLS	UsrNls
Owner Draw	UsrOwn
Resources	UsrRes
New Shell issues	UsrShell

	Strict	UsrStrict
	Window Manager	UsrWndw
WIN32S		
	Win32s	W32s

Product-Specific Keywords

You can use the KBSubcategory keywords to organize Win32 SDK articles or to search for specific groups of Win32 SDK articles. For information about KBSubcategory keywords for other Microsoft developer products, please query the Microsoft Knowledge Base using these keywords:

   dskbguide and kbkeyword

KB-Wide Keywords

Each article in the Win32 SDK collection also contains at least one generic, KB-wide category keyword (called a KBCategory keyword). The KBCategory keywords are standard throughout the Microsoft Knowledge Base, appearing in all KB articles, regardless of product. You can use the KBCategory keywords to organize all KB articles or to search for articles across several Microsoft products. For more information about these KBCategory keywords, please see the following article in the Microsoft Knowledge Base:

   ARTICLE-ID: Q94671
   TITLE   : Categories and Keywords for All Knowledge Base Articles

Additional reference words: 3.10 3.50 kbkeyword kbsubcats dskbguide
KBCategory: kbref
KBSubcatgory:

How to Select a Listview Item Programmatically in Windows 95

PSS ID Number: Q131284
Authored 07-Jun-1995 Last modified 10-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.51, 4.0
- Microsoft Win32s version 1.3

SUMMARY

Selecting a listview item in Windows 95 is not as easy as selecting a list box item was in Windows version 3.1. To select a list box item in Windows version 3.1, an application sends an LB_SETCURSEL or LB_SETSEL to a single- or multiple-selection list box respectively. To select a listview item in Windows 95, an application sends an LVM_SETITEMSTATE message or calls the ListView_SetItemState() macro.

MORE INFORMATION

An application can force a selection of a listview item. You might want the application to do this when a user clicks a column other than the first column of a listview of multiple subitems or columns.

Currently, a listview item is selected only when the user clicks the first column of that item. However, you many want the application to select the item regardless of which column in the listview is clicked.

Windows 95 does not provide a separate message or function to set the current selection in a listview. Instead, it defines item states or LVIS_* values that determine the listview item's appearance and functionality. LVIS_FOCUSED and LVIS_SELECTED in particular are the states that determine a listview item's selection state.

To select a listview item programmatically, an application sets the listview item's state as follows:

```
   ListView_SetItemState (hWndListView,         // handle to listview
                          iWhichItem,         // index to listview item
                          LVIS_FOCUSED | LVIS_SELECTED, // item state
                          0x000F);                      // mask
```

Note that the last parameter passed to this macro is a mask specifying which bits are about to change. LVIS_FOCUSED and LVIS_SELECTED are defined in <commctrl.h> as 0x0001 and 0x0002 respectively, so you need to set the last four bits of the mask.

The same principle applies to selecting a treeview item programmatically. The only difference is that an application sends a TVM_SETITEM message or calls the TreeView_SetItem() macro.

Because listviews allow multiple selection by default, you can program an application to select multiple items by simulating a CTRL keydown (or SHIFT keydown event) prior to setting the item state. For example, the following code simulates the pressing of the CTRL key:

```
BYTE  pbKeyState [256];

GetKeyboardState ((LPBYTE)&pbKeyState);
pbKeyState[VK_CONTROL] |= 0x80;
SetKeyboardState ((LPBYTE)&pbKeyState);
```

Note that if an application simulates a keypress, it must also be responsible for releasing it by resetting the appropriate bit. For example, the following code simulates the release of a CTRL key:

```
BYTE  pbKeyState [256];

GetKeyboardState ((LPBYTE)&pbKeyState);
pbKeyState[VK_CONTROL] = 0;
SetKeyboardState ((LPBYTE)&pbKeyState);
```

Similarly, retrieving the currently selected item in a listview control in Windows 95 is not as easy as sending an LB_GETCURSEL message to a listbox control was in Windows version 3.1.

For listviews, call the ListView_GetNextItem() function with the LVNI_SELECTED flag specified:

```
iCurSel = ListView_GetNextItem (ghwndLV, -1, LVNI_SELECTED);
```

For treeviews, retrieve the currently selected item by calling the TreeView_GetNextItem() function with the TVGN_CARET flag specified or by calling the TreeView_GetSelection() macro directly:

```
iCurSel = TreeView_GetNextItem (ghwndTV, NULL, TVGN_CARET);
   or
iCurSel = TreeView_GetSelection (ghwndTV);
```

Additional reference words: 4.00 1.30
KBCategory: kbprg kbcode
KBSubcategory: UsrCtl

How to Set Foreground/Background Responsiveness in Code

PSS ID Number: Q125660
Authored 01-Feb-1995 Last modified 03-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 3.5

SUMMARY

In Windows NT version 3.5, you can set foreground/background responsiveness by using the System Control Panel in Program Manager. Chose Tasking, then select one of the following through the dialog that is displayed:

- Best Foreground Application Response Time.

- Foreground Application More Responsive than Background.

- Foreground and Background Applications Equally Responsive.

This article describes how to achieve the same thing by using code in a program. It also explains how to override this setting by using code in a program.

MORE INFORMATION

You can use the Registry APIs to set foreground/background responsiveness. The following registry key allows you to specify the priority to give to the application running in the foreground:

```
HKEY_LOCAL_MACHINE\SYSTEM
   CurrentControlSet\
   Control\
   PriorityControl\
   Win32PrioritySeparation
```

The following values are supported:

Value	Meaning
0	Foreground and background applications equally responsive
1	Foreground application more responsive than background
2	Best foreground application response time

These values correspond to the choices offered in the Tasking dialog described in the "Summary" section of this article.

To override the setting from your application, use SetPriorityClass() to change your application's priority class and SetThreadPriority() to set the priority for a given thread.

NOTE: The thread priority together with the priority class for the process determine the thread's base priority.

Additional reference words: 3.50
KBCategory: kbprg
KBSubcategory: BseProcThrd

How to Set the Current Normal Vector in an OpenGL Application

PSS ID Number: Q131130
Authored 04-Jun-1995 Last modified 07-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5

SUMMARY

In a Microsoft Win32 OpenGL application, it is common practice to construct objects with the glBegin function followed by several calls to glVertex. For example, to create a flat polygon in three-dimensional space, you could write this code:

```
glBegin(GL_POLYGON);
glVertex3f(.....);
glVertex3f(.....);
glVertex3f(.....);
.  .  .
.  .  .
glEnd();
```

Now, if you want to implement a light source or multiple light sources in your OpenGL application, it is important that you include a call to the glNormal function between the calls to glBegin and glEnd so that the normal vector can be used by OpenGL when calculating the color to use when filling the polygon.

You can perform a vector cross product on two vectors to obtain a third vector that is perpendicular to the plane containing the two vectors. Using a vector cross product, you can calculate the vector normal to the polygon and use that value in your call to glNormal.

MORE INFORMATION

Using cross product math on two vectors of a polygon, you can obtain a vector that is perpendicular to the polygon. Two sides of a polygon describe two vectors in the plane of the polygon. With those two vectors, you can calculate a vector that is perpendicular to the polygon. The length of the normal vector calculated will not be unit length, and the normal vector needs to be unit length. Therefore, you need to call glEnable(GL_NORMALIZE) when you initialize your OpenGL application, so that normal vectors specified with glNormal are scaled to unit length after transformation.

Code Sample

You can use the following function to calculate a normal vector for a

polygon. You need to give it three points of the polygon and the points should be given in clock-wise order when you are facing the front of the polygon:

```
//*************************************************************************
// Function: CalculateVectorNormal
//
// Purpose: Given three points of a 3D plane, this function calculates
//          the normal vector of that plane.
//
// Parameters:
//     fVert1[]   == array for 1st point (3 elements are x, y, and z).
//     fVert2[]   == array for 2nd point (3 elements are x, y, and z).
//     fVert3[]   == array for 3rd point (3 elements are x, y, and z).
//
// Returns:
//     fNormalX   == X vector for the normal vector
//     fNormalY   == Y vector for the normal vector
//     fNormalZ   == Z vector for the normal vector
//
// Comments:
//
// History:  Date       Author        Reason
//           3/22/95     GGB           Created
//*************************************************************************

GLvoid CalculateVectorNormal(GLfloat fVert1[], GLfloat fVert2[],
                             GLfloat fVert3[], GLfloat *fNormalX,
                             GLfloat *fNormalY, GLfloat *fNormalZ)
    {
    GLfloat Qx, Qy, Qz, Px, Py, Pz;

    Qx = fVert2[0]-fVert1[0];
    Qy = fVert2[1]-fVert1[1];
    Qz = fVert2[2]-fVert1[2];
    Px = fVert3[0]-fVert1[0];
    Py = fVert3[1]-fVert1[1];
    Pz = fVert3[2]-fVert1[2];

    *fNormalX = Py*Qz - Pz*Qy;
    *fNormalY = Pz*Qx - Px*Qz;
    *fNormalZ = Px*Qy - Py*Qx;
}
```

Code to Call and Use the CalculateVectorNormal Function

Here is an example of how you might call and use the function:

```
   glBegin(GL_POLYGON);
   glVertex3fv(fVert1);
   glVertex3fv(fVert2);
   glVertex3fv(fVert3);
   glVertex3fv(fVert4);
```

```
   // Calculate the vector normal coming out of the 3D polygon.
   CalculateVectorNormal(fVert1, fVert2, fVert3, &fNormalX,
                         &fNormalY, &fNormalZ);
   // Set the normal vector for the polygon
   glNormal3f(fNormalX, fNormalY, fNormalZ);
   glEnd();
```

Additional reference words: 3.50 graphics
KBCategory: kbprg kbcode
KBSubcategory: GdiOpenGL

How to Set Up and Run the RNR Sample Included in the Win32

PSS ID Number: Q131505
Authored 13-Jun-1995 Last modified 14-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.5, 3.51

SUMMARY

This article explains how to set up and operate the Service Registration and Resolution (RNR) sample over the TCP and SPX network protocols. The RNR sample comes with the Microsoft Win32 (SDK) versions 3.5 and 3.51. It illustrates the use of the service registration and resolution APIs.

MORE INFORMATION

How to Set Up the RNR Sample

Run rnrsetup /ADD on each client and on the server to call the SetService API with SERVICE_ADD_TYPE. This call stores the service name type, its associated GUID, and relevant addressing information for the specified name spaces in the registry path:

```
\HKEY_LOCAL_MACHINE
  \CurrentControlSet
    \Control
      \ServiceProvider
        \ServiceTypes
```

This information is then retrieved by GetTypeByName() and GetAddressByName() calls respectively to identify the server's address.

How to Operate the RNR Sample

1. Start rnrsrv on one machine. After setting listening ports on the available protocols, the RNR server executes the SetService() call with the SERVICE_REGISTER flag to register the network service with the specified name spaces. For example, it enables advertising the EchoExample service name on the SAP protocol.

2. On the other machine, start rnrclnt using the following syntax:

   ```
   rnrclnt /?

   Usage: rnrclnt [/name:SVCNAME] [/type:TYPENAME] [/size:N]
             [/count:N]  [/rcvbuf:N] [/sndbuf:N]
   ```

    - TYPENAME is initially passed to GetTypeByName() call to return

the GUID value. The GUID value and SVCNAME is then passed to GetAddressByName() to return the address of the server that the client can connect to. TYPENAME is defined as EchoExample for the RNR sample.

- SVCNAME specifies which EchoExample server to connect to. If SVCNAME is specified as the server name in the Internet domain, the TCP protocol will be used. If SVCNAME is specified as EchoServer (the RNR service name advertised on SAP), the SPX protocol will be used.

- The other parameters to the rnrclnt have appropriate default values and are self explanatory.

Additional reference words: 3.50 3.51
KBCategory: kbprg kbnetwork
KBSubcategory: NtwkWinsock

How to Shade Images to Look Like Windows 95 Active Icon

PSS ID Number: Q128786
Authored 10-Apr-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.5 and 4.0

SUMMARY

This article shows by example how to display an image or an icon in a shaded state, as Windows 95 does for the active icon.

MORE INFORMATION

Step-by-Step Procedure

To obtain the shaded look for your image or icon, follow these six steps:

1. Create a compatible DC and bitmap.

2. Create a monochrome pattern brush with every other pixel on.

3. Fill the memory image with the pattern.

4. BitBlt the source image over the pattern using SRCAND so that only the 'on' destination pixels are transferred.

5. Color the destination with the pattern, using the highlight color for the 'off' pixels and using black for the 'on' pixels.

6. Copy the filtered original from the memory DC to the destination using SRCPAINT so that only the 'on' pixels are transferred.

This results in the destination having the original image with every other pixel colored with the highlight color.

Sample Code

The following function implements these six steps to shade a rectangular area on a device context:

```
// ShadeRect
// hDC    : the DC on which the area is to be shaded
// lpRect : the coordinates within which to shade
BOOL ShadeRect( HDC hDC, LPRECT lpRect )
{
   COLORREF  crHighlightColor, crOldBkColor, crOldTextColor;
   HBRUSH    hBrush, hOldBrush;
```

```
HBITMAP   hBitmap, hBrushBitmap, hOldMemBitmap;
int       OldBkMode, nWidth, nHeight;
HDC       hMemDC;
RECT      rcRect = { 0, 0, 0, 0};
// The bitmap bits are for a monochrome "every-other-pixel"
//     bitmap (for a pattern brush)
WORD      Bits[8] = { 0x0055, 0x00aa, 0x0055, 0x00aa,
                      0x0055, 0x00aa, 0x0055, 0x00aa };

// The Width and Height of the target area
nWidth = lpRect->right - lpRect->left + 1;
nHeight = lpRect->bottom - lpRect->top + 1;

// Need a pattern bitmap
hBrushBitmap = CreateBitmap( 8, 8, 1, 1, &Bits );
// Need to store the original image
hBitmap = CreateCompatibleBitmap( hDC, nWidth, nHeight );
// Need a memory DC to work in
hMemDC = CreateCompatibleDC( hDC );
// Create the pattern brush
hBrush = CreatePatternBrush( hBrushBitmap );

// Has anything failed so far? If so, abort!
if( (hBrushBitmap==NULL) || (hBitmap==NULL) ||
    (hMemDC==NULL) || (hBrush==NULL) )
{
   if( hBrushBitmap != NULL ) DeleteObject(hBrushBitmap);
   if( hBitmap != NULL ) DeleteObject( hBitmap );
   if( hMemDC != NULL ) DeleteDC( hMemDC );
   if( hBrush != NULL ) DeleteObject( hBrush );
   return FALSE;
}

// Select the bitmap into the memory DC
hOldMemBitmap = SelectObject( hMemDC, hBitmap );

// How wide/tall is the original?
rcRect.right = nWidth;
rcRect.bottom = nHeight;

// Lay down the pattern in the memory DC
FillRect( hMemDC, &rcRect, hBrush );

// Fill in the non-color pixels with the original image
BitBlt( hMemDC, 0, 0, nWidth, nHeight, hDC,
lpRect->left, lpRect->top, SRCAND );

// For the "Shutdown" look, use black or gray here instead
crHighlightColor = GetSysColor( COLOR_HIGHLIGHT );

// Set the color scheme
crOldTextColor = SetTextColor( hDC, crHighlightColor );
crOldBkColor = SetBkColor( hDC, RGB(0,0,0) );
SetBkMode( hDC, OPAQUE );
```

```
   // Select the pattern brush
   hOldBrush = SelectObject( hDC, hBrush );
   // Fill in the color pixels, and set the others to black
   FillRect( hDC, lpRect, hBrush );
   // Fill in the black ones with the original image
   BitBlt( hDC, lpRect->left, lpRect->top, nWidth, nHeight,
           hMemDC, 0, 0, SRCPAINT );

   // Restore target DC settings
   SetBkMode( hDC, OldBkMode );
   SetBkColor( hDC, crOldBkColor );
   SetTextColor( hDC, crOldTextColor );

   // Clean up
   SelectObject( hMemDC, hOldMemBitmap );
   DeleteObject( hBitmap );
   DeleteDC( hMemDC );
   DeleteObject( hBrushBitmap );
   SelectObject( hDC, hOldBrush );
   DeleteObject( hBrush );

   return TRUE;
}
```

Additional reference words: 3.50 4.00 hatch darken shadow
KBCategory: kbgraphic kbcode
KBSubcategory: GdiMisc

How to Share Data Between Different Mappings of a DLL

PSS ID Number: Q125677
Authored 01-Feb-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

Under certain circumstances, 32-bit DLLs might have to share data with other 32-bit DLLs loaded by a different application or with different mappings of the same DLL. Because 32-bit DLLs are mapped into the the calling process's address space, which is private, sharing data with other DLLs mapped into the address spaces of different applications involves creating shared data section(s) or using memory mapped files. This article discusses the former -- creating shared data sections by using the #pragma statement. Typically, system-wide hooks installed in a DLL need to share some common data among different mappings.

MORE INFORMATION

Each Win32-based application runs in its own private address space. If a 32-bit application installs a system-wide hook with the hook callback function in a DLL, this DLL is mapped into the address space of every application for which the hook event occured.

Every application that the DLL gets mapped into, gets its own set of variables (data). Often there will be a scenario where hook callback functions mapped into different application or process address spaces need to share some data variables -- such as HHOOK or a Window Handle -- among all mappings of the DLL.

Because each application's address space is private, DLLs with hook callback functions mapped into one application's address spaces cannot share data (variables) with other hook callback functions mapped into a different application's address space unless a shared data SECTION exists in the DLL.

Every 32-bit DLL (or EXE) is composed of a collection of sections. Each section name begins with a period. The section of interest in this article is the data section. These sections can have one of the following attributes: READ, WRITE, SHARED, and EXECUTE.

DLLs that need to share data among different mappings can use the #pragma pre-processor command in the DLL source file to create a shared data section that contains the data to be shared.

The following sample code shows by example how to define a named-data section (.sdata) in a DLL.

Sample Code

```
#pragma data_seg(".sdata")
int iSharedVar = 0;
#pragma data_seg()
```

The first line directs the compiler to place all the data declared in this section into the .sdata data segment. Therefore, the iSharedVar variable is stored in the MYSEC segment. By default, data is not shared. Note that you must initialize all data in the named section. The data_seg pragma applies only to initialized data. The third line, #pragma data_seg(), resets allocation to the default data section.

If one application makes any changes to variables in the shared data section, all mappings of this DLL will reflect the same changes, so you need to be carefule when dealing with shared data in applications or DLLs.

You must also tell the linker that the variables in the section you defined are to be shared by modifying your .DEF file to include a SECTIONS section or by specifying /SECTION:.sdata, RWS in your link line. Here's an example SECTIONS section:

```
   SECTIONS
   .sdata   READ WRITE SHARED
```

In the case of a typical hook DLL, the HHOOK, HINSTDLL, and other variables can go into the shared data section.

Additional reference words: 4.00 95
KBCategory: kbprg
KBSubcategory: BseMisc

How to Simulate Changing the Font in a Message Box

PSS ID Number: Q68586
Authored 22-Jan-1991 Last modified 16-May-1995

-
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

-

SUMMARY

To simulate changing the font in a message box, create a dialog box that uses the desired font. Specify the style and contents of the dialog box to reflect the style of the desired message box. The application can also draw a system icon in the dialog box.

MORE INFORMATION

The message box is a unique object in Windows. Its handle is not available to an application; therefore, it cannot be modified. An application can simulate a message box with a different font by creating a dialog box that looks like a message box.

To change the font in a dialog box, use the optional statement FONT in the dialog statement of the resource script (.RC) file. For example, resource file statements for a dialog box displaying an error in Courier point size 12 would be as follows:

```
FontError DIALOG  45, 17, 143, 46
CAPTION "Font Error"
FONT 12, "Courier"
STYLE WS_CAPTION | WS_SYSMENU | DS_MODALFRAME
BEGIN
    CTEXT "Please select the right font", -1, 0, 7, 143, 9
    DEFPUSHBUTTON "OK"  IDOK,   56, 25,  32, 14, WS_GROUP
END
```

To center the dialog box in the screen, use GetWindowRect() to retrieve the dimensions of the screen and MoveWindow() to place the dialog box appropriately. The following code demonstrates this procedure:

```
case WM_INITDIALOG:
    GetWindowRect(hDlg, &rc);
    x = GetSystemMetrics(SM_CXSCREEN);
    y = GetSystemMetrics(SM_CYSCREEN);
    MoveWindow (hDlg,
       (x - (rc.right - rc.left)) >> 1,  /* x position */
       (y - (rc.bottom - rc.top)) >> 1,  /* y position */
       rc.right - rc.left,               /* x size */
```

```
          rc.bottom - rc.top,               /* y size */
          TRUE);                            /* paint the window */
       return TRUE;
```

To display a system icon in the dialog box, call the DrawIcon() function during the processing of a WM_PAINT message. After drawing the desired icon, the dialog procedure passes control back to the dialog manager by returning FALSE. The code to paint the exclamation point icon (used in warning messages) is as follows:

```
   case WM_PAINT:
       hIcon = LoadIcon(NULL, IDI_EXCLAMATION);
       hDC = GetDC(hDlg);
       DrawIcon(hDC, 20, 40, hIcon);
       ReleaseDC(hDlg, hDC);
       return FALSE;
```

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

How to Spawn a Console App and Redirect Standard Handles

PSS ID Number: Q126628
Authored 27-Feb-1995 Last modified 22-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.5, 3.51, and 4.0

This article discusses spawning a console application with CreateProcess() and redirecting its output. The standard handles are controlled with the STARTUPINFO fields hStdInput, hStdOutput, and hStdError.

In Windows NT version 3.1, if a windowed application spawned a console application, you could:

- Redirect none of its standard handles (don't use STARTF_USESTDHANDLES).

  -or-

- Redirect all of its standard handles (use STARTF_USESTDHANDLES).

For example, if you redirected hStdInput and hStdOutput, but left hStdError as 0 or INVALID_HANDLE_VALUE, the console application would fail if it tried to write to stderr. This is not a problem for a console application spawning another console application.

In Windows NT version 3.5 and later and in Windows 95, if you set any of these fields to INVALID_HANDLE_VALUE, Windows NT will assign the default value to that handle in the console application, rather than leaving it an invalid value. Therefore, if you set STARTF_USESTDHANDLES, but fail to set one of the handle fields, this will not cause a problem for the console application. You can now redirect standard input, but not standard output, and so forth.

Additional reference words: 3.50 4.00
KBCategory: kbprg
KBSubcategory: BseProcThrd

How to Specify a Full Path in the ExecProgram Macro

PSS ID Number: Q86477
Authored 07-Jul-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.51 and 4.0

SUMMARY

Windows help files (.HLP files) can be written such that selecting a designated topic executes a Windows-based application. This is done with the ExecProgram() macro. If the desired application does not reside in the same directory as the .HLP file, a full path to the .EXE must be specified. An invalid path produces an error.

SYMPTOMS

Attempting to execute an application with an invalid path causes Windows to display the following Help message box:

```
Unable to Run Specified File.
```

CAUSE

A common mistake is to incorrectly specify the subdirectory delimiters in the path description. This is not a problem with the Windows Help Compiler.

MORE INFORMATION

The Windows Help Compiler can recognize escape sequences expressed using the backslash (\) character. When using the ExecProgram() macro, four backslashes (\\\\) separate each directory in a full path description:

```
ExecProgram( "c:\\\\winapps\\\\excel\\\\excel", 0 )
```

With HC31.EXE version 3.10.505 and HCW.EXE version 4.0, you would use only two backslashes:

```
ExecProgram( "c:\\winapps\\excel\\excel", 0 )
```

Additional reference words: 3.00 3.10 4.00 95
KBCategory: kbtool
KBSubcategory: TlsHlp

How to Specify Filenames/Paths in Viewer/WinHelp Commands

PSS ID Number: Q120251
Authored 07-Sep-1994 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

Both WinHelp and Viewer have commands such as JumpId() that take a filename as a parameter. Depending on the circumstances under which such commands are called, the number of backslashes (\) included in the filename as part of the path may have to be doubled for the command to work correctly. For example, if JumpId() is called from a HotSpot, you need to use two backslashes to separate the subdirectory names as in this example:

```
JumpId("C:\\MYAPP\\HLPFILE\\MYHLP.HLP","topicx")
```

But if JumpId() is used as the command associated with a menu item, you need to use four backslashes to separate the subdirectory names within the JumpId command, which is itself part of an InsertItem() command. For example:

```
InsertItem("MNU_FILE","my_id","Jump",
   "JumpId('C:\\\\MYAPP\\\\HLPFILE\\\\MY HLP.HLP','topicx')",0)
```

To avoid having to modify the filename parameter depending on how the function is called, Microsoft recommends that you use a single forward slash (/) to separate subdirectory names within the filename. For example:

```
JumpId("C:/MYAPP/HLPFILE/MYHLP.HLP","topicx")
```

A single forward slash will work regardless of how the command is called.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsHlp

How to Specify Shared and Nonshared Data in a DLL

PSS ID Number: Q89817
Authored 30-Sep-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

To have both shared and nonshared data in a dynamic-link library (DLL) which is built with a 32-bit Microsoft C compiler, you need to use the #pragma data_seg directive to set up a new named section. You then must specify the correct sharing attributes for this new named data section in your .DEF file.

The system will try to load the shared memory block created by #pragma data_seg at the same address in each process. However, if the block cannot be loaded into the same memory address, the system allocates a new block of memory for that process and the memory is not shared. No run time warnings are given when this happens. Using memory-mapped files backed by pagefile (named shared memory) is a safer option than using #pragma data_seg, because the APIs will return an error when the mapping fails.

MORE INFORMATION

Below is a sample of how to define a named data section in your DLL. The first line directs the compiler to include all the data declared in this section in the .MYSEC data segment. This means that the iSharedVar variable would be considered part of the .MYSEC data segment. By default, data will be nonshared.

Note that you must initialize all data in your named section. The data_seg pragma only applies to initialized data.

The third line, "#pragma data_seg()", directs the compiler to reset allocation to the default data section.

```
#pragma data_seg(".MYSEC")
int iSharedVar = 0;
#pragma data_seg()
```

Below is a sample of the .DEF file that supports the shared and nonshared segments. This definition will set the default section .MYSEC to be shared. The default data section is by default non-shared, so any data not in section .MYSEC will be non-shared.

```
LIBRARY MyDll
SECTIONS
  .MYSEC READ WRITE SHARED
```

```
   EXPORTS
    ...
```

NOTE: All section names must begin with a period character ('.') and must not be longer than 8 characters, including the period character.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseDll

How to Start an Application at Boot Time Under Windows 95

PSS ID Number: Q125714
Authored 02-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

Windows NT supports a Win32-based application type known as a Service. A Service may be started at boot time, automatically, by a user with the Service Control Manager facility or by Win32-based applications that use the service-related Win32 APIs. The Service Control subsystem and the associated Win32 APIs are not supported in Windows 95. In place of services, Windows 95 has two registry keys that will allow users to run applications before a user logs in when the system first starts up.

MORE INFORMATION

Microsoft recognizes the value that Services and the Service Control manager have, therefore, we have implemented a smaller version of the Service Control Manager so that applications can run before a user logs in. This is known as MPREXE.EXE. At boot time, MPREXE checks two new registry keys: "RunServices" and "RunServicesOnce".

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion
   RunServices  [key]
      bubba95=service.exe /params [string value]

   ...

   RunServicesOnce  [key]

   ...
```

The value names are arbitrary. The value data is the command line passed to CreateProcess(). Values under the key RunServicesOnce are deleted after the application is launched. Because these applications are started before the user logs onto the system, the user has not been validated and the applications cannot assume that they have particular networking permissions enabled. Window 95, unlike Windows NT, only has one security context for the entire system. Therefore, don't assume that any application that MPREXE starts has access to a particular network resource because a particular user has access to this network resource.

Applications started by the RunServices and RunServicesOnce keys will be closed when the user selects "Close all programs and log on as a different user" from the Shutdown dialog on the Start Menu. To keep itself from being closed in this manner, a Win32-based application needs to call RegisterServiceProcess(). A Win32-based application may also call

RegisterServiceProcess to keep other processes from closing when the user logs out.

RegisterServiceProcess is implemented only in Windows 95, and is documented in the online help (WIN32.HLP).

Additional reference words: 3.95 4.00
KBCategory: kbprg
KBSubcategory: BseService

How to Stop a Journal Playback

PSS ID Number: Q98486
Authored 05-May-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0, 3.1, and 4.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

To stop a "journal playback" when a specified key is pressed, the filter function must determine whether the key was pressed and then call the UnhookWindowsHookEx function to remove the WH_JOURNALPLAYBACK hook.

MORE INFORMATION

To determine the state of a specified key, the filter function must call the GetAsyncKeyState function when the nCode parameter equals HC_SKIP. The HC_SKIP hook code notifies the WH_JOURNALPLAYBACK filter function that Windows is done processing the current event.

The GetAsyncKeyState function 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 the GetAsyncKeyState function. If the most significant bit of the return value is set, the key is down; if the least significant bit is set, the key was pressed after a preceding GetAsyncKeyState call.

If the filter function calls the GetAsyncKeyState function after the specified key was pressed and released, then the most significant bit will not be set to reflect a key-down. Thus, a test to check whether the specified key is down fails. Therefore, the least significant bit of the return value must be checked to determine whether the specified key was pressed after a preceding call to GetAsyncKeyState function. Using this technique of checking the least significant bit requires a call to the GetAsyncKeyState function before setting the WH_JOURNALPLAYBACK hook. For example:

```
// When setting the journal playback hook.
   .
   .
   .
   // Reset the lease significant bit.
   GetAsyncKeyState( VK_CANCEL );

   // Set a system-wide journal playback hook.
   g_hJP = SetWindowsHookEx( WH_JOURNALPLAYBACK,
            FilterFunc,
```

```
            g_hInstDLLModule,
            NULL );
    .
    .
    .

// Inside the filter function
    .
    .
    .
    if ( nCode == HC_SKIP )
        if ( GetAsyncKeyState( VK_CANCEL) )
            UnhookWindowsHookEx( g_hJP );
    .
    .
    .
```

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrHks

How to Subclass a Window in Windows 95

PSS ID Number: Q125680
Authored 01-Feb-1995 Last modified 10-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

While subclassing windows within the same application in Windows 95 is unchanged from Windows version 3.1, subclassing windows belonging to other applications is somewhat more complicated in Windows 95. This article explains the process.

MORE INFORMATION

For a 16-bit application, subclassing methods are the same as they were in Windows version 3.1. However, Windows 95 performs some behind-the-scenes magic to make it possible for a 16-bit window to subclass a 32-bit window.

Usually, a subclass consists of saving one window procedure and substituting another in its place. However, this could present a problem when a 16-bit application tries to call a 32-bit window procedure. Windows 95 works around this potential problem by providing 32-bit windows with a 16-bit window procedure. All 32-bit windows will have the same selector for their wndProcs that references code in KRNL386.EXE where the 16-bit wndProcs for all 32-bit windows are stored. Eventually, each of these 16-bit wndProcs will jump to the real 32-bit window procedure.

Subclassing windows belonging to another process, either 16-bit or 32-bit, from a 32-bit process or application works as it does in Windows NT. The difficulty is that each 32-bit process has its own private address space. Hence, a window procedure's address in one process is not valid in another. To get a window procedure from one process into another, you need to inject the subclass procedure code into the other process's address space. There are a number of ways to do this.

Three Ways to Inject Code Into Another Process's Address Space

You can use the registry, hooks, or remote threads and the WriteProcessMemory() API to inject code into another process's address space.

If you use the registry, the code that needs to be injected should reside in a DLL. By either running REGEDIT.EXE or using the registry APIs, add the \HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\AppInit_DLLs key to the registry if it does not exist. Set its value to a string containing the DLL's pathname. This key may contain more than one DLL pathname separated by single spaces. This has the effect, once the machine is restarted, of

loading the library with DLL_PROCESS_ATTACH into every process at its creation time. While this method is very easy, it also has several disadvantages. For example, the computer must be restarted before it takes effect, and the DLL will last the lifetime of the process.

You can also use hooks to inject code into another process's address space. When a window hooks another thread belonging to a different process, the system maps the DLL containing the hook procedure into the address space of the hooked thread. Windows will map the entire DLL, not just the hook procedure. So to subclass a window in another process, install a WH_GETMESSAGE hook or another such hook on the thread that owns the window to be subclassed. In the DLL that contains the hook procedure, include the subclass window procedure. In the hook procedure, call SetWindowLong() to enact the subclass. It is important to leave the hook in place until the subclass is no longer needed, so the DLL remains in the target window's address space. When the subclass is removed, the hook would be unhooked, thus unmapping the DLL.

A third way to inject a DLL into another address space involves the use of remote threads and the WriteProcessMemory() API. It is more flexible and significantly more complicated than the previously mentioned methods, and is described in the following reference.

REFERENCES

"Load Your 32-bit DLL into Another Process's Address Space Using INJLIB" by Jeffrey Richter, MSJ May 1994.

Additional reference words: 4.00
KBCategory: kbprg
KBSubcategory: UsrWndw

How to Support Language Independent Strings in Event Logging

PSS ID Number: Q125661
Authored 01-Feb-1995 Last modified 07-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1 and 3.5

SUMMARY

Insertion strings in the event log entries are language-independent. Instead of using string literals as the insertion string, use "%%n" as the insertion string.

MORE INFORMATION

When the event viewer sees "%%n", it looks up the ParameterMessageFile value in the registry, under the source of the event, as in this example:

```
HKEY_LOCAL_MACHINE\SYSTEM\
   CurrentControlSet\
   Services\
   EventLog\
   Security\
   ...

-or-

HKEY_LOCAL_MACHINE\SYSTEM\
   CurrentControlSet\
   Services\
   EventLog\
   System\
   Service Control Manager
```

It then calls the LoadLibrary() function of the ParameterMessageFile. Then it calls FormatMessage() using "n" as the ID.

For example, suppose an event log entry has the source "Service Control Manager" and the description is "Failed to start the service due to the following error: %%245."

In the registry, you find:

```
HKEY_LOCAL_MACHINE\SYSTEM\
   CurrentControlSet\
   Services\
   EventLog\
   System\
   Service Control Manager
```

```
        EventMessageFile...
        ParameterMessageFile REG_SZ  kernel32.dll
        TypesSupported...
        ...
```

Therefore, you need to follow these steps:

1. Use LoadLibrary() with KERNEL32.DLL.

2. Call FormatMessage() using the module handle obtained in step 1 and a string ID of 245.

3. Replace %%245 in the description with the string obtained in step 2.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseMisc

How to Toggle the NUM LOCK, CAPS LOCK, and SCROLL LOCK Keys

PSS ID Number: Q127190
Authored 14-Mar-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.5, 3.51, and 4.0

SUMMARY

The documentation for SetKeyboardState() correctly says that you cannot use this API to toggle the NUM LOCK, CAPS LOCK, and SCROLL LOCK keys.

You can use keybd_event() to toggle the NUM LOCK, CAPS LOCK, and SCROLL LOCK keys.

MORE INFORMATION

The following sample program turns the NUM LOCK light on if it is off. The SetNumLock function defined here simulates pressing the NUM LOCK key, using keybd_event() with a virtual key of VK_NUMLOCK. It takes a boolean value that indicates whether the light should be turned off (FALSE) or on (TRUE).

The same technique can be used for the CAPS LOCK key (VK_CAPITAL) and the SCROLL LOCK key (VK_SCROLL).

Sample Code

```
/* Compile options needed:
*/

#include <windows.h>

void SetNumLock( BOOL bState )
{
   BYTE keyState[256];

   GetKeyboardState((LPBYTE)&keyState);
   if( (bState && !(keyState[VK_NUMLOCK] & 1)) ||
       (!bState && (keyState[VK_NUMLOCK] & 1)) )
   {
   // Simulate a key press
      keybd_event( VK_NUMLOCK,
                   0x45,
                   KEYEVENTF_EXTENDEDKEY | 0,
                   0 );

   // Simulate a key release
      keybd_event( VK_NUMLOCK,
```

```
                    0x45,
                    KEYEVENTF_EXTENDEDKEY | KEYEVENTF_KEYUP,
                    0);
   }
}

void main()
{
   SetNumLock( TRUE );
}
```

Additional reference words: 3.50 4.00 95
KBCategory: kbprg kbcode
KBSubcategory: UsrInp

How to Update the List of Files in the Common Dialogs

PSS ID Number: Q109696
Authored 06-Jan-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, version 3.5

SUMMARY

Sometimes it is necessary to update the list of files, without terminating the dialog box, when using the File Open or Save As common dialog box. This can be done by simulating a double-click on the list box of directories. Although the message can be posted from any application, a hook procedure should be used to post the message to the dialog box window.

MORE INFORMATION

The common dialog box functions that update the list of files and directories are internal to the common dialog boxes and are not accessible by applications using the common dialog box routines. The functions are invoked and the list boxes are updated only when the user double-clicks a list box.

Sample Code

The following code uses the Cancel button of the common dialog boxes to update the list boxes:

```
BOOL CALLBACK __export FileOpenHook (HWND hDlg, UINT message,
                        WPARAM wParam, LPARAM lParam)
{
   switch (message)
   {
      case WM_COMMAND:
         switch(wParam)
         {
         // This simulates a double-click on the list of directories,
         // effectively forcing the common dialogs to re-read the current
         // directory of files and to refresh the list of files.
            case IDCANCEL :
               PostMessage( hDlg, WM_COMMAND, lst2,
                            MAKELPARAM(GetDlgItem(hDlg, lst2), LBN_DBLCLK);
               return TRUE;
         }
      break;
   }
   return FALSE;
}
```

If the application targets Win32, the notification message to the list box is sent differently; here is the PostMessage for Win32 applications:

```
PostMessage (hDlg, WM_COMMAND, MAKEWPARAM (lst2,LBN_DBLCLK),
          (LPARAM)GetDlgItem (hDlg, lst2));
```

Applications using the IDs of the common dialog box's controls must include the DLGS.H file.

The templates for the common dialog boxes are in the \SAMPLES\COMMDLG directory or in the \INCLUDE directory of the Windows SDK installation.

Additional reference words: 3.10 3.50 refresh redraw
KBCategory: kbprg
KBSubcategory: UsrCmnDlg

How to Use a DIB Stored as a Windows Resource

PSS ID Number: Q67883
Authored 28-Dec-1990 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

Device-independent bitmaps (DIBs) are a very useful tool for displaying graphic information in a variety of device environments. With the appropriate device drivers, Windows can display a DIB with varying results on any video display or on a graphics printer.

This article discusses the differences between two methods that can be used to access a DIB from a resource.

MORE INFORMATION

Bitmaps retrieved from resources are very similar to those stored in .BMP files on disk. The header information is the same for each type of bitmap. However, depending upon the method used to retrieve the bitmap from the resource, the bitmap may be a device-independent bitmap (DIB) or a device-dependent bitmap (DDB).

When the LoadBitmap() function is used to obtain a bitmap from a resource, the bitmap is converted to a DDB. Typically, the DDB will be selected into a memory device context (DC) and blt'ed to the screen for display.

NOTE: If a 256-color bitmap with a palette is loaded from a resource, some colors will be lost. To display a bitmap with a palette correctly, the palette must be selected into the destination DC before the image is transferred to the DC. LoadBitmap() cannot return the palette associated with the bitmap; therefore, this information is lost. Instead, the colors in the bitmap are mapped to colors available in the default system palette, and a bitmap with the system default color depth is returned.

For example, if LoadBitmap() loads a 256-color image into an application running on a VGA display, the 256 colors used in the bitmap will be mapped to the 16 available colors, and a 4 bits-per-pixel bitmap will be returned. When the display is a 256-color 8514 unit, the same action will map the 256 bitmap colors into the 20 reserved system colors, and an 8 bits-per-pixel bitmap will be returned.

If, instead of calling LoadBitmap(), the application calls FindResource() (with RT_BITMAP type), LoadResource(), and LockResource(), a pointer to a packed DIB will be the result. A packed DIB is a BITMAPINFO structure

followed by an array of bytes containing the bitmap bits.

NOTE: If the resource was originally stored as a DDB, the bitmap returned will be in the DDB format. In other words, no conversion is done.

The BITMAPINFO structure is a BITMAPINFOHEADER structure and an array of RGBQUADs that define the colors used in the DIB. The pointer to the packed DIB may be used in the same manner as a bitmap read from disk.

NOTE: The BITMAPFILEHEADER structure is NOT present in the packed DIB; however, it is present in a DIB read from disk.

REFERENCES

For sample code demonstrating how to use FindResource() with RT_BITMAP, LoadResource(), and LockResource(), please see the following article in the Microsoft Knowledge Base:

   ARTICLE-ID: Q124947
   TITLE   : Retrieving Palette Information from a Bitmap Resource

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiBmp

How to Use a Program to Calculate Print Margins

PSS ID Number: Q122037
Authored 25-Oct-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The Windows Software Development Kit (SDK) does not provide a function to calculate printer margins directly. An application can calculate this information using a combination of printer escapes and calls to the GetDeviceCaps() function in Windows or by using GetDeviceCaps() in Windows NT. This article discusses those functions and provides code fragments as illustrations.

MORE INFORMATION

An application can determine printer margins as follows:

1. Calculate the left and top margins

   a. Determine the upper left corner of the printable area by using the GETPRINTINGOFFSET printer escape in Windows or by calling GetDeviceCaps() with the PHYSICALOFFSETX and PHYSICALOFFSETY indices in Windows NT. For example:

```
// Init our pt struct in case escape not supported
pt.x = 0; pt.y = 0;

// In Windows NT, the following 2 calls replace GETPRINTINGOFFSET:
// pt.x = GetDeviceCaps(hPrnDC, PHYSICALOFFSETX);
// pt.y = GetDeviceCaps(hPrnDC, PHYSICALOFFSETY);

// In Windows, use GETPRINTINGOFFSET to fill the POINT struct
// Drivers are not required to support the GETPRINTINGOFFSET escape,
// so call the QUERYESCSUPPORT printer escape to make sure
// it is supported.
Escape (hPrnDC, GETPRINTINGOFFSET, NULL, NULL, (LPPOINT) &pt);
```

   b. Determine the number of pixels required to yield the desired margin (x and y offsets) by calling GetDeviceCaps() using the LOGPIXELSX and LOGPIXELSY flags.

```
// Figure out how much you need to offset output. Note the
// use of the "max" macro. It is possible that you are asking for
// margins that are not possible on this printer. For example, the HP
// LaserJet has a 0.25" unprintable area so we cannot get margins of
```

```
      // 0.1".

      xOffset = max (0, GetDeviceCaps (hPrnDC, LOGPIXELSX) *
                        nInchesWeWant - pt.x);

      yOffset = max (0, GetDeviceCaps (hPrnDC, LOGPIXELSY) *
                        nInchesWeWant - pt.y);

   // When doing all the output, you can either offset it by the above
   // values or call SetViewportOrg() to set the point (0,0) at
   // the margin offset you calculated.

   SetViewportOrg (hPrnDC, xOffset, yOffset);
   ... all other output here ...
```

2. calculate the bottom and right margins

   a. Obtain the total size of the physical page (including printable and unprintable areas) by using the GETPHYSPAGESIZE printer escape in Windows or by calling GetDeviceCaps() with the PHYSICALWIDTH and PHYSICALHEIGHT indices in Windows NT.

   b. Determine the number of pixels required to yield the desired right and bottom margins by calling GetDeviceCaps using the LOGPIXELSX and LOGPIXELSY flags.

   c. Calculate the size of the printable area with GetDeviceCaps() using the HORZRES and VERTRES flags.

   The following code fragment illustrates steps 2a through 2c:

```
   // In Windows NT, the following 2 calls replace GETPHYSPAGESIZE
   // pt.x = GetDeviceCaps(hPrnDC, PHYSICALWIDTH);
   // pt.y = GetDeviceCaps(hPrnDC, PHYSICALHEIGHT);

   // In Windows, use GETPHYSPAGESIZE to fill the POINT struct
   // Drivers are not required to support the GETPHYSPAGESIZE escape,
   // so call the QUERYESCSUPPORT printer escape to make sure
   // it is supported.
   Escape (hPrnDC, GETPHYSPAGESIZE, NULL, NULL, (LPPOINT) &pt);

   xOffsetOfRightMargin = xOffset +
                          GetDeviceCaps (hPrnDC, HORZRES) -
                          pt.x -
                          GetDeviceCaps (hPrnDC, LOGPIXELSX) *
                          wInchesWeWant;

   yOffsetOfBottomMargin = yOffset +
                           GetDeviceCaps (hPrnDC, VERTRES) -
                           pt.y -
                           GetDeviceCaps (hPrnDC, LOGPIXELSY) *
                           wInchesWeWant;
```

NOTE: Now, you can clip all output to the rectangle bounded by xOffset, yOffset, xOffsetOfRightMargin, and yOffsetOfBottomMargin.

For further information about margins, query in the Microsoft Knowledge Base by using these words:

   GETPHYSPAGESIZE and GETPRINTINGOFFSET and GetDeviceCaps

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprint kbprg kbcode
KBSubcategory: GdiPrn

How to Use CTL3D Under the Windows 95 Operating System

PSS ID Number: Q130693
Authored 24-May-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SUMMARY

When an application that uses CTL3D is run under Windows 95, CTL3D disables itself if any dialog box has the DS_3DLOOK style. By default, all applications based on Windows version 4.0 get the DS_3DLOOK style for all dialog boxes. This article explains how this affects the way dialog boxes and controls are displayed under the Windows 95 operating system.

MORE INFORMATION

When CTL3D is disabled, Windows 95 draws dialog boxes and controls using its own 3D drawing properties. Windows 95 does not draw the static rectangles and frames in 3D as CTL3D does. For more information about how these frames and rectangles are drawn under Windows 95, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID: Q125684
TITLE : How to Use SS_GRAYRECT SS_BLACKRECT SS_WHITERECT in Windows 95

There are two new static control styles (SS_SUNKEN and SS_ETCHEDFRAME) in Windows 95 that simulate two of the static panels used in CTL3D. SS_SUNKEN creates a sunken panel, and SS_ETCHEDFRAME creates a panel with a dipped edge. There is no static style for creating a raised panel, but you can use the DrawEdge API to draw a raised panel.

There are also two new static control styles that you can use to create 3D lines. SS_ETCHEDHORZ creates a dipped horizontal line, and SS_ETCHEDVERT creates a dipped vertical line.

An application should check the platform version at run time by using the GetVersion or GetVersionEx function, and then implement appropriate 3D effects. If the major version is less than 4, the application can use the CTL3D functions, messages, and controls. If the major version is 4 or greater, the application should not implement CTL3D; it should create the proper Windows 95 style controls (or use DrawEdge to draw its 3D panels) to achieve the desired effects.

Additional reference words: 4.00
KBCategory: kbprg kbui
KBSubcategory: UsrCtl

How to Use DWL_MSGRESULT in Property Sheets & Wizard Controls

PSS ID Number: Q130762
Authored 25-May-1995 Last modified 22-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SUMMARY

Each page in a property sheet or wizard control is an application-defined modeless dialog box that manages the control windows used to view and edit the properties of an item. Applications provide the dialog box template used to create each page as well as the dialog box procedure.

A property sheet or wizard control sends notification messages to the dialog box procedure for a page when the page is gaining or losing the focus and when the user chooses the OK, Cancel, or other buttons. The notifications are sent in the form of WM_NOTIFY messages. The dialog box procedure(s) for the correspoding page(s) should use the SetWindowLong() function to set the DWL_MSGRESULT value of the page dialog box to specify the return value from the dialog box procedure to prevent or accecpt the change. After doing so, the dialog box procedure must return TRUE in response to processing the WM_NOTIFY message. If it does not return TRUE, the return value set in the DWL_MSGRESULT index using the SetWindowLong() function is ignored by the property sheet or wizard control.

MORE INFORMATION

Dialog box procedures return a BOOL value (TRUE or FALSE). This return value indicates to the caller of the dialog box function that the dialog box function either handled the message that it received or did not handle it. When the dialog box function returns FALSE, it is indicating that it did not handle the message it received. When the dialog box handles the message and generates a return value, it typically sets the DWL_MSGRESULT index of the dialog box with the return value.

The dialog box function of the property sheet or wizard page handles messages (WM_NOTIFY) sent by the property sheet or wizard control. The property sheet or wizard control determines whether the page that received the message processed the message or not by checking the retrun value from the call to SendMessage(). If the return value is FALSE, the control goes ahead and does what needs to be done by default. But if the return value is TRUE, the control checks for the return value by looking at the value stored in the DWL_MSGRESULT index of that page.

For example, the dialog box function of a property page might trap the PSN_SETACTIVE notification to prevent it from being activated under certain circumstances. In this case, the page dialog box function uses the SetWindowLong() function to set the DWL_MSGRESULT value to -1. If the dialog box does not return TRUE after setting the DWL_MSGRESULT, the

property sheet control that sent the message completely ignores the return value because it assumes there is no return value.

Additional reference words: 4.00 common controls user Windows 95
KBCategory: kbprg
KBSubcategory: UsrCtl

How to use ExitExecRestart to Install System Files

PSS ID Number: Q114606
Authored 08-May-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, version 3.5

SUMMARY

Some installation procedures require the installation of files (such as CTL3D.DLL, COMMDLG.DLL, and fonts) that may be in use by Windows at the time the setup program is run. Windows is likely to have these files open, so they cannot be installed without causing sharing violations. The Setup Toolkit provides features to exit Windows, install these files, and then restart Windows when complete.

MORE INFORMATION

The Setup Toolkit accomplishes the installation of system files as follows:

1. Before the Setup Toolkit copies a system file, it checks to see if the file is currently open. If it is, it copies the file to the destination directory, but under a different file name. It then adds this file to the "restart list".

2. When CopyFilesInCopyList() is complete, the Setup Toolkit checks the "restart list" and generates a .BAT file (named _MSSETUP.BAT) in the "restart directory". This .BAT file contains commands which delete the system files which were open (in step #1) and rename the new versions to their correct names.

3. Windows is exited, the .BAT file executed, then Windows restarted.

4. The .BAT file is then deleted.

   NOTE: The "restart directory" is not deleted. Hence, you should use your application's installation directory as your restart directory.

Hence, to install system files, perform the following steps:

1. Mark the system files as "system" in the DSKLAYT program. This is accomplished by highlighting all the system files (clicking with the CTRL key down) and placing a check in the "System File" check box under "File Attributes".

2. Before calling CopyFilesInCopyList() specify the name of your "restart directory". Assuming the target directory for your application is stored in DEST$ (as in the samples), use the following line:

```
SetRestartDir DEST$
```

The specified directory does not need to exist. It will be created if necessary.

3. After your installation is complete, execute the following code before exiting your setup script. Normally this code will be placed at the end of the Install subroutine.

```
if RestartListEmpty ()=0 then
    '' The following two lines must go on one line.
    MessageBox hwndFrame (), "Windows will now be exited and
    restarted.", "Sample Setup Script", MB_OK+MB_ICONINFORMATION
eer:
    i%=ExitExecRestart ()
    '' The following three lines must go on one line.
    MessageBox hwndFrame (), "Windows cannot be restared because
    MS-DOS-based applications are active. Close all MS-DOS-based
    applications, and then click OK.", "Sample Setup Script",
    MB_OK+MB_ICONSTOP
    goto eer
end if
```

NOTE: In order to use the MessageBox() function you must add the following lines at the beginning of your setup script:

```
const MB_ICONINFORMATION = 64
'' The following two lines must go on one line.
declare sub MessageBox lib "user.exe" (hwnd%, message$,
title$, options%)
```

4. Add the file _MSSETUP.EXE to your source directory and lay it out on Disk #1 in DSKLAYT.

5. Add a reference to _MSSETUP.EXE to the [files] section of your .LST file. For example,

if you marked _MSSETUP.EXE to be compressed,

```
[files]
    _mssetup.ex_ = _mssetup.exe
```

if you did not mark it as compressed,

```
[files]
    _mssetup.exe = _mssetup.exe
```

NOTES:

1. If ExitExecRestart () is successful, your script will be exited. That is, ExitExecRestart () will not return. If it does return, an error has occurred.

2. This functionality is not available under Windows 3.0. If the user runs

the above setup script on Windows 3.0, they will receive the message that MS-DOS-based applications are running and they will not be able to complete the setup. If this is a concern, check the version of Windows before executing the above code.

3. If _MSSETUP.EXE is not in your .LST file or not laid out in DSKLAYT, you will receive an "assertion failure" message when calling ExitExecRestart ().

Additional reference words: 3.10 3.50 setup toolkit mssetup
KBCategory: kbtool kbsetup kbprg kbcode
KBSubcategory: TlsMss

How to Use Hangeul (Korean) Windows Input Method Editor (IME)

PSS ID Number: Q130053
Authored 10-May-1995 Last modified 16-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 Software Development Kit (SDK) version 3.5
- Microsoft Win32s version 1.2

SUMMARY

Hangeul (Korean) Windows supports the KSC5601-1987 code set, which consists of several thousands Hangeul characters. The Hangeul Windows IME (Input Method Editor) allows the user to enter Hangeul Jamos (24 basic components of Hangeul characters), compose the final characters, and send them to applications.

MORE INFORMATION

When running Hangeul Windows, you will see a small window with three buttons in the lower-left corner:

- The left button toggles between the Roman "A" character and the Korean character that is pronounced "GA," which Windows uses to symbolize Korean characters.

- The center button is a graphic of a half box or full box, which selects SBCS or DBCS storage (or display) of Roman (but not Hangeul) characters. Hangeul characters always take up a double-byte space, unlike some Japanese characters (katakana).

- The right button is the "Hangeul to Hanja" converter.

You can type in English by having the Roman toggle selected. To try typing Hangeul, the main thing to remember is that there is no apparent logical connection between the US 101 keyboard and what the you end up typing. To type like a pro, you need Korean keycaps, a cheat sheet, or a lot of memory in your brain.

For example, to type the word "Hangeul" type these characters:

```
   GKS   RMF
```

(Please ignore the spaces between the two characters.) Notice how each group of three keyboard characters assemble a single Hangeul character. Hangeul is often made up of three components (called "Jamos"), but characters can actually be composed of from two to several Jamos.

Here is how to type "Seoul, Korea." Seoul is pronounced locally "sa-ul,"

so try typing these characters:

   TJ DNF ZH FL DK

(Please ignore the spaces between the characters.) The word "Korea" is not a Korean name; it is the English equivalent, just as Japan is really Nippon. The word "Korea" in Korean is "Han-guk," so "Seoul, Hanguk" would be:

   TJD NF   GKS RNR

The reason to try "Hanguk" instead of "Korea" is that Hanguk can also be spelled with Chinese characters. Put the mouse pointer (cursor) on the left edge of "Han" and click the "Hangeul to Hanja" button. A list box appears. Select choice #1 by typing 1 or by selecting it with the mouse. Do the same for the next character, again selecting choice #1. Then it will display in Hanja.

Most Korean text is in Hangeul not Hanja, notice the 3.1-H readme is all Hangeul. Hanja is still used in Korea - sometimes.

Additional reference words: 1.20 3.10 3.50 Hangul kbinf
KBCategory: kbother
KBSubcategory: wintldev

How to Use LVIF_DI_SETITEM on an LVN_GETDISPINFO Notification

PSS ID Number: Q131285
Authored 07-Jun-1995 Last modified 10-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.51, 4.0
- Microsoft Win32s version 1.3

SUMMARY

Windows 95 provides two flags, LVIF_DI_SETITEM and TVIF_DI_SETITEM, for the listview and treeview controls respectively. When set, these flags instruct Windows to start storing information for that particular item previously set as a callback item.

MORE INFORMATION

Windows 95 introduces the concept of callback items for the new listview and treeview common controls. A callback item is a listview or treeview item for which the application, not the control, stores the text, icon, or any appropriate information about the item. If the application already maintains this information anyway, setting up callback items could decrease the memory requirements of the control. Callback items are just as useful for items that display constantly changing information. Setting these items up as callback items allows the application to display the most current values appropriate for that item.

Take for example a SPY application that displays information in a hierarchical form (or a treeview) about the window being browsed or spied on. One of the things it displays is the window rectangle, or the dimensions of the window.

Because the user could resize this window at any time, this particular item is a good candidate for a callback item because it displays constantly changing information.

The application defines a callback item by specifying LPSTR_TEXTCALLBACK for the pszText member of the TV_ITEM structure. Whenever the item needs to be displayed, Windows requests the callback information by sending the treeview's parent a TVN_GETDISPINFO notification in the form of a WM_NOTIFY message. The parent window then fills the pszText member of the TV_ITEM structure as the following sample code demonstrates:

```
   LRESULT MsgNotify(HWND   hwnd,
                     UINT   uMessage,
                     WPARAM wparam,
                     LPARAM lparam)
   {
       TV_DISPINFO *ptvdi = (TV_DISPINFO *)lparam;
```

```
      switch (ptvdi->hdr.code)
      {
         case TVN_GETDISPINFO:

           if (ptvdi.mask & TVIF_TEXT)
           {
              RECT  rect;
              char  szBuf [30];

              GetWindowRect (hWndToBrowse, &rect);

              // where FormatRectText formats the rect information
              // in a nice <WindowRect: (x,y):cx,cy> format
              // and stores it in szBuf.
              FormatRectText (&rect, szBuf, sizeof (szBuf));

              lstrcpy (ptvdi.pszText, szBuf);
           }
           :

          default: break;

       }
   return 0;
   }
```

At a certain point, the application may determine during run time, that the window dimensions will no longer change. At this point, there may be no reason for this particular treeview item to remain as a callback item. This time, you need to process the TVN_GETDISPINFO message in a slightly different manner, specifying the TVIF_DI_SETITEM flag as demonstrated in the following code:

```
   case TVN_GETDISPINFO:

      if (ptvdi.mask & TVIF_TEXT)
      {
         RECT  rect;
         char  szBuf [30];

         GetWindowRect (hWndToBrowse, &rect);

        // where FormatRectText formats the rect information
        // in a nice <WindowRect: (x,y):cx,cy> format
        // and stores it in szBuf.
         FormatRectText (&rect, szBuf, sizeof (szBuf));

         lstrcpy (ptvdi.pszText, szBuf);
         plvdi->item.mask = plvdi->item.mask | TVIF_DI_SETITEM;
      }
```

By ORing the mask with TVIF_DI_SETITEM, you instruct Windows to start storing text information for the particular treeview item. At that point, the application stops receiving a TVN_GETDISPINFO notification whenever the item needs to be redrawn. This works almost as well as calling

TreeView_SetItem() on the item and replacing pszText's value with LPSTR_TEXTCALLBACK to the appropriate string.

The same holds true for listview controls when the mask is ORed with the LVIF_DI_SETITEM flag. However, note that setting the LVIF_DI_SETITEM flag for listviews works only for the first column of text (iSubItem ==0).

If an application specifies LPSTR_TEXTCALLBACK therefore for a column other than 0 in report view, LVIF_DI_SETITEM does not store the text information for that listview item column.

For more information on other members of the LV_ITEM and TV_ITEM structures that can be set up for callback, refer to the documentation on LV_DISPINFO and TV_DISPINFO structures.

Additional reference words: 4.00 1.30 I_IMAGECALLBACK win95 I_CHILDRENCALLBACK
KBCategory: kbprg kbcode
KBSubcategory: UsrCtl

How to Use PeekMessage() Correctly in Windows

PSS ID Number: Q74042
Authored 11-Jul-1991 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

In the Windows environment, many applications use a PeekMessage() loop to perform background processing. Such applications must allow the Windows system to enter an idle state when their background processing is complete. Otherwise, system performance, "idle-time" system processes such as paging optimizations, and power management on battery-powered systems will be adversely affected.

While an application is in a PeekMessage() loop, the Windows system cannot go idle. Therefore, an application should not remain in a PeekMessage() loop after its background processing has completed.

NOTE: The PeekMessage method described in this article is only needed if your application is a 32-bit application for Windows and is, for some reason, unable to create threads and perform background processing.

MORE INFORMATION

Many Windows-based applications use PeekMessage() to retrieve messages while they are in the middle of a long process, such as printing, repaginating, or recalculating, that must be done "in the background." PeekMessage() is used in these situations because, unlike GetMessage(), it does not wait for a message to be placed in the queue before it returns.

An application should not call PeekMessage() unless it has background processing to do between the calls to PeekMessage(). When an application is waiting for an input event, it should call GetMessage() or WaitMessage().

Remaining in a PeekMessage() loop when there is no background work causes system performance problems. A program in a PeekMessage() loop continues to be rescheduled by the Windows scheduler, consuming CPU time and taking time away from other processes.

In enhanced mode, the virtual machine (VM) in which Windows is running will not appear to be idle as long as an application is calling the PeekMessage function. Therefore, the Windows VM will continue to receive a considerable fraction of CPU time.

Many power management methods employed on laptop and notebook computers are based on the system going idle when there is no processing to do. An

application that remains in a PeekMessage() loop will make the system appear busy to power management software, resulting in excessive power consumption and shortening the time that the user can run the system.

In the future, the Windows system will make more and more use of idle time to do background processing, which is designed to optimize system performance. Applications that do not allow the system to go idle will adversely affect the performance of these techniques.

All these problems can be avoided by calling the PeekMessage() function only when there is background work to do, and calling GetMessage() or WaitMessage() when there is no background work to do.

For example, consider the following PeekMessage() loop. If there is no background processing to do, this loop will continue to run without waiting for messages, preventing the system from going idle and causing the negative effects described above.

```
   // This PeekMessage loop will NOT let the system go idle.

   for( ;; )
   {
      while (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
      {
         if (msg.message == WM_QUIT)
            return TRUE;

         TranslateMessage(&msg);
         DispatchMessage(&msg);
      }

      BackgroundProcessing();
    }
```

This loop can be rewritten in two ways, as shown below. Both of the following PeekMessage() loops have two desirable properties:

- They process all input messages before performing background processing, providing good response to user input.

- The application "idles" (waits for an input message) when no background processing needs to be done.

Improved PeekMessage Loop 1

```
   // Improved PeekMessage() loop

   for(;;)
   {
      while (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
      {
         if (msg.message == WM_QUIT)
            return TRUE;
```

```
         TranslateMessage(&msg);
         DispatchMessage(&msg);
      }

      if (IfBackgroundProcessingRequired())
         BackgroundProcessing();
      else
         WaitMessage(); // Will not return until a message is posted.
   }
```

Improved PeekMessage Loop 2

```
   // Another improved PeekMessage() loop

   for (;;)
   {
      for (;;)
      {
         if (IfBackgroundProcessingRequired())
         {
            if (!PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
               break;
         }
         else
            GetMessage(&msg, NULL, 0, 0, 0);

         if (msg.message == WM_QUIT)
            return TRUE;

         TranslateMessage(&msg);
         DispatchMessage(&msg);
      }
      BackgroundProcessing();
   }
```

Note that calls to functions such as IsDialogMessage() and TranslateAccelerator() can be added to these loops as appropriate.

There is one case in which the loops above need additional support: if the application waits for input from a device (for example, a fax board) that does not send standard Windows messages. For the reasons outlined above, a Windows-based application should not use a PeekMessage() loop to continuously poll the device. Rather, implement an interrupt service routine (ISR) in a dynamic-link library (DLL). When the ISR is called, the DLL can use the PostMessage function to inform the application that the device requires service. DLL functions can safely call the PostMessage() function because the PostMessage() function is reentrant.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMsg

How to Use RPC Under Win32s

PSS ID Number: Q127903
Authored 21-Mar-1995 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Win32s, versions 1.2 and 1.25a

Win32s does not thunk Remote Procedure Call (RPC) calls. Therefore, if you have a Win32-based application that uses RPC and you want it to run on Win32s, you will need to write a thunking layer for your application to thunk the 32-bit calls to the 16-bit RPC implementation.

One issue to keep in mind when writing the thunking layer is how to handle stub code. It is convenient to direct the MIDL compiler to produce 16-bit stub code for the application. The stub code can be built as a 16-bit DLL, which can be called from the Win32-based application via the Universal Thunk. This eliminates the need to write a thunking layer for the RPC run-time functions that appear in 32-bit client stub code.

One limitation is that RPC servers are not supported with the 16-bit RPC, only clients. Microsoft's 32-bit RPC implements servers by spawning threads for each call, and threads are not supported under Win32s. There may be other vendors who supply a 16-bit RPC that supports servers.

Additional reference words: 1.20
KBCategory: kbprg
KBSubcategory: W32s

How to Use SS_GRAYRECT SS_BLACKRECT SS_WHITERECT in Windows

PSS ID Number: Q125684
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

The colors used in the gray, white, and black rectangle static controls has changed in Windows 95. In previous versions of Windows, these colors were based on the system colors for windows. In Windows 95, these colors are based on the colors for 3D objects.

MORE INFORMATION

In Windows 95, the definitions for white, gray, and black rectangle static controls are as follows:

- SS_WHITERECT: Specifies a rectangle filled with the highlight color for three-dimensional display elements (for edges facing the light source). This is the same color retrieved by using GetSysColor() with COLOR_3DHILIGHT.

- SS_GRAYRECT: Specifies a rectangle filled with the shadow color for three-dimensional display elements (for edges facing away from the light source). This is the same color retrieved by using GetSysColor() with COLOR_3DSHADOW.

- SS_BLACKRECT: Specifies a rectangle filled with the Shadow color for three-dimensional display elements (for edges facing away from the light source). This is the same color retrieved by using GetSysColor() with COLOR_3DDKSHADOW. This is not the same color as COLOR_3DSHADOW. There are two shadow colors used on 3D objects.

In previous versions of Windows, the definitions for white, gray, and black rectangle static controls were as follows:

- SS_WHITERECT: Specifies a rectangle filled with the color used to fill window backgrounds. This color is white in the default Windows color scheme. This is the same color retrieved by using GetSysColor() with COLOR_WINDOW.

- SS_GRAYRECT: Specifies a rectangle filled with the color used to fill the screen background. This color is gray in the default Windows color scheme. This is the same color retrieved by using GetSysColor() with COLOR_BACKGROUND.

- SS_BLACKRECT: Specifies a rectangle filled with the color used to draw window frames. This color is black in the default Windows color scheme.

This is the same color retrieved by using GetSysColor() with COLOR_WINDOWFRAME.

Additional reference words: 4.00
KBCategory: kbprg
KBSubcategory: UsrCtl

How to Use the Small Icon in Windows 95

PSS ID Number: Q125682
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

In Windows 95, each application is associated with two icons: a small icon (16x16) and a large icon (32x32). The small icon is displayed in the upper-left hand corner of the application and on the taskbar.

MORE INFORMATION

Large and small icons are associated with an application by using the RegisterClassEx() function. This function takes a pointer to a WNDCLASSEX structure. The WNDCLASSEX structure is similar to the WNDCLASS structure except for the addition of the hIconSm parameter, which is used for the handle to the small icon. If no small icon is associated with an application, Windows 95 will use a 16x16 representation of the large icon.

NOTE: RegisterClassEx() is not currently implemented in Windows NT where it returns NULL.

The LoadIcon() function loads the large icon member of an icon resource. To load the small icon, use the new LoadImage() function as follows:

```
LoadImage(  hInstance,
            MAKEINTRESOURCE(<icon identifier>),
            IMAGE_ICON,
            16,
            16,
            0);
```

The small icon currently associated with the application will be displayed in the upper-left corner of the application's main window and on the task bar. Both the large and the small icon association can be changed at runtime by using the WM_SETICON message.

By default, the start menu will display the first icon defined in an application's resources. This can be changed through the start menu property sheets.

Explorer displays the first defined icon in an application's resources unless the application adds an entry to the registry under the program information called DefaultIcon or defines an icon handler shell extension for the file type. Refer to the Shell Extension documentation for more information on shell extensions.

Additional reference words: 4.00
KBCategory: kbprg
KBSubcategory: UsrWndw

How to Use WinSock to Enumerate Addresses

PSS ID Number: Q129315
Authored 24-Apr-1995 Last modified 04-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.5 and 3.51

SUMMARY

The gethostbyname() and gethostname() WinSock database APIs can be used to list IP addresses for a multihomed host. However, these functions work only for IP addresses. This article shows by example how to give addresses for other address families. Two different methods are given.

MORE INFORMATION

Method One Code Sample

AF_IPX:

This function can be used to give an IPX address:

```
   #include <winsock.h>
   #include <wsipx.h>
   #include <wsnwlink.h>

   #include <stdlib.h>

   // Note:  In the interest of clarity, the following code does not check
   //        return values or handle error conditions.

   void IPXEnum()
   {
      int            cAdapters,
                     cbOpt  = sizeof( cAdapters ),
                     cbAddr = sizeof( SOCKADDR_IPX );

      SOCKET         s;
      SOCKADDR_IPX   Addr;

      // Create IPX socket.
      s = socket( AF_IPX, SOCK_DGRAM, NSPROTO_IPX );

      // Socket must be bound prior to calling IPX_MAX_ADAPTER_NUM
      memset( &Addr, 0, sizeof( Addr ));
      Addr.sa_family = AF_IPX;
      bind( s, (SOCKADDR*) &Addr, cbAddr);
```

```
      // Get the number of adapters => cAdapters.
      getsockopt( (SOCKET) s, NSPROTO_IPX, IPX_MAX_ADAPTER_NUM,
                  (char *) &cAdapters, &cbOpt );
      // At this point cAdapters is the number of installed adapters.
      while ( cAdapters > 0 )
      {
         IPX_ADDRESS_DATA  IpxData;

         memset( &IpxData, 0, sizeof(IpxData));

         // Specify which adapter to check.
         IpxData.adapternum = cAdapters - 1;
         cbOpt = sizeof( IpxData );

         // Get information for the current adapter.
         getsockopt( s, NSPROTO_IPX, IPX_ADDRESS,
                     (char*) &IpxData, &cbOpt );

         // IpxData contains the address for the current adapter.
         cAdapters--;
      }
   }
```

AF_NETBIOS:

This function uses the EnumProtocols() function to give lana numbers for the available NetBIOS transports. NOTE: This doesn't work under Windows NT 3.5 because of a bug in EnumProtocols(), but it does work under Windows NT 3.51.

```
   void NBEnum()
   {
      DWORD          cb = 0;
      PROTOCOL_INFO *pPI;
      BOOL           pfLanas[100];

      int            iRes,
                     nLanas = sizeof(pfLanas) / sizeof(BOOL);

      // Specify NULL for lpiProtocols to enumerate all protocols.

      // First, determine the output buffer size.
      iRes = EnumProtocols( NULL, NULL, &cb );

      // Verify the expected error was received.
      assert( iRes == -1 && GetLastError() == ERROR_INSUFFICIENT_BUFFER );
      if (!cb)
      {
         fprintf( stderr, "No available NetBIOS transports.\n");
         break;
      }

      // Allocate a buffer of the specified size.
      pPI = (PROTOCOL_INFO*) malloc( cb );
```

```
      // Enumerate all protocols.
      iRes = EnumProtocols( NULL, pPI, &cb );

      // EnumProtocols() lists each lana number twice, once for
      // SOCK_DGRAM and once for SOCK_SEQPACKET. Set a flag in pfLanas
      // so unique lanas can be identified.

      memset( pfLanas, 0, sizeof( pfLanas ));

      while (iRes > 0)
         // Scan protocols looking for AF_NETBIOS.
         if ( pPI[--iRes].iAddressFamily == AF_NETBIOS )
            // found one
            pfLanas[ pPI[iRes].iProtocol ] = TRUE;

      fprintf( stderr, "Available NetBIOS lana numbers: " );
      while( nLanas-- )
         if ( pfLanas[nLanas] )
            fprintf( stderr, "%d ", nLanas );

      free( pPI );
   }
```

AF_APPLETALK:

Address enumeration is not meaningful for AF_APPLETALK. On a multihomed host with routing disabled, only the default adapter is used. If routing is enabled, a single AppleTalk address is used for all installed network adapters.

Method Two: Code Sample

Listed below is an example of how to use the WinSock database APIs to give IP addresses:

```
   void EnumIP()
   {
      char     szHostname[100];
      HOSTENT *pHostEnt;
      int      nAdapter = 0;

      gethostname( szHostname, sizeof( szHostname ));
      pHostEnt = gethostbyname( szHostname );

      while ( pHostEnt->h_addr_list[nAdapter] )
      {
         // pHostEnt->h_addr_list[nAdapter] is the current address in host
         //  order.

         nAdapter++;
      }
   }
```

Additional reference words: 3.50

KBCategory: kbnetwork kbcode
KBSubcategory: NtwkWinsock

How Win32-Based Applications Are Loaded Under Windows

PSS ID Number: Q110845
Authored 31-Jan-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.0, 1.1, and 1.2

SUMMARY

Microsoft Win32s is an operating system extension that allows Win32-based applications to be run on Windows 3.1. Win32s consists of a VxD and a set of dynamic-link libraries (DLLs).

It is possible to distinguish whether an executable was built for Win32 or Win16. Win32 executables use the Portable Executable (PE) format, while Win16 executables use the New Executable (NE) format.

The Windows 3.1 loader was designed to be aware that Win32-based applications would potentially be loaded. When the user starts a Win32-based application, the Windows 3.1 loader tries to load the Win32-based application via WinExec(). WinExec() calls LoadModule(), which will fail with an error indicating that it was passed an .EXE with the PE format. At this point, WinExec() calls a special function to start up Win32s. This function loads W32SYS.DLL (16-bit DLL) via LoadModule(). If W32SYS determines that the EXE is indeed a valid PE file, it calls LoadModule() on WIN32S.EXE (16-bit EXE) (it is similar to WinOldApp for MS-DOS-based programs running in Windows). WIN32S.EXE contains the task database, PSP, task queue, and module database. WIN32S.EXE calls its only function to load the Win32s 16-bit translator DLL, W32S16.DLL, which does work as a translator between the Win32-based application and the 16-bit world that it is running in.

MORE INFORMATION

Win32-based applications are loaded in the upper 2 gigabytes (GB) of the 4 GB address space under Win32s, whereas Windows NT loads them in the lower 2 GBs. This is because W32S.386, a VxD, allocates the memory, and VxDs get memory in the 2 GB to 4 GB range. The first 64K and the last 64K cannot be read or written to (similar to a null page on other architectures).

On Windows 3.1, all applications, even the Win32-based applications, share the same address space, unlike Windows NT where each Win32-based application gets its own address space. Each Windows-based application may be given its own address space, starting with Windows NT 3.5.

Additional reference words: 1.00 1.10 1.20 G byte
KBCategory: kbprg
KBSubcategory: W32s

How Windows NT Handles Floating-Point Calculations

PSS ID Number: Q102555
Authored 04-Aug-1993 Last modified 21-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SUMMARY

Each thread in a Win32-based application has its own general register set (provided by the kernel) and its own floating-point state. However, the floating-point support available in a specific instance depends on the subsystem or component in question.

Below is a list of various components and environments and what type of floating-point support is present for them.

MORE INFORMATION

Win32

Floating-point support is available to any thread running in the Win32 or POSIX subsystem.

This is true even on x86 machines that do not have floating-point hardware. The floating-point instructions are emulated automatically for the thread.

This feature allows people writing code in higher level languages, such as C, to assume floating-point support. The compiler generates the appropriate floating-point machine code. In addition, some standard floating-point functions allow applications to use floating-point exceptions in a portable manner. [For more information, see the header file FPIEE.H and the filter function fpieee_flt().]

POSIX

Again, POSIX applications can be created with the assumption of available floating-point support.

Note that the POSIX standard does not define a way to enable floating-point exceptions, so POSIX applications that do so must rely on some system-specific features. Under Windows NT, a POSIX application can enable floating exceptions by using _controlfp(). Floating-point exceptions can then be caught by SIGFPE, or, if the application needs to do more than simply catch the exception, by fpieee_flt().

MS-DOS/WOW: x86

MS-DOS-based and Windows-based applications are run directly by the processor in virtual-86 mode. An MS-DOS/WOW application has access to the floating-point hardware just as it would appear in MS-DOS. (If no floating-point hardware is present, no emulation is provided for the application.)

MS-DOS/WOW: Non-x86

When run on a RISC-based computer, or other non-x86 machine using the Windows/NT 80286 emulation code, the 80287 floating-point instructions are directly emulated. The MS-DOS/WOW application behaves as if an 80287 processor were present.

OS/2 (Supported Only on x86 Versions of Windows NT)

Floating-point support in this subsystem matches that of OS/2: if there's no floating-point hardware installed, the OS/2 application is expected to provide its own emulation.

Device Drivers: x86

On x86 platforms, a driver cannot use the coprocessor. Moreover, using any floating-point instruction (including fnsave or fwait) in the driver could cause either corruption of the user's numeric state or a bug check.

Device Drivers: MIPS/Alpha

For non-ISR time (that is, execution that does not occur during the interrupt service routine [ISR]), floating-point support can be assumed. At ISR time, floating-point support is available so long as the driver registered the ISR with IoConnectInterrupt and passed FloatingSave as TRUE. This causes the system to save and restore the volatile floating-point registers around the associated ISR.

Note that these functions on RISC-based computers use floating-point: RtlFillMemory(), RtlZeroMemory(), RtlCopyMemory(), RtlMoveMemory(). If an ISR calls any of these functions, it must connect to the interrupt with FloatingSave=TRUE on a MIPS.

Additional reference words: 3.10
KBCategory: kbprg
KBSubcategory: BseMisc

Icons for Console Applications

PSS ID Number: Q91150
Authored 29-Oct-1992 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

Under OS/2, when adding an application named CONAPP.EXE to a program group, the system uses the file CONAPP.ICO (if it exists) as the icon. This does not happen automatically under Windows NT and Windows 95; the item will have a generic icon.

To specify the icon that appears in the program group, use the following steps:

1. Create a resource file containing an ICON statement:

```
ConApp ICON ConApp.ICO
```

2. Compile the resource using RC:

```
rc -r $(rcvars) -fo conapp.res conapp.rc
```

3. Add the .rc file to the link command line

MORE INFORMATION

If the application is started by clicking its icon in Program Manager, the icon that appears when the application is minimized will be that icon, whether it is a generic icon or an icon imbedded in the executable file.

If the application is started from the MS-DOS prompt or the File menu, then the icon that appears when the application is minimized will be the icon that is used for the MS-DOS prompt.

Additional reference words: 3.10 3.50 4.00
KBCategory: kbprg
KBSubcategory: BseCon

Ideas to Remember as You Convert from ASCII or ANSI to

PSS ID Number: Q130052
Authored 10-May-1995 Last modified 16-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 3.5
- Microsoft Win32s version 1.2

SUMMARY

Because the industry is moving to Windows NT, and Windows NT supports Unicode, many independent software vendors (ISVs) want to know how to convert existing ASCII (or ANSI) help and resource files into Unicode. This article gives you some notes on the subject.

MORE INFORMATION

When converting ASCII to Unicode, remember that the entire ASCII characters map perfectly to the first characters in Unicode. You need only add a second null bit and a 0x00 in the high byte.

When converting ANSI to Unicode, UCONVERT.EXE sources in the Win32 SDK and the good illustration of win32 MultiBytetoWideChar conversion API are helpful sources to consult.

When converting a Help file, you face the challenge presented by the fact that the old ANSI RTF format is clueless about wide characters. Also, you need an RTF format for each specific country. For example, you'll need one specific to Japan, another separate RTF format for Korea, and so on. In addition, you might want to consider converting to the newer, much more powerful Help file formats supported in the Windows 95 Help file system; this may be an easier solution.

Help files in Windows NT versions 3.1 and 3.5 use the same Help file format as Windows version 3.1 does. But newer operating systems such as Windows 95 contains a new help file compiler, engine, and format. One reason that Microsoft made this change was to improve international support.

One final idea you might want to consider is to develop your own converter by using the MutltiByteToWideChar() API. In fact, this may be the best approach, because application developers know exactly what kind of user interface they want to implement.

Additional reference words: 1.20 3.50 kbinf
KBCategory: kbother
KBSubcategory: wintldev

Identifying a Previous Instance of an Application

PSS ID Number: Q106385
Authored 07-Nov-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5 3.51, and 4.0

SUMMARY

The entry point of both Windows and Windows NT applications is documented to be:

```
int WinMain( hInstance, hPrevInstance, lpszCmdLine, nCmdShow )

HINSTANCE hInstance;        /* Handle of current instance  */
HINSTANCE hPrevInstance;    /* Handle of previous instance */
LPSTR lpszCmdLine;          /* Address of command line     */
int nCmdShow;               /* Show state of window            */
```

However, under Windows NT, hPrevInstance is documented to always be NULL. The reason is that each application runs in its own address space and may have the same ID as another application.

To determine whether another instance of the application is running, use a named mutex. If opening the mutex fails, then there are no other instances of the application running. FindWindow() can be used with the class and window name. However, note that a second instance of the application could be started, and could execute the FindWindow() call before the first instance has created its window. Use a named object to ensure that this does not happen.

MORE INFORMATION

The fact that hPrevInstance is set to NULL simplifies porting Win16 applications. Most 16-bit Windows-based applications contain the following logic:

```
if( !hPrevInstance )
   if( !InitApplication(hInstance) )
      return FALSE;
```

Under Windows, window classes only are registered by the first instance of an application. Consequently, if hPrevInstance is not NULL, then the window classes have already been registered and InitApplication() is not called.

Under Windows NT, because hPrevInstance is always NULL, InitApplication() is always called, and each instance of an application will correctly register its window classes.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMisc

Identifying the Versions of International Windows

PSS ID Number: Q130062
Authored 10-May-1995 Last modified 16-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 Software Development Kit (SDK) version 3.5
- Microsoft Win32s version 1.2

SUMMARY

This article suggests three ways to check for the language used in an international version of a Windows-based application.

MORE INFORMATION

Option One: Least Coding and Least Accurate

Check the "sCountry" entry under the [intl] section of the WIN.INI file by using the GetProfileString API. It is likely that this will match the Windows language version. For example, German Windows will probably have "Deutchland" and English Windows will probably have "United States" or "United Kingdom." However, because the user can change this setting by using the Control Panel, it is not always accurate.

Option Two: Most Coding and Most Accurate

Check the "deflang" entry under the [data] section of the SETUP.INF file. This is a three-letter language code that SETUP.EXE uses. The setting will be one of these:

```
English=ENU or ENG
Spanish=ESP
German=DEU
French=FRA or FRC (French Canadian)
Italian=ITA
```

The problem with this method is getting at the "deflang" entry in the SETUP.INF file. The applications should parse SETUP.INF. It's not that difficult, but it does involve extra coding.

Option Three: Let the User Choose

Suggest to the user what the application found, and let the user make final decision. Here's the algorithm:

```
   if Windows Version < 3.1
           Look at Win.ini, Setup.inf files
           Suggest a good guess and let the user choose;
           Register a profile string for your app;
   else
           Use version stamping;
```

For Windows version 3.1, the way to identify the character set is to use the version stamping API. The translation value from the GetFileVersionInfo when performed on GDI or SHELL.DLL is the only way in version 3.1 to find out the character set of the system. Please refer to the SDK documentation for more details on this API. Look for both GetFileVersionInfo and VERSIONINFO.

The VERSIONINFO statement creates a version-information resource. The resource contains information about the file such as its version number, its intended operating system, and its original filename. One of the parameters is langID, which specifies the language identifiers.

On Windows NT, look for the version resource information in one of the system DLLs, such as KERNEL32.DLL. Look at result returned by GetFileVersionInfo and its associated structures and references in the Win32 API help file or documentation. This should be correct on Windows NT version 3.5.

Additional reference words: 1.20 3.10 3.50 kbinf
KBCategory: kbother
KBSubcategory: wintldev

IME (Input Method Editor) Usage in Windows 95

PSS ID Number: Q118496
Authored 20-Jul-1994 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Japanese Windows 95

SUMMARY

IMEs (Input Method Editors) are dynamic-link libraries that allow users to type complex ideographic characters using a standard keyboard. IMEs are available in Asian versions of the Microsoft Windows operating system, and help minimize the effort needed by users to enter text containing characters from Unicode and double-byte character sets (DBCS). IMEs relieve users of the need to remember all possible character values. Instead, IMEs monitor the user's keystrokes, anticipate the character the user may want, and present a list of candidate characters from which to choose.

MORE INFORMATION

Asian versions of Windows 3.1 and Windows NT each have a separate IME application programming interface (API). However, these APIs have been merged into a single API for Windows 95.

Applications for Far East versions of Windows 95 can choose from three levels of IME support:

1. IME-unaware: Merely retrieves double-byte characters through two WM_CHAR messages.

2. IME-aware: Takes control of the IME module's default user interface and properly handles Kanji strings passed to it by the IME.

3. Fully IME-aware: Controls the entire process of composing characters, including the display of intermediate keystrokes, and can customize the IME user interface.

IME, by default, provides an IME window through which users enter keystrokes and view and select candidate characters. Applications developed for the WIN32 APIs can use the Input Method Manager(IMM) functions and messages to create and manage their own IME windows, providing a custom interface while using the conversion capabilities of the IME.

Additional reference words: 4.00 international IME IMM
KBCategory: kbother
KBSubcategory: WIntlDev

Impersonation Provided by ImpersonateNamedPipeClient()

PSS ID Number: Q101378
Authored 12-Jul-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, version 3.1

SUMMARY

The following information is from the Win32 application programming interface (API) "Programmer's Reference" in the section regarding ImpersonateNamedPipeClient():

BOOL ImpersonateNamedPipeClient(HANDLE hNamedPipe)

The ImpersonateNamedPipeClient function impersonates a named-pipe client application.

The level of impersonation can be specified by the client when the named pipe is opened. If the client does not explicitly specify a level, then the default is SecurityImpersonation.

MORE INFORMATION

Suppose there are three threads (A, B, and C) where:

A calls B

B calls C

B does a SecurityImpersonation of A

If A and B both specify dynamic tracking, then C can see the context of A when it makes a call on the pipe, as long as B impersonates A. Otherwise, C will see the context of A only if B was impersonating A when the pipe between B and C was connected.

Note that dynamic tracking is not supported between machines.

Additional reference words: 3.10
KBCategory: kbprg
KBSubcategory: BseSecurity

Implementing a "Kill" Operation in Windows NT

PSS ID Number: Q90749
Authored 21-Oct-1992 Last modified 02-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SUMMARY

The following sample demonstrates how to implement a "kill" operation, such as a UNIX ps/kill, under Windows NT. Note that PView will give you the PID you need.

The code sample makes use of the Win32 API TerminateProcess(). While TerminateProcess() does clean up the objects owned by the process, it does not notify any DLLs hooked to the process. Therefore, one can easily leave the DLL in an unstable state.

In general, the Task List is a much cleaner method of killing processes.

MORE INFORMATION

The following sample shows how to "kill" a process, given its process ID (PID).

Sample Code

```
#include <windows.h>
#include <stdio.h>

void ErrorOut(char errstring[30])
/*
Purpose: Print out an meaningful error code by means of
         GetLastError and printf.

Inputs:  errstring - the action that failed, passed by the
                     calling proc.

Returns: none

Calls:   GetLastError
*/

{
   DWORD Error;

   Error= GetLastError();
   printf("Error on %s = %d\n", errstring, Error);
```

```
}


void main(int argc, char *argv[])
{
   HANDLE hProcess;
   DWORD ProcId;
   BOOL TermSucc;

   if (argc == 2)
   {
      sscanf(agrv[1],"%x", &ProcId);
      hProcess= OpenProcess(PROCESS_ALL_ACCESS, TRUE, ProcId);
      if (hProcess == NULL)
         ErrorOut("OpenProcess");
      TermSucc= TerminateProcess(hProcess, 0);
      if (TermSucc == FALSE)
         ErrorOut("TerminateProcess");
      else
         printf("Process# %.0lx terminated successfully!\n", ProcId);
   }
   else
   {
      printf("\nKills an active Process\n");
      printf("Usage: killproc ProcessID\n");
   }
}
```

Additional reference words: 3.10
KBCategory: kbprg
KBSubcategory: BseProcThrd

Implementing a Line-Based Interface for Edit Controls

PSS ID Number: Q92626
Authored 11-Nov-1992 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, versions 3.50, 3.51, and 4.0

SUMMARY

In specific situations, it may be desirable to make multiline edit controls behave similar to list boxes, such that entries can be selected and manipulated on a per-line basis. This article describes how to implement the line-based interface.

MORE INFORMATION

A multiline edit control must be subclassed to achieve the desired behavior. The subclass function is outlined below.

Most of the work necessary to implement a line-based interface is done by the predefined window function of the edit control class. With the return value from the EM_LINEINDEX message, the offset of the line under the caret can be determined; with the length of that line retrieved via the EM_LINELENGTH message, the EM_SETSEL message can be used to highlight the current line.

There are two problems with this approach:

- The first problem is that the EM_LINEINDEX message, when sent to the control with wParam=-1, returns the line index of the caret, which is not necessarily the same as the current mouse position. Thus, upon receiving the WM_LBUTTONDOWN message, the subclass function should first call the old window function, which will set the caret to the character under the current mouse position, then compute the beginning and ending offsets of the corresponding line, and eventually set the selection to that line.

- The other problem is that the WM_MOUSEMOVE message should be ignored by the subclassing function because otherwise the built-in selection mechanism will change the selection when the mouse is being dragged with the left mouse button pressed, thus defeating the purpose.

Following is the subclassing function that follows from this discussion:

```
WNDPROC EditSubClassProc(HWND hWnd,
               UINT wMsg,
               WPARAM wParam,
               LPARAM lParam)
{ int iLineBeg, iLineEnd;
```

```
  long lSelection;
  switch (wMsg)
  { case WM_MOUSEMOVE:
     break;                        /* Swallow mouse move messages.  */
    case WM_LBUTTONDOWN:           /* First pass on, then process.  */
     CallWindowProc((FARPROC)lpfnOldEditFn,hWnd,wMsg,wParam,lParam);
     iLineBeg = SendMessage(hWnd,EM_LINEINDEX,-1,0);
     iLineEnd=iLineBeg+SendMessage(hWnd,EM_LINELENGTH,iLineBeg,0);
#ifndef WIN32
     SendMessage(hWnd,EM_SETSEL,0,MAKELPARAM(iLeneBeg,iLineEnd));
#else
     SendMessage(hWnd,EM_SETSEL,iLineBeg,iLine)  /* Win 32 rearranges
                      parameters.  */
#endif
     break;
    case WM_LBUTTONDBLCLK:
     lSelection = SendMessage(hWnd,EM_GETSEL,0,0);
     /* Now we have the indices to the beginning and end of the line in
        the LOWORD and HIWORD of lSelection, respectively.
        Do something with it... */
     break;
    default:

return(CallWindowProc((FARPROC)lpfnOldEditFn,hWnd,wMsg,wParam,lParam));
};
return(0);
}
```

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Importance of Calling DefHookProc()

PSS ID Number: Q74547
Authored 23-Jul-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

When an application installs a hook using SetWindowsHook(), Windows adds the hook's callback filter function to the hook chain. It is the responsibility of each callback function to call the next function in the chain. DefHookProc() is used to call the next function in the hook chain for Windows 3.0. DefHookProc() is retained in Windows 3.1 for backwards compatibility. For Windows 3.1, you should use CallNextHookEx() to call the next function in the hook chain.

For Win32, mouse and keyboard hooks can suppress messages by return value and do not have to call CallNextHookEx(), unless they want to pass the message on. Other hooks, like WH_CALLWNDPROC, don't need to call CallNextHookEx(), because it will be called by the system. However, all hooks should call CallNextHookEx() immediately if nCode<0.

MORE INFORMATION

Windows 3.0

If a callback function does not call DefHookProc(), none of the filter functions that were installed before the current filter will be called. Windows will try to process the message and this could hang the system.

Only a keyboard hook (WH_KEYBOARD) can suppress a keyboard event by not calling DefHookProc() and returning a 1. When the system gets a value of 1 from a keyboard hook callback function, it discards the message.

Windows 3.1

In Windows 3.1, the WH_MOUSE hook will work like the WH_KEYBOARD hook in that the mouse event can be suppressed by returning 1 instead of calling DefHookProc().

Furthermore, when the hook callback function receives a negative value for the nCode parameter, it should pass the message and the parameters to DefHookProc() without further processing. When nCode is negative, Windows is in the process of removing a hook callback function from the hook chain.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrHks

Improve System Performance by Using Proper Working Set Size

PSS ID Number: Q126767
Authored 01-Mar-1995 Last modified 03-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 3.5

SUMMARY

While increasing your working set size and locking pages in physical memory can reduce paging for your application, it can adversely affect the system performance. When making decisions for your application, it is important to consider the whole system, and then test your application under a heavily loaded system, such as the users of your application might have.

MORE INFORMATION

The Win32 SDK provides a tool called the working set tuner (WST.EXE). The working set tuner decreases the working set size, which decreases memory demand. However, you can also choose to set the process working set minimum and maximum using SetProcessWorkingSetSize() and/or lock pages into memory with VirtualLock(). These APIs should be used with care. Suppose you have a 16-megabyte system and you set your minimum to four megabytes. In effect, this takes away four megabytes from the system. Other applications may be unable to get their minimum working set. You or other applications may be unable to create processes or threads or perform other operations that require non-paged pool. This can have an extremely negative impact on the overall system.

Reducing memory consumption is always a beneficial goal. If you call SetProcessWorkingSetSize(0xffffffff, 0xffffffff), this tells the system that your working set can be released. This does not change the current sizing of the working set, it just allows the memory to be used by other applications. It is a good idea to do this when your application goes into a wait state. When you call SetProcessWorkingSet(0, 0), your working set is reset to the default values. In addition, if you call VirtualUnlock() on a range that was not locked, it is used as a hint that those pages can be removed from the working set.

Additional reference words: 3.50
KBCategory: kprg
KBSubcategory: BseMm

Improving the Performance of MCI Wave Playback

PSS ID Number: Q77700
Authored 23-Oct-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

This article discusses two methods to improve playback performance for a series of MCI wave files in an application developed for the Microsoft Windows environment.

MORE INFORMATION

The following code fragment demonstrates opening the device and wave file at the same time. This method does not give the best performance.

```
   mciopen.lpstrDeviceType = (LPSTR)"waveaudio";
   mciopen.lpstrElementName = (LPSTR)lpWavefile;

   // The following two fields must be initialized or the debugging
   // version of MMSYSTEM will cause an unrecoverable application
   // error (UAE).
   mciopen.lpstrDeviceType = "\0";
   mciopen.lpstrAlias = "\0";

   dwFlags = MCI_OPEN_TYPE | MCI_OPEN_ELEMENT;

   dwRes = mciSendCommand(0, MCI_OPEN, dwFlags,
                          (DWORD)(LPSTR)&mciopen);
```

To improve performance, open the device separately from the wave file (element) and leave the device open until the last element in the series has been played. Alternately, open and close elements but leave the global (waveaudio) device open during the entire process. The following code fragment demonstrates this process:

```
   // Open the waveaudio driver separate from the element.
   mciopen.lpstrDeviceType = (LPSTR)MCI_DEVTYPE_WAVEFORM_AUDIO;
   dwFlags = MCI_OPEN_TYPE;
   dwRes = mciSendCommand(0, MCI_OPEN, dwFlags,
                          (DWORD)(LPSTR)&mciopen);
```

The following code fragment demonstrates using the global device ID to open the wave file separately:

```
   dwFlags = MCI_OPEN_ELEMENT;
   mciopen.lpstrElementName = (LPSTR)lpWaveName;
```

```
   dwRes = mciSendCommand(wGlobalDeviceID, MCI_OPEN, dwFlags,
                          (DWORD)(LPSTR)&mciopen);
```

This allows the application to open and play wave files without incurring the performance penalty involved with opening the device. Another method to speed loading a wave file is to use the fully qualified path. For example, rather than specifying LASER.WAV, specify C:\MMWIN\MMDATA\LASER.WAV. If this is done, MCI is not required to search the directories in the MS-DOS PATH environment variable for the wave file.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbmm kbprg
KBSubcategory: MMWave

Increased Performance Using FILE_FLAG_SEQUENTIAL_SCAN

PSS ID Number: Q98756
Authored 13-May-1993 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

There is a flag for CreateFile() called FILE_FLAG_SEQUENTIAL_SCAN which will direct the Cache Manager to access the file sequentially.

Anyone reading potentially large files with sequential access can specify this flag for increased performance. This flag is useful if you are reading files that are "mostly" sequential, but you occasionally skip over small ranges of bytes.

MORE INFORMATION

The effect on the Cache Manager of this flag is two-fold:

- There is a minor savings because the Cache Manager dispenses with keeping a history of reads on the file, and tries to maintain a high-water mark on read ahead, which is always a certain delta from the most recent read.

- More importantly, the Cache Manager reads further ahead for sequential access files--currently about three times more than files that are currently detected for sequential access.

If the caller makes multiple passes through a file, there are no negative effects of specifying the sequential flag, because the Cache Manager will still disable read ahead for as long as the application is getting hits on the file (such as on the second or subsequent pass).

If you are working on an application where your ability to sequentially read file data is key to performance, you may want to consider adding the sequential flag to your create file call. This is especially true of applications that use this flag to read from a CD-ROM.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseFileio

Initializing Menus Dynamically

PSS ID Number: Q75630
Authored 26-Aug-1991 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

Many commercial applications developed for Windows allow the user to customize the menus of the application. This ability introduces some complexity when the application must disable particular menu items at certain times. This article provides a method to perform this task.

Windows sends a WM_INITMENUPOPUP message just before a pop-up menu is displayed. The parameters to this message provide the handle to the menu and the index of the pop-up menu on the main menu.

To process this message properly, each menu item must have a unique identifier. When the application starts up, it creates a mapping array that lists the items on each menu. When the WM_INITMENUPOPUP message is received, the application checks the conditions necessary for each menu item to be disabled or checked and modifies the menu appropriately.

The application must maintain the mapping array when the user modifies the menus in any way.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMen

Installing the Win32s NLS Files

PSS ID Number: Q124136
Authored 19-Dec-1994 Last modified 25-Apr-1995

The information in this article applies to:

- Microsoft Win32s version 1.2

SUMMARY

The help file for Win32s version 1.2 states that the following NLS files need to be installed in the <windows>\SYSTEM\WIN32S directory when you install Win32s version 1.2 with your application. However, if your Win32-based application is only targeting U.S. Windows, you do not need to install all these files.

.NLS files:

CTYPE.NLS
LOCALE.NLS
L_INTL.NLS
P_037.NLS
P_10000.NLS
P_10001.NLS
P_10006.NLS
P_10007.NLS
P_10029.NLS
P_10081.NLS
P_1026.NLS
P_1050.NLS
P_1051.NLS
P_1252.NLS
P_1053.NLS
P_1054.NLS
P_437.NLS
P_500.NLS
P_737.NLS
P_850.NLS
P_852.NLS
P_855.NLS
P_857.NLS
P_860.NLS
P_861.NLS
P_863.NLS
P_865.NLS
P_866.NLS
P_869.NLS
P_875.NLS
P_932.NLS
P_936.NLS
P_949.NLS
P_950.NLS
SORTKEY.NLS
SORTTBLS.NLS
UNICODE.NLS

MORE INFORMATION

Use the following guidelines when shipping a Win32-based application that targets Win32s:

Ship National Language Support (.NLS) files corresponding to the market of the Win32-based application. For a Win32-based application released for an international market, ship all the .NLS files found in MSTOOLS\WIN32S\NLS. For a Win32-based application released for use only in the United States, ship P_437.NLS, P_850.NLS, and P_1252.NLS.

NOTE: When installing Win32s, be sure to remove the obsolete files. These are tagged in WIN32S\SETUP\W32S.LYT with Win32sSystemObsoleteFiles.

Additional reference words: 1.20
KBCategory: kbusage kbpolicy kbdocerr
KBSubcategory: W32s

Instance-Specific String Handles (HSZs) in DDEML

PSS ID Number: Q94953
Authored 26-Jan-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

Instance-specific string handles in DDEML may be used in DdeConnect() or DdeConnectList() in order to connect to a particular instance of a server. These string handles are received by a DDEML callback as the HSZ2 parameter to the XTYP_REGISTER/XTYP_UNREGISTER transactions whenever a server application registers or unregisters the service name it supports.

This article explains how instance-specific HSZs are internally implemented in the Windows 3.1 DDEML; however, this is for purposes of illustration only because the implementation may change in future versions of DDEML, particularly in Win32. However, the behavior will be the same.

MORE INFORMATION

Currently, instance-specific string handles contain two pieces of information: the original service name string plus the handle to a hidden window created by DDEML, which is associated with that string. These two pieces of information are then merged [that is, MAKELONG (SvcNameAtom, hWnd)] into an HSZ.

It is important to underscore what the documents on DdeCreateStringHandle() say in reference to instance-specific HSZs (see the Comments section of the Windows 3.1 Software Development Kit (SDK) "Programmer's Reference, Volume 2: Functions," page 169):

> An instance-specific string handle is not mappable from string handle to string to string handle again. The DdeQueryString() function creates a string from a string handle and then DdeCreateStringHandle() creates a string handle from that string, but the two handles are not the same.

This might be better explained as follows:

1. Server registers itself:

```
0x0000C18F = DdeCreateStringHandle (,"SERVER",);
DdeNameService (,0x0000C18F,,);
```

2. Callbacks receive two HSZs in XTYP_REGISTER:

```
      HSZ1 = 0x0000C18F  (normal HSZ)
      HSZ2 = 0x56F8C18F  (instance-specific HSZ)
```

3. Client does a DdeQueryString() on the HSZ2 returned above, and creates a string handle with the string returned.

```
      DdeQueryString (,0x56F8C18F, myLpstr,,,);
            // where myLpstr returned = "SERVER:(56F8)"

      0x0000C193= DdeCreateStringHandle (,myLpstr,);
```

   Note how instance-specific 0x56F8C18F passed in to DdeQueryString() is not the same as the HSZ returned (0x0000C193) from the DdeCreateStringHandle() on the same string; whereas regular string handles (that is, non-instance-specific HSZs) would have mapped to the same string handle.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDde

Intergraph's NFS causes WinSock APIs to return error 10093

PSS ID Number: Q127015
Authored 08-Mar-1995 Last modified 13-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5

SYMPTOMS

The presence of Intergraph's NFS (Network File System) may cause Windows Sockets applications to function incorrectly. Specifically, if the 'Choose File' common dialog is displayed in a Windows Sockets application, subsequent Windows Sockets API calls may return error 10093, WSAENOTINITIALIZED.

STATUS

This problem has been corrected in an updated version of PC-NFS, version 2.0.8.0. The update is available from Intergraph's FTP server or bulletin board (IBBS): (205) 730-7248. For more information, please call Intergraph technical support at (800) 633-7248.

MORE INFORMATION

Steps to Reproduce Problem
---

1. Call WSAStartup(...).

2. Call any WinSock API function to verify that WSOCK32.DLL is initialized.

3. Call GetOpenFileName to display the 'Open File' common dialog.

4. Call any WinSock API function. If Intergraph's NFS implementation is installed, you may receive error 10093.

Additional reference words: 3.50 Windows Sockets
bug CFileDialog
KBCategory: kbprg
KBSubcategory: NtwkWinsock

International Versions of Windows 95

PSS ID Number: Q118495
Authored 20-Jul-1994 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

Microsoft Windows version 3.1 is available in 27 different language versions. Windows 95 will also be released in 27 different language versions, with possibly more versions following later.

MORE INFORMATION

There are three categories of the Windows 95 platform:

1. Western and Eastern European languages, plus Indonesian, which are based on single-byte character sets (SBCSs) and are written from left to right:

   English
   German
   French
   Spanish
   Swedish
   Dutch
   Italian
   Norwegian
   Danish
   Finnish
   Portuguese-Brazil
   Portuguese-Portugal
   Russian
   Czech
   Polish
   Hungarian
   Turkish
   Greek
   Basque
   Catalan
   Indonesian

2. Middle East languages, plus Thai, which are based on SBCSs from both right to left and left to right:

   Arabic
   Hebrew
   Thai

3. Far East languages, which are based on double-byte character sets (DBCSs) and are written from both left to right and, in certain types of applications, from top to bottom:

   Japanese
   Korean
   Simplified Chinese
   Traditional Chinese

Additional reference words: 4.00 versions international
KBCategory: kbother
KBSubcategory: WIntlDev

Interpreting Executable Base Addresses

PSS ID Number: Q101187
Authored 07-Jul-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

LINK.EXE and DUMPBIN.EXE (from Visual C++ 32-bit edition) can be used to dump the portable executable (PE) header of an executable file. Below is a fragment of a dump:

```
   7300 address of entry point
   7000 base of code
   B000 base of data
        ----- new -----
   10000 image base
```

The "image base" value of 10000 is the address where the program begins in memory. The value associated with "base of code," "base of data," and "address of entry point" are all offsets from the image base.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsMisc

Interprocess Communication on Windows NT, Windows 95, &

PSS ID Number: Q95900
Authored 01-Mar-1993 Last modified 22-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The following are some of the standard mechanisms available for interprocess communication (IPC): NetBIOS, mailslots, windows sockets (winsock), named pipes, anonymous pipes, semaphores, shared memory, and shared files. Other IPC mechanisms available on Microsoft systems include DDE, OLE, memory-mapped files, Windows messages, Windows atoms, the registration database, and the clipboard.

MORE INFORMATION

The table below denotes what platforms and subsystems provide which IPC mechanisms (this does not imply that all the mechanisms will interoperate between different subsystems):

Interprocess Communication Mechanisms

IPC Mechanism	WinNT	Win95	Win32s(1)	Win16(2)	MS-DOS(2)	POSIX	OS/2
DDE	YES	YES	YES	YES	NO	NO	NO
OLE	YES	NO	YES	YES	NO	NO	NO
OLE 2.0	YES	YES	NO	YES	NO	NO	NO
NetBIOS	YES	YES	YES	YES	YES	NO	YES
Named pipes	YES	YES(3)	YES(3)	YES(3)	YES(3)	YES(4)	YES
Windows sockets	YES(5)	YES	YES	YES(5)	NO	NO(6)	NO
Mailslots	YES	YES	YES(3)	NO	NO	NO	YES
Semaphores	YES	YES	NO	NO	NO	YES	YES
RPC	YES	YES(7)	YES(8)	YES	YES	NO	NO
Mem-Mapped File	YES	YES	YES	NO	NO	NO	NO
WM_COPYDATA	YES	YES	YES(9)	YES	NO	NO	NO

(1) Win32s an extension to Windows 3.1, which allows Win32-based applications to run under Windows 3.1. Win32s supports all the Win32 APIs, but only a subset provides functionality under Windows 3.1. Those APIs that are not functional return ERROR_CALL_NOT_IMPLEMENTED.

(2) This is technically not a subsystem.

(3) Cannot be created on Win16, Windows 95 and MS-DOS workstations, but can be opened.

(4) The POSIX subsystem supports FIFO queues, which do not interoperate with Microsoft's implementation of named pipes.

(5) Via the Windows sockets API.

(6) Currently BSD-style sockets are under consideration for the POSIX subsystem.

(7) Windows 95 supports the RPC 1 protocol only. The NetBios protocol is not supported. Namedpipe servers are not supported.

(8) Win32s version 1.1 provides network support through Universal Thunks.

(9) Under Win32s, WM_COPYDATA does not actually copy the data -- it only translates the pointers to the data. If the receiving application changes the buffer, then the data is changed for both applications.

Additional reference words: 3.10 3.50 4.00
KBCategory: kbprg
KBSubcategory: BseIpc

Interrupting Threads in Critical Sections

PSS ID Number: Q101193
Authored 07-Jul-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

If a thread enters a critical section and then terminates abnormally, the critical section object will not be released. Many components of the C Run-time library are not reentrant and use a resource locking scheme to maintain coherency in the multithreaded environment. Thus, a thread that has entered a C Run-time function, such as printf(), could deadlock all access (within that process) to printf() if it terminates abnormally.

This situation could arise if a thread is terminated with TerminateThread() while it holds a resource lock. If this occurs, any thread that tries to acquire that resource lock will become deadlocked.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseMisc

Inverting Color Inverts Palette Index, Not RGB Value

PSS ID Number: Q71227
Authored 10-Apr-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

Performing any bitwise logical operator on a color, such as inversion, does not modify the color's RGB value; it instead changes the index into the system palette. This applies also to the ROP codes associated with the blt functions (BitBlt, StretchBlt, and PatBlt) and in the SetRop2 function. For display devices with hardware palettes [generally, devices with fewer than 24 bits-per-pixel (BPP)], this can produce unexpected or undesirable results.

MORE INFORMATION

Suppose the system palette contained the following colors:

	Entry #							
Color	0	1	2	3	4	5	6	7
Red =	0	0x80	0	0	0x80	0x80	0	0x80
Green =	0	0	0x80	0	0x80	0	0x80	0x80
Blue =	0	0	0	0x80	0	0x80	0x80	0x80

	Entry #							
Color	8	9	A	B	C	D	E	F
Red =	0xC0	0xFF	0	0	0xFF	0xFF	0	0xFF
Green =	0xC0	0	0xFF	0	0xFF	0	0xFF	0xFF
Blue =	0xC0	0	0	0xFF	0	0xFF	0xFF	0xFF

The inversion of colors would look like this:
(Half = half intensity, Full = full intensity)

Color	Index	Inverse Color	Index
Black	0	White	F
Half Red	1	Full Cyan	E
Half Green	2	Full Magenta	D
Half Blue	3	Full Yellow	C
Half Yellow	4	Full Blue	B
Half Magenta	5	Full Green	A
Half Cyan	6	Full Red	9
Dark Gray	7	Light Gray	8

This obviously is not the desired effect. Inverting a full-intensity color such as red will not invert to full-intensity cyan; instead, it is inverted to half-intensity cyan.

This is also true for any logical operations performed on the bits of a bitmap, pen, or flood fill through ROP codes. All operations are done on the index of the color and not its RGB value.

Note that when using custom palettes on a palette capable device, the application does not have control over the precise mapping between logical palette indexes and system palette indexes. The results of bitwise logical operations are unpredictable in such a case.

The only way for an application to precisely control color mixing is to perform the operation on RGB values, then translate the RGB result back into the most appropriate palette index.

For example, one way to do this is to mix colors in a 24 BPP device-independent bitmap (DIB), then set the DIB bits into the device context (DC) again when finished. Another method is to query the RGB color of pixels to modify, do the mixing, and then use the SetPixel function to apply the change to the DC.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiPal

Joliet Specification for CD-ROM

PSS ID Number: Q125630
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

Content authors who are developing Windows 95 applications on CD-ROM should develop their titles according to the Joliet specification in order to incorporate Unicode file names and take full advantage of Windows 95 long file name support.

The Joliet specification complies with the ISO 9660:1988 standard. It is designed to resolve some of its restrictions and ambiguities including:

- Character Set limitations.

- File Name Length limitations.

- Directory Tree Depth limitations.

- Directory Name Format limitations.

- Unicode Character ambiguities.

Because the Joliet specification is ISO 9660 compliant, CD-ROM disks recorded according to the Joliet specification may continue to interchange data with non-Joliet systems. The designs for the System Use Sharing Protocol, Rock Ridge extensions for POSIX semantics, CD-XA System Use Area Semantics, and Apple's Finder Flags and Resource Forks all port in a straightforward manner to the Joliet specification. These protocols are not an integral part of the Joliet specification, however.

Support for Joliet is included in Windows 95 and will also be included in a future version of Windows NT. To obtain a copy of the Joliet specification, send email to mmdinfo@microsoft.com.

Additional reference words: 4.00 CD CD-ROM XA SUSP ROCKRIDGE LFN
KBCategory: kbprg
KBSubcategory: SubSys

Journal Hooks and Compatibility

PSS ID Number: Q106717
Authored 14-Nov-1993 Last modified 03-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

Journal hooks are used to record and play back events, such as keyboard and mouse events. Journal hooks are system-wide hooks that take control of all user input, and therefore should be used as little as possible.

MORE INFORMATION

Note that Windows NT does not ship with a Recorder application, as Windows 3.1 does. Therefore, it may be desirable to create an application that can play back macros recorded under Windows 3.1. However, there are a number of different problems with the Windows NT implementation of journaling that make it difficult to use macros recorded under Windows 3.1.

The EVENTMSG structures recorded under Windows 3.1 that hold keystrokes do not play back under Windows NT. They must be modified, because the journal playback hook parses a scan code out of the EVENTMSG structure differently than the Windows 3.1 journal record hook put it in the structure. Under Windows 3.1, paramH specifies the repeat count. Under Windows NT, there is no way to specify a repeat count; it is always assumed to be 1.

For more information on hooks, please see the Hooks Overview in Volume 1 of the Win32 "Programmer's Reference" and the article "Win32 Hooks" included in the MSDN CD #5.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: UsrHks

Jumping to a Keyword in the Middle of a Help Topic

PSS ID Number: Q94611
Authored 11-Jan-1993 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The following information was extracted from the Windows Help Authoring Guide available on the Microsoft Developer's Network CD-ROM:

Placing Keywords in the Topic

When the user goes to a topic by choosing a keyword from the Search dialog box, Help displays the selected topic in the main window, starting from the beginning of the topic. If the information related to the keyword is located in the middle or toward the end of the topic, the user may not be able to see the relevant information without scrolling the topic. This result may not be what you want.

If you want users to be able to go directly to relevant information within a topic (and see it without scrolling), you can place additional keywords with the information you want users to find. Keywords placed within a topic function as spot references (similar to context-string spot references) because they index specific locations, or "spots," within the topic. To access the spot-referenced material, users choose the keyword from the Search dialog box.

In the Search dialog box, all keywords appear the same. The user cannot tell the difference between keywords placed at the beginning of the topic and those placed elsewhere in the topic. However, when the user goes to the topic, Windows Help uses the location of the keyword footnote as a reference point. If the keyword footnote is located in the middle of the topic, Help displays the topic as if the middle location were the "top" of the topic.

NOTE:

Because you cannot insert a title footnote in the middle of a topic, any keywords that you place in the middle of the topic use the main topic title in the Search dialog box.

To define a keyword in the middle of the topic:

1. Place the insertion point where you want to define the keyword. A keyword inserted anywhere except the beginning of the topic is treated as a spot reference.

2. From the Insert menu, choose Footnote. The Footnote dialog box appears.

3. Type an uppercase K as the custom footnote mark, and then choose OK. A superscript K appears in the text window, and the insertion point moves to the footnote window.

4. Type the keyword(s) to the right of the K in the footnote window. Use only a single space between the K and the first keyword. Separate multiple keywords with a semicolon (;).

Additional reference words: 3.10 3.50 4.00 95 winhelp hc
KBCategory: kbprg
KBSubcategory: TlsHlp

LB_GETCARETINDEX Returns 0 for Zero Entries in List Box

PSS ID Number: Q97922
Authored 25-Apr-1993 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, and 4.0

To determine whether a multiple selection list box is empty or has no items to select, two messages are required. First, call LB_GETCOUNT to determine whether or not the list box is empty. If the list box is not empty, then use LB_GETCARETINDEX to determine the position of the caret.

If you want a list box to contain selections that remain after the focus goes elsewhere, Microsoft recommends using visible check marks next to the items in the list box. This method provides better visual feedback to the user than a selection bar.

Additional reference words: 3.50 3.51 4.00 95 listbox
KBCategory: kbprg
KBSubcategory: UsrCtl

Length of STRINGTABLE Resources

PSS ID Number: Q20011
Authored 17-Dec-1987 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

In order to find the length of a string in the STRINGTABLE you need to do a FindResource() and then a SizeofResource() to find the total size in bytes of the current block of 16 strings. Remember that STRINGTABLEs are stored specially; to FindResource() you will ask for RT_STRING as the Type, and the (string number / 16) + 1 as the name.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrRsc

Limit on the Number of Bytes Written Asynchronously

PSS ID Number: Q98893
Authored 18-May-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SUMMARY

There is a limit to the number of bytes that can be written with WriteFile() using asynchronous I/O (FILE_FLAG_OVERLAPPED specified). This limit depends on the size of your system.

Asynchronous (overlapped) I/O consumes system resources for a long time. For example, the memory used is locked in the process working set until the I/O completes. To limit the amount of system resources used asynchronously by an application, the system charges asynchronous I/O to the working set of the process requesting the I/O.

While the working set size is dynamically raised and lowered based on the load, there are minimum and maximum values. These values are based on system size: consider up to 12 megabytes (MB) a small system, between 12 MB and 19 MB a medium system, and greater than 19 MB a large system. Each process is guaranteed a minimum working set for performance reasons; about 120K for small systems, 160K for large systems, and 245K for large systems.

When system resources are heavily taxed, a process is confined to its maximum working set. Asynchronous I/Os may never cause you to exceed your maximum working set, because once you are allowed to initiate an asynchronous I/O, the page cannot be taken away if memory becomes tight. The maximum working set sizes are about 300K for a small system, 716K for a medium system, and 1.5 MB for a large system.

MORE INFORMATION

The following code can be used to experiment with the maximum number of bytes that can be written using asynchronous I/O. Simply change the line to vary the number of bytes that the code attempts to write:

```
   #define NBR_BYTE 700000
```

Sample Code

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <assert.h>
```

```
#include <windows.h>

#define NBR_BYTE 700000

int main(void)
{
   char        *c;
   HANDLE      hFile;
   DWORD       byteWrite;
   OVERLAPPED  overLap;
   DWORD       err;
   BOOL        result;

   c = malloc( NBR_BYTE );
   assert( c != NULL );

   overLap.hEvent = CreateEvent( NULL, FALSE, FALSE, "event1" );
   assert( overLap.hEvent );

   hFile = CreateFile( "test", GENERIC_WRITE, 0,  NULL, OPEN_ALWAYS,
                       FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED |
                       FILE_FLAG_WRITE_THROUGH,
                       NULL);

   if( hFile == INVALID_HANDLE_VALUE )
   {
      free( c );
      printf( "error opening file\n" );
      exit( 0 );
   }

   overLap.Offset     = 0;
   overLap.OffsetHigh = 0;
   result = WriteFile( hFile, c, NBR_BYTE, &byteWrite, &overLap );
   if( result == FALSE )
   {
      err = GetLastError( );
      if( err != ERROR_IO_PENDING )
      {
         free( c );
         printf( "Error: %d\n", GetLastError() );
         exit( 0 );
      }
   }

   free( c );

   return 0;
}
```

Additional reference words: 3.10 asynch
KBCategory: kbprg
KBSubcategory: BseFileio

Limitations of Overlapped I/O in Windows 95

PSS ID Number: Q125717
Authored 02-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

Windows 95 does not support overlapped operations on files, disk, pipes, or mail slots, but does support overlapped operations on serial and parallel communication ports. Non-overlapped file write operations are similar to overlapped file writes, because Windows 95 uses a lazy-write disk cache. Overlapped I/O can be implemented in a Win32-based application by creating multiple threads to handle I/O.

MORE INFORMATION

Asynchronous I/O on files, disk, pipes and mail slots is not implemented in Windows 95. If a Win32-based application running on Windows 95 attempts to perform asynchronous file I/O (such as ReadFile() with any value other than NULL in the lpOverlapped field) on any of these objects, the ReadFile() or WriteFile fails and GetLastError() returns ERROR_INVALID_PARAMETER (87).

Overlapped I/O to serial and parallel communication ports is fully supported in Windows 95. To implement overlapped I/O, call CreateFile() with the FILE_FLAG_OVERLAPPED flag set in the flags attribute.

Overlapped I/O on disk and files was not implemented in Windows 95, because the added performance benefits (which would only affect a certain class of I/O-intensive applications) were not judged to be worth the extra cost.

The system uses a lazy-write disk cache algorithm which automatically provides many of the benefits of overlapped writes. When a process writes data to a file, the data is written to the cache and then the write immediately returns to the calling process. Then, at some later time, the cache manager writes the data to disk. This is similar to behavior that is achieved with overlapped I/O. In the case of disk/file reads, many applications that need to read data do so because it is needed for further processing. Some applications benefit greatly by prefetching data from files while doing other work.

Although Windows 95 does not implement overlapped I/O, it is possible for Win32-based applications on Windows 95 to create additional threads for implementing an effect similar to overlapped file I/O. One way to implement this effect is to communicate with an I/O thread using a request packet mechanism. The thread can queue request packets from other threads and service them as it is able, signalling the other threads on the completion of each request with an event. Even though the application may be "waiting" on some I/O activity, it can still be responsive to the user since the

main, or user interface, thread is not blocking on I/O requests. However, the use of multiple threads will increase the amount of time spent in the kernel, which leads to inefficiencies, as compared with an operating system that supports overlapped I/O.

NOTE: Windows NT has a lazy-write disk cache as well. It has been found that a write-back cache is not as good as overlapped I/O, particularly when the data is much larger than the file cache or noncached I/O. One of the biggest benefits of overlapped I/O is that it allows you to quickly get lots of outstanding I/O posted to the disk controller, thereby keeping the disks busy with tagged command queueing in the controller. Using multiple threads is a reasonable substitute, but I/O may be serialized in the filesystem.

Additional reference words: 4.00 95 asynchronous overlapped
KBCategory: kbprg
KBSubcategory: BseFileio

Limitations of WINOLDAP's Terminal Fonts

PSS ID Number: Q117742
Authored 06-Jul-1994 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows, version 3.1
- Microsoft Win32 SDK, version 3.1

SUMMARY

The MS-DOS prompt window (WINOLDAP) uses a set of private "Terminal" fonts. Applications developed for the Windows Graphical Environment should not rely on these fonts.

MORE INFORMATION

WINOLDAP uses Terminal-like fonts to present its output in the Enhanced mode. The font resources are loaded when WINOLDAP is launched and are removed when it terminates. To verify this, run a ChooseFont common dialog box (such as the one in FONTEST sample in the SAMPLES directory of the Windows 3.1 SDK) when WINOLDAP is not running. Launch WINOLDAP and notice that "Terminal" fonts are added to the list of fonts in the common dialog box.

Developers whose applications benefit from Terminal-like fonts may try to use Terminal fonts out of convenience. However, it is important to realize that these fonts (DOSAPP.FON, EGA40WOA.FON, EGA80WOA.FON, CGA40WOA.FON, AND CGA80WOA.FON) are unlike the standard fonts in that they are shipped for use by WINOLDAP. The standard Windows fonts are guaranteed to be available in future versions of Windows; such guarantees do not apply to the Terminal fonts. The Terminal fonts have actually been altered from version 3.0 to version 3.1 of Microsoft Windows. A simple resolution to this lack of guarantee is to ship the fonts along with the application. These fonts, however, may not be shipped freely with third-party applications without permission from Microsoft.

The paragraph above may lead you to believe that applications running exclusively under Microsoft Windows version 3.1 can rely on the Terminal fonts for the applications' displays because all Windows-based applications, by default, have the above fonts installed. However, a bug in WINOLDAP denies applications that benefit: When the last WINOLDAP session exits, it removes the DOSAPP.FON module twice. Any application with the DOSAPP.FON module loaded then loses the fonts without prior notice.

Additional reference words: 3.10
KBCategory: kbprg
KBSubcategory: GdiFnt

Limiting the Number of Entries in a List Box

PSS ID Number: Q78241
Authored 10-Nov-1991 Last modified 16-May-1995

-
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

-

SUMMARY

Although there is no single message that restricts the number of entries (lines) allowed in a list box, the limit can be imposed through the use of subclassing.

MORE INFORMATION

The following code fragment is an excerpt from a subclassing function that can be used to restrict the number of entries in a list box to no more than the constant MAXENTRIES where the lpfnOldLBFn variable points to the original list box window procedure:

```
long FAR PASCAL SubClassFn(hWnd, message, wParam, lParam)
HWND hWnd;
unsigned message;
WORD wParam;
LONG lParam;
{
   int iCount;

   switch (message)
      {
   case LB_ADDSTRING:
   case LB_INSERTSTRING:
      iCount = SendMessage(hWnd, LB_GETCOUNT, 0, 0L);
      if (iCount > MAXENTRIES)
         { /* Insert action here to inform user of limit violation */
         break;
         }
      /* fall through if less entries than maximum */

   default:
      return CallWindowProc(lpfnOldLBProc, hWnd, message, wParam,
            lParam);
      }
}
```

Additional reference words: 3.00 3.10 3.50 4.00 95 list box
KBCategory: kbprg

KBSubcategory: UsrCtl

Limits on Overlapped Pipe Operations

PSS ID Number: Q115522
Authored 29-May-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

The Windows NT version 3.1 redirector allows only 17 outstanding overlapped pipe operations at any given time. The Windows NT 3.5 redirector does not have this restriction on overlapped pipe writes. However, the Windows NT 3.5 redirector allows only 17 outstanding overlapped pipe reads at any given time.

If the client uses overlapped I/O through the redirector, it is possible for the client to become deadlocked. You may need to increase the number of threads that the redirector uses for I/O; the same thing is true for the server. If your application is doing a lot of I/O, you can avoid this deadlock by creating extra threads and having them use non-overlapped I/O.

MORE INFORMATION

Increasing the workstation services MaxThreads parameter increases the number of kernel threads that the redirector will create, thus allowing more operations to be outstanding at any given time.

This parameter is located in:

```
HKEY_LOCAL_MACHINE\SYSTEM\
   CurrentControlSet\
   Services\
   LanmanWorkstation\
   Parameters\
   MaxThreads
```

The parameter can be set from 0 to 255 (the default is 17).

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseFileio

Line Continuation and Carriage Returns in String Tables

PSS ID Number: Q44385
Authored 12-May-1989 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

The RC compiler does not offer a line-continuation symbol for strings in string tables.

To force a carriage return into a long line of text, use one of the methods described below.

One method is to force the carriage return using \012\015. The following example demonstrates the use of \012\015 and should be considered to be on one continuous line:

```
STRINGTABLE
BEGIN
IDSLONGSTRING, "This is a long line of text so I would like \012\015
to force a carriage return."
END
```

For more information on this method, query in the Microsoft Knowledge Base on the following words:

```
STRINGTABLE WinLoadString
```

Another method of forcing a carriage return is to press ENTER and continue the line on the next line. The following example will force a carriage return after the word "like."

```
STRINGTABLE
BEGIN
IDSLONGSTRING, "This is a long line of text so I would like
to force a carriage return"
END
```

If you try to use the \n or other \ characters, the RC compiler will ignore them.

NOTE: There is a 255-character limit (per string) in a string table. For more information on this limit, please query on the following words in the Microsoft Knowledge Base:

```
STRINGTABLE length 255
```

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsRc

List All NetBIOS Names on a Lana

PSS ID Number: Q124960
Authored 17-Jan-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1 and 3.5

SUMMARY

You can get a list of NetBIOS names for a lana by using the Adapter Status NetBIOS request and using the "*" character as the call name. However, on Windows NT, this method lists only the names added by the current process.

If you want to list all of the NetBIOS names on the lana, use a unique local name as the call name. This method causes the Adapter Status to be treated as a remote call, which will disable the "filtering" of names added by other processes. The sample code below demonstrates this technique.

SAMPLE CODE

```
/* The following makefile may be used to build this sample:

 !include <ntwin32.mak>

 PROJ = test.exe
 DEPS = test.obj
 LIBS_EXT = netapi32.lib

 .c.obj:
   $(cc) /YX $(cdebug) $(cflags) $(cvars) $<

 $(PROJ) : $(DEPS)
   $(link) @<<
 $**
 -out:$@
 $(conlibs)
 $(conlflags)
 $(ldebug)
 $(LIBS_EXT)
 <<

*/

#include <windows.h>
#include <stdio.h>
#include <string.h>

/*
 * LANANUM and LOCALNAME should be set as appropriate for
 * your system
```

```
 */
#define LANANUM     0
#define LOCALNAME   "MAKEUNIQUE"

#define NBCheck(x)  if (NRC_GOODRET != x.ncb_retcode) { \
                        printf("Line %d: Got 0x%x from NetBios()\n", \
                               __LINE__, x.ncb_retcode); \
                    }


void MakeNetbiosName (char *achDest, LPCSTR szSrc);
BOOL NBAddName (int nLana, LPCSTR szName);
BOOL NBReset (int nLana, int nSessions, int nNames);
BOOL NBListNames (int nLana, LPCSTR szName);
BOOL NBAdapterStatus (int nLana, PVOID pBuffer, int cbBuffer,
                      LPCSTR szName);

void
main ()
{
    if (!NBReset (LANANUM, 20, 30))
        return;

    if (!NBAddName (LANANUM, LOCALNAME))
        return;

    if (!NBListNames (LANANUM, LOCALNAME))
        return;

    printf ("Succeeded.\n");
}

BOOL
NBReset (int nLana, int nSessions, int nNames)
{
    NCB ncb;

    memset (&ncb, 0, sizeof (ncb));
    ncb.ncb_command = NCBRESET;
    ncb.ncb_lsn = 0;            /* Allocate new lana_num resources */
    ncb.ncb_lana_num = nLana;
    ncb.ncb_callname[0] = nSessions;  /* max sessions */
    ncb.ncb_callname[2] = nNames;  /* max names */

    Netbios (&ncb);
    NBCheck (ncb);

    return (NRC_GOODRET == ncb.ncb_retcode);
}

BOOL
NBAddName (int nLana, LPCSTR szName)
{
    NCB ncb;
```

```
    memset (&ncb, 0, sizeof (ncb));
    ncb.ncb_command = NCBADDNAME;
    ncb.ncb_lana_num = nLana;

    MakeNetbiosName (ncb.ncb_name, szName);

    Netbios (&ncb);
    NBCheck (ncb);

    return (NRC_GOODRET == ncb.ncb_retcode);
}

/*
 * MakeNetbiosName - Builds a name padded with spaces up to
 * the length of a NetBIOS name (NCBNAMSZ).
 */
void
MakeNetbiosName (char *achDest, LPCSTR szSrc)
{
    int cchSrc;

    cchSrc = lstrlen (szSrc);
    if (cchSrc > NCBNAMSZ)
        cchSrc = NCBNAMSZ;

    memset (achDest, ' ', NCBNAMSZ);
    memcpy (achDest, szSrc, cchSrc);
}

BOOL
NBListNames (int nLana, LPCSTR szName)
{
    int cbBuffer;
    ADAPTER_STATUS *pStatus;
    NAME_BUFFER *pNames;
    int i;

    // Allocate the largest buffer we might need
    cbBuffer = sizeof (ADAPTER_STATUS) + 255 * sizeof (NAME_BUFFER);
    pStatus = (ADAPTER_STATUS *) HeapAlloc (GetProcessHeap (), 0,
                                            cbBuffer);
    if (NULL == pStatus)
        return FALSE;


    if (!NBAdapterStatus (nLana, (PVOID) pStatus, cbBuffer, szName))
    {
        HeapFree (GetProcessHeap (), 0, pStatus);
        return FALSE;
    }

    // The list of names immediately follows the adapter status
    // structure.
    pNames = (NAME_BUFFER *) (pStatus + 1);
```

```
    for (i = 0; i < pStatus->name_count; i++)
        printf ("\t%.*s\n", NCBNAMSZ, pNames[i].name);

    HeapFree (GetProcessHeap (), 0, pStatus);

    return TRUE;
}

BOOL
NBAdapterStatus (int nLana, PVOID pBuffer, int cbBuffer, LPCSTR szName)
{
    NCB ncb;

    memset (&ncb, 0, sizeof (ncb));
    ncb.ncb_command = NCBASTAT;
    ncb.ncb_lana_num = nLana;

    ncb.ncb_buffer = (PUCHAR) pBuffer;
    ncb.ncb_length = cbBuffer;

    MakeNetbiosName (ncb.ncb_callname, szName);

    Netbios (&ncb);
    NBCheck (ncb);

    return (NRC_GOODRET == ncb.ncb_retcode);
}
```

Additional reference words: 3.10 3.50
KBCategory: kbprg kbnetwork
KBSubcategory: NtwkNetios

List of Articles for Win32 SDK Base Programming Issues

PSS ID Number: Q89989
Authored 17-Jan-1994 Last modified 25-May-1995

The information in this article applies to:

- FastTips for the Microsoft Win32 Software Development Kit (SDK), versions 3.1 and 3.5

INSTRUCTIONS

Microsoft FastTips is available 24 hours a day, 7 days a week, from a touch-tone telephone. To order items from this catalog, first select the items you want to receive, noting the five- to six-digit number of the Item ID shown below for each item, and then:

- Dial the toll-free FastTips number (800) 936-4300.
- When prompted, select the Win32 Software Development Kit.
- Press one (1) on your phone keypad to select Express Order Service.
- When prompted, select the delivery method, fax.
- When prompted, enter your three-digit area code and seven-digit fax number on your phone keypad.
- When prompted, enter the number of the Item ID and press #, for up to five items.

When finished, simply hang up. If you have problems receiving a fax, please call (206) 635-3105.

ARTICLE LISTING

Q 98892	PRB: Unexpected Result of SetFilePointer() with Devices		2
Q 98893	Limit on the Number of Bytes Written Asynchronously	2	
Q 98952	Setting File Permissions	1	
Q 99026	Possible Serial Baud Rates on Various Machines	1	
Q 99114	Using GMEM_DDESHARE in Win32 Programming	1	
Q 99115	Preventing the Console from Disappearing	1	
Q 99173	Types of File I/O Under Win32	2	
Q 99261	Performing a Clear Screen (CLS) in a Console Application	1	
Q 99456	Win32 Equivalents for C Run-Time Functions	6	
Q 99794	FILE_FLAG_WRITE_THROUGH and FILE_FLAG_NO_BUFFERING	1	
Q 99795	PRB: SetConsoleOutputCP() Not Functional		1
Q 100027	Direct Drive Access Under Win32	1	
Q 100291	Restriction on Named-Pipe Names	1	
Q 100328	Replacing the Shell (Program Manager)	2	
Q 100329	CPU Quota Limits Not Enforced	1	
Q 101186	Time Stamps Under the FAT File System	1	
Q 101190	Examining the dwOemId Value	1	
Q 101193	Interrupting Threads in Critical Sections	1	
Q 101378	Impersonation Provided by ImpersonateNamedPipeClient()	1	
Q 102098	Gaining Access to ACLs	1	
Q 102099	Administrator Access to Files	1	
Q 102100	Passing Security Information to SetFileSecurity()	1	
Q 102101	Extracting the SID from an ACE	1	
Q 102102	How to Add an Access-Allowed ACE to a File	5	
Q 102103	Computing the Size of a New ACL	1	
Q 102104	FILE_READ_EA and FILE_WRITE_EA Specific Types	1	
Q 102105	System GENERIC_MAPPING Structures	1	
Q 102128	Why LoadLibraryEx() Returns an HINSTANCE	1	
Q 102352	Passing a Pointer to a Member Function to the Win32 API	2	
Q 102429	Detecting Closure of Command Window from a Console App	1	
Q 102447	Definition of a Protected Server	1	
Q 102555	How Windows NT Handles Floating-Point Calculations	2	
Q 102798	Security Attributes on Named Pipes	2	
Q 103193	Managing Heap Memory in Win32	1	
Q 103237	Using Temporary File Can Improve Application Performance	1	
Q 103858	Copy on Write Page Protection for Windows NT	2	
Q 103862	Symbolic Information for System DLLs	1	
Q 104122	Detecting Logoff from a Service	1	
Q 104136	Mapping .INI File Entries to the Registry	1	
Q 104378	PRB: Win32 SDK and VC++ NT Help Files Are Incompatible		1
Q 105299	Creating a Font for Use with the Console	1	
Q 105302	Cancelling WaitCommEvent() with SetCommMask()	1	
Q 105303	FIX: Redirecting Output to an MS-DOS-Based Application		1
Q 105304	SetErrorMode() Is Inherited	1	
Q 105305	Calling CRT Output Routines from a GUI Application	2	
Q 105306	Getting and Using a Handle to a Directory	1	

Q 115946 PRB: AccessCheck() Returns ERROR_INVALID_SECURITY_DESCR 1

Q 115947 Adding Categories for Events 1
Q 115948 Creating Access Control Lists for Directories 2
Q 117223 BUG: Byte-Range File Locking Deadlock Condition 2
Q 117261 PRB: RegCreateKeyEx() Gives Error 161 Under Windows NT 3.5 2
Q 117330 PRB: Error "Invalid DLL Entrypoint" when Loading a DLL 2

Q 117867 Detecting Parallel Out-of-Paper Status 1
Q 117872 Windows NT 3.5 Hives Not Compatible with Windows NT 3.1 1
Q 117892 Memory Requirements for a Win32 App vs. the Win16 Version 1
Q 118625 Detecting Data on the Communications Port 1
Q 118626 Determining Whether the User is an Administrator 2

Q 118816 PRB: LoadLibrary() Fails with _declspec(thread) 1
Q 119163 Getting the Filename Given a Window Handle 5
Q 119218 PRB: Named Pipe Write() Limited to 64K 1
Q 119219 PRB: GetVolumeInformation() Fails with UNC Name 1
Q 119220 ReadFile() at EOF Changed in Windows NT 3.5 1

Q 119669 Listing Account Privileges 2
Q 120556 PRB: Starting a Service Returns "Logon Failure" Error 1
Q 120557 Dealing w/ Lengthy Processing in Service Control Handler 1
Q 120697 Time Format for WIN32_FIND_DATA 1

End of listing.

Additional reference words: DSKBGuide 3.10 3.50 4.00 95
KBCategory: kbref kbtlc
KBSubcategory: BseMisc

List of Articles for Win32 SDK GDI Programming Issues

PSS ID Number: Q80828
Authored 13-Feb-1992 Last modified 05-May-1995

The information in this article applies to:

- FastTips for the Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

INSTRUCTIONS

Microsoft FastTips is available 24 hours a day, 7 days a week, from a touch-tone telephone. To order items from this catalog, first select the items you want to receive, noting the five- to six-digit number of the Item ID shown below for each item, and then:

- Dial the toll-free FastTips number (800) 936-4300.
- When prompted, select the Win32 Software Development Kit.
- Press one (1) on your phone keypad to select Express Order Service.
- When prompted, select the delivery method, fax.
- When prompted, enter your three-digit area code and seven-digit fax number on your phone keypad.
- When prompted, enter the number of the Item ID and press #, for up to five items.

When finished, simply hang up. If you have problems receiving a fax, please call (206) 635-3105.

ARTICLE LISTING

Q 105733 FIX: SetCaret API May Not Work Correctly in Win32-Based Apps 1
Q 106384 ClipCursor() Requires WINSTA_WRITEATTRIBUTES 1
Q 108234 DOCERR: LoadCursorFromFile() and SetSystemCursor() Are Missing 2
Q 108929 Querying Device Support for MaskBlt 1

Q 110702 DOCERR: DeviceCapabilitiesEx Not Implemented in Windows NT 3.1 1
Q 115762 Printing Offset, Page Size, and Scaling with Win32 1
Q 118622 Using the Document Properties Dialog Box 2
Q 119164 Use of Polygon() Versus PolyPolygon() 1
Q 122564 Prototypes for SetSystemCursor() & LoadCursorFromFile() 1

End of listing.

Additional reference words: DSKBGuide 3.10
KBCategory: kbref kbtlc
KBSubcategory: GdiMisc

List of Articles for Win32 SDK Networking Issues

PSS ID Number: Q81246
Authored 17-Jan-1994 Last modified 05-Jan-1995

The information in this article applies to:

- FastTips for the Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

INSTRUCTIONS

Microsoft FastTips is available 24 hours a day, 7 days a week, from a touch-tone telephone. To order items from this catalog, first select the items you want to receive, noting the five- to six-digit number of the Item ID shown below for each item, and then:

- Dial the toll-free FastTips number (800) 936-4300.
- When prompted, select the Win32 Software Development Kit.
- Press one (1) on your phone keypad to select Express Order Service.
- When prompted, select the delivery method, fax.
- When prompted, enter your three-digit area code and seven-digit fax number on your phone keypad.
- When prompted, enter the number of the Item ID and press #, for up to five items.

When finished, simply hang up. If you have problems receiving a fax, please call (206) 635-3105.

ARTICLE LISTING

End of listing.

Additional reference words: DSKBGuide 3.10 3.50
KBCategory: kbref kbtlc
KBSubcategory: NtwkMisc

List of Articles for Win32 SDK Tools

PSS ID Number: Q89345
Authored 17-Jan-1994 Last modified 26-May-1995

The information in this article applies to:

- FastTips for the Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

INSTRUCTIONS

Microsoft FastTips is available 24 hours a day, 7 days a week, from a touch-tone telephone. To order items from this catalog, first select the items you want to receive, noting the five- to six-digit number of the Item ID shown below for each item, and then:

- Dial the toll-free FastTips number (800) 936-4300.
- When prompted, select the Win32 Software Development Kit.
- Press one (1) on your phone keypad to select Express Order Service.
- When prompted, select the delivery method, fax.
- When prompted, enter your three-digit area code and seven-digit fax number on your phone keypad.
- When prompted, enter the number of the Item ID and press #, for up to five items.

When finished, simply hang up. If you have problems receiving a fax, please call (206) 635-3105.

ARTICLE LISTING

Q 98918 PRB: Cannot Compile from Win32 SDK M Editor (MEP.EXE) 1
Q 99053 Source-level Debugging Under NTSD 1
Q 99516 Compile Errors Caused by Missing Option -D_X86_ 1
Q 99952 PRB: Debugging the Open Common Dialog Box in WinDbg 1

Q 99953 WinDbg Message "Breakpoint Not Instantiated" 1
Q 100289 Enabling Disk Performance Counters 1
Q 100292 PRB: Data Section Names Limited to Eight Characters 1
Q 100642 Setting Dynamic Breakpoints in WinDbg 2
Q 100957 PRB: Debugging an Application Driven by MS-TEST 1

Q 101187 Interpreting Executable Base Addresses 1
Q 102351 Debugging Console Apps Using Redirection 1
Q 102764 FIX: MIPS Compiler Assertion with C version 8.0 1
Q 103861 Choosing the Debugger That the System Will Spawn 1
Q 103863 Cannot Load <exe> Because NTVDM Is Already Running 1

Q 105583 Viewing Globals Out of Context in WinDbg 1
Q 105584 PRB: RC Does Not Support __DATE__ or __TIME__ 1
Q 105585 PRB: Unable to Freeze One Thread in WinDbg 1
Q 105586 FIX: WinDbg FIND Dialog Box Slows Down the System 1
Q 105677 Debugging the Win32 Subsystem 1

Q 105679 Differences Between the Win32 SDK and 32-Bit VC++ 2
Q 105764 Listing the Named Shared Objects 1
Q 106064 PRB: RW1016 Error Due to Unexpected End of File (EOF) 1
Q 106066 Additional Remote Debugging Requirement 1
Q 106382 PRB: Problems with the Microsoft Setup Toolkit 2

Q 108055 FIX: Internal Error from NetUserEnum() 1
Q 108227 Changes to the MSTest WFndWndC() 1
Q 114610 PRB: "Out of Memory Error" in the Win32 SDK Setup Sample 1
Q 115234 Win32 SDK 3.5 LINK32.EXE Calls CVTRES.EXE 1
Q 118890 Using the Call-Attributed Profiler (CAP) 2

End of listing.

Additional reference words: DSKBGuide 3.10 3.50
KBCategory: kbref kbtlc
KBSubcategory: TlsMisc

List of Articles for Win32 SDK User Programming Issues

PSS ID Number: Q89372
Authored 17-Jan-1994 Last modified 13-Jun-1995

The information in this article applies to:

- FastTips for the Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, 4.0

INSTRUCTIONS

Microsoft FastTips is available 24 hours a day, 7 days a week, from a touch-tone telephone. To order items from this catalog, first select the items you want to receive, noting the five- to six-digit number of the Item ID shown below for each item, and then:

- Dial the toll-free FastTips number (800) 936-4300.
- When prompted, select the Win32 Software Development Kit.
- Press one (1) on your phone keypad to select Express Order Service.
- When prompted, select the delivery method, fax.
- When prompted, enter your three-digit area code and seven-digit fax number on your phone keypad.
- When prompted, enter the number of the Item ID and press #, for up to five items.

When finished, simply hang up. If you have problems receiving a fax, please call (206) 635-3105.

ARTICLE LISTING

Q 97925 SetActiveWindow() and SetForegroundWindow() Clarification 1
Q 99359 UNICODE and _UNICODE Needed to Compile for Unicode   1
Q 99360 Memory Handles and Icons   1
Q 99392 Using SetThreadLocale() for Language Resources   1

Q 100486 PRB: AttachThreadInput() Resets Keyboard State   1
Q 100488 System Versus User Locale Identifiers   1
Q 102428 Debugging a System-Wide Hook   1
Q 102446 DOCERR: WM_ENTERIDLE Documentation Is Misleading   1
Q 102482 SetTimer() Should Not Be Used in Console Applications   1

Q 102485 The SBS_SIZEBOX Style   1
Q 102765 Clarification of the "Country" Setting   1
Q 103240 DOCERR: CloseClipboard() Suggests Calling DuplicateHandle()   1
Q 103644 Differences Between hInstance on Win 3.1 and Windows NT   2
Q 103977 Unicode Implementation in Windows NT 3.1   1

Q 104011 Propagating Environment Variables to the System   1
Q 104311 32-Bit Scroll Ranges   1
Q 104316 How Keyboard Data Gets Translated   1
Q 105300 COMCTL32 APIs Unsupported in the Win32 SDK   1
Q 105446 Win32 Shell Dynamic Data Exchange (DDE) Interface   2

Q 105530 Win32 Drag and Drop Server   1
Q 105575 FIX: Low Memory Condition Cause APIs to Return Random Values   1
Q 106065 Development Tools Do Not Accept Unicode Text   1
Q 106385 Identifying a Previous Instance of an Application   1
Q 106386 Retrieving DIBs from the Clipboard   1

Q 106716 Using SendMessageTimeout() in a Multithreaded Application 2
Q 106717 Journal Hooks and Compatibility   1
Q 108232 Hooking Console Applications and the Desktop   1
Q 108233 PRB: GetOpenFileName() and Spaces in Long Filenames   1
Q 108450 MultiByteToWideChar() Codepages CP_ACP/CP_OEMCP   1

Q 110704 Replacing Windows NT Control Panel's Mouse Applet   1
Q 115081 DOCERR: DragQueryFile() Return Code Can Be Misleading   1
Q 118624 Using GetForegroundWindow() When Desktop Is Not Active   1

End of listing.

Additional reference words: DSKBGuide 3.10 3.50 3.51 4.00 95
KBCategory: kbref kbtlc
KBSubcategory: UsrMisc

List of Articles for Win32s Programming Issues

PSS ID Number: Q93064
Authored 17-Jan-1994 Last modified 21-Feb-1995

The information in this article applies to:

- FastTips for Microsoft Win32s version 1.1

INSTRUCTIONS

Microsoft FastTips is available 24 hours a day, 7 days a week, from a touch-tone telephone. To order items from this catalog, first select the items you want to receive, noting the five- to six-digit number of the Item ID shown below for each item, and then:

- Dial the toll-free FastTips number (800) 936-4300.
- When prompted, select the Win32 Software Development Kit.
- Press one (1) on your phone keypad to select Express Order Service.
- When prompted, select the delivery method, fax.
- When prompted, enter your three-digit area code and seven-digit fax number on your phone keypad.
- When prompted, enter the number of the Item ID and press #, for up to five items.

When finished, simply hang up. If you have problems receiving a fax, please call (206) 635-3105.

ARTICLE LISTING

ITEM ID	ARTICLE TITLE	PAGES
Q 83520	General Overview of Win32s	2
Q 93639	PRB: Win32s GetVolumeInformation() Returns 0x12345678	1
Q 97785	Calling a Win32 DLL from a Win16 App on Win32s	2
Q 97918	Win32s Message Queue Checking	1
Q 98286	PRB: _getdcwd() Returns the Root Directory Under Win32s	2
Q 98723	FIX: GetVersion() Returns Invalid Value Under Win32s	1
Q 100713	Support for Sleep() on Win32s	1
Q 100833	Win32s Translated Pointers Guaranteed for 32K	1
Q 102430	Debugging Applications Under Win32s	2
Q 102762	GetCommandLine() Under Win32s	1
Q 104314	Win32s NetBIOS Programming Considerations	1
Q 105170	SAMPLE: Win32s Universal Thunks	1
Q 105680	Win32s Cannot Support _environ in DLLs	1
Q 105756	Debugging Universal Thunks	1
Q 105757	Using Windows Sockets Under Win32s and WOW	1
Q 105758	Win32s and Windows NT Timer Differences	1
Q 105759	Using Serial Communications Under Win32s	1

Q 105760 Using VxDs and Software Interrupts Under Win32s 1
Q 105761 Getting Resources from 16-Bit DLLs Under Win32s 1
Q 105762 Sharing Memory Between 32-Bit and 16-Bit Code on Win32s 1

Q 106715 Troubleshooting Win32s Installation Problems 3
Q 108403 FIX: Opening Named Pipes on Win32s 2
Q 108497 DIB_PAL_INDICES and CBM_CREATEDIB Not Supported in Win32s 1
Q 108722 PRB: "Routine Not Found" Errors in Windows Help 1
Q 109620 Creating Instance Data in a Win32s DLL 1

Q 110705 BUG: Problems with Win32s CreateFileMapping() 2
Q 110844 Detecting the Presence of NetBIOS in Win32s 1
Q 110845 How Win32-Based Applications Are Loaded Under Windows 1
Q 113739 BUG: Win32s 1.1 Bug List 3
Q 114340 Win32s 1.1 Limitations 7

Q 115080 Converting a Linear Address to a Flat Offset on Win32s 1
Q 115082 PRB: Page Fault in WIN32S16.DLL Under Win32s 1
Q 115084 Win32s Device-Independent Bitmap Limit 1
Q 117153 PRB: Display Problems with Win32s and the S3 Driver 1
Q 117825 Handling COMMDLG File Open Network Button Under Win32s 1

Q 117864 PRB: GP Fault Caused by GROWSTUB in POINTER.DLL 1
Q 117893 PRB: Result of localtime() Differs on Win32s and Windows NT 1
Q 120486 How to Remove Win32s 1
Q 121091 How to Determine Which Version of Win32s Is Installed 1
Q 121092 PRB: Local Reboot (CTRL+ALT+DEL) Doesn't Work Under Win32s 1

Q 121093 Points to Remember When Writing a Debugger for Win32s 7
Q 121094 PRB: Controls Do Not Receive Expected Messages 2
Q 121095 PRB: GPF When Spawn Windows-Based App w/ WinExec() in Win32s 1
Q 122235 Microsoft Win32s Upgrade 2

End of listing.

Additional reference words: 1.10 dskbguide
KBCategory: kbref kbtlc
KBSubcategory: W32s

Listing Account Privileges

PSS ID Number: Q119669
Authored 20-Aug-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

When a user starts a process, that process takes on the security attributes of the user. The security attributes inherited from the user include privileges, which control access to system services.

MORE INFORMATION

To list the privileges belonging to a process (and thus to the current user), perform the following steps:

1. Call GetCurrentProcess() to obtain a handle to the current process.

2. Call GetProcessToken() to obtain the process' access token.

3. Call GetTokenInformation() to obtain the list of privileges (among other information).

4. Step through the list of privileges, using LookupPrivilegeName() and LookupPrivilegeDisplayName() to obtain the names for the program to display.

The following sample code lists the displayable privilege names and the privilege names as defined in the WINNT.H header file:

Sample Code

```
#include <windows.h>
#include <stdio.h>

void main()
{
   HANDLE hProcess, hAccessToken;
   UCHAR InfoBuffer[1000];
   PTOKEN_PRIVILEGES ptgPrivileges = (PTOKEN_PRIVILEGES)InfoBuffer;
   DWORD dwInfoBufferSize;
   DWORD dwPrivilegeNameSize;
   DWORD dwDisplayNameSize;
   UCHAR ucPrivilegeName[500];
   UCHAR ucDisplayName[500];
   DWORD dwLangId;
```

```
      UINT x;

      hProcess = GetCurrentProcess();

      OpenProcessToken( hProcess, TOKEN_READ, &hAccessToken );

      GetTokenInformation( hAccessToken, TokenPrivileges, InfoBuffer,
         sizeof(InfoBuffer), &dwInfoBufferSize);

      printf( "Account privileges: \n\n" );
      for( x=0; x<ptgPrivileges->PrivilegeCount; x++ )
      {
         dwPrivilegeNameSize = sizeof( ucPrivilegeName );
         dwDisplayNameSize = sizeof( ucDisplayName );
         LookupPrivilegeName( NULL, &ptgPrivileges->Privileges[x].Luid,
            ucPrivilegeName, &dwPrivilegeNameSize );
         LookupPrivilegeDisplayName( NULL, ucPrivilegeName,
            ucDisplayName, &dwDisplayNameSize, &dwLangId );
         printf( "%40s (%s)\n", ucDisplayName, ucPrivilegeName );
      }
   }
```

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

Listing the Named Shared Objects

PSS ID Number: Q105764
Authored 24-Oct-1993 Last modified 23-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

Included with the Win32 SDK is an object viewer utility called WinObj that be used to list named objects, devices, dynamic-link libraries (DLLs), and so forth. To find objects such as pipes, memory, and semaphores, start WinObj and select the folder BaseNamedObjects.

Additional reference words: 3.10 3.50
KBCategory: kbtool
KBSubcategory: TlsMisc

Localizing the Setup Toolkit for Foreign Markets

PSS ID Number: Q92523
Authored 09-Nov-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, version 3.5

SUMMARY

There are no localized versions of the Setup Toolkit .DLLs available. However, the resource files for these .DLLs are provided with the Windows 3.1 Software Development Kit (SDK). They can be found in the INTLDLL subdirectory in the MSSETUP directory created by the SDK Setup.

The strings in the STRINGTABLES in these .RC files can be translated. The localized .RC files can then be bound to the .DLLs using the resource compiler.

Additional reference words: 3.10 3.50 MSSETUP tool kit
KBCategory: kbtool
KBSubcategory: TlsMss

Location of the Cursor in a List Box

PSS ID Number: Q29961
Authored 09-May-1988 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

There is no way to determine which item in the list box has the cursor when the LBN_DBLCLK message is received. You must keep track of which item has the cursor as it moves among the items. When you receive the double-click message, you will know which box has the cursor.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Location of WNet* API Functions

PSS ID Number: Q102381
Authored 03-Aug-1993 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

The WNet* API routines are implemented in MPR.DLL. When linking an application that uses these routines, link with the import library MPR.LIB. For a list of which APIs can be resolved with which import libraries, see the file WIN32API.CSV, which is included in the Win32 SDK and the Visual C++ 32-bit edition.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubCategory: NtwkWinnet

Long Filenames on Windows NT FAT Partitions

PSS ID Number: Q115236
Authored 22-May-1994 Last modified 21-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, version 3.5

SUMMARY

Windows NT, version 3.5, introduces the ability to create or open files with long filenames (LFN) on Windows NT file allocation table (FAT) partitions.

UNICODE is stored on disk, so that the original name is always preserved for extended characters regardless of which code page happens to be active when reading from or writing to the disk.

The legal character set is that of the Windows NT file system (NTFS) (except for the ":" for opening NTFS alternative file streams), so you can copy arbitrary files between NTFS and FAT without losing any of the filename information.

MORE INFORMATION

The LFNs are not available from the MS-DOS DIR command, but they are available from the Windows NT DIR command. When you create an LFN on a Windows NT FAT partition, an accompanying short name is created just as on an NTFS partition. You can access the file or directory with either the long names or the short names under Windows NT.

For example, use the Microsoft Editor (MEP) to create a file named as follows on a FAT partition under Windows NT:

```
longfilename.fat
```

This is exactly how the filename appears when you run the DIR command from the Windows NT command prompt. However, when you boot the machine into MS-DOS and run the DIR command, the filename appears as follows:

```
longfi~1.fat
```

NOTE: NTFS partitions are not available under MS-DOS, so you cannot perform this experiment using an NTFS partition.

The same result can also be achieved programmatically. Build and run the following sample code on Windows NT:

Sample Code

```
#include <windows.h>

void main( )
{
   WIN32_FIND_DATA fd;
   char buf[80];

   FindFirstFile( "long*", &fd );
   wsprintf( buf, "File name is %s", fd.cFileName );
   MessageBox( NULL, buf, "Test", MB_OK );
   wsprintf( buf, "Alternate file name is %s", fd.cAlternateFileName );
   MessageBox( NULL, buf, "Test", MB_OK );
}
```

The first message box will read:

```
File name is longfilename.fat
```

The second message box will read:

```
Alternate file name is longfi~1.fat
```

Additional reference words: 3.50
KBCategory: kbprg
KBSubcategory: BseFileio

Looking Up the Current User and Domain

PSS ID Number: Q111544
Authored 14-Feb-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

Program Manager displays the logged in user account and domain name in its windows title. This information can be retrieved programmatically by extracting the user SID from the current access token and then looking up the account and domain name via the LookupAccountSid() Win32 API. Below is sample code demonstrating this technique:

Sample Code

```
void ShowUserDomain(void)
{
     HANDLE hProcess, hAccessToken;
       UCHAR InfoBuffer[1000],szAccountName[200], szDomainName[200];
       PTOKEN_USER pTokenUser = (PTOKEN_USER)InfoBuffer;
       DWORD dwInfoBufferSize,dwAccountSize = 200, dwDomainSize = 200;
       SID_NAME_USE snu;

       hProcess = GetCurrentProcess();

       OpenProcessToken(hProcess,TOKEN_READ,&hAccessToken);

       GetTokenInformation(hAccessToken,TokenUser,InfoBuffer,
            1000, &dwInfoBufferSize);

       LookupAccountSid(NULL, pTokenUser->User.Sid, szAccountName,
            &dwAccountSize,szDomainName, &dwDomainSize, &snu);

    printf("%s\\%s\n",szDomainName,szAccountName);
}
```

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

MAKEINTATOM() Does Not Return a Valid LPSTR

PSS ID Number: Q61980
Authored 17-May-1990 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The LPSTR returned from MAKEINTATOM() cannot be treated as a general-purpose string pointer. Instead, either use it with the Atom family or convert it into a valid string before passing it off as one.

MORE INFORMATION

The MAKEINTATOM() macro is documented on Page 370 of the "Microsoft Windows Software Development Kit Programmer's Reference" versions 2.x as returning a value of type LPSTR. This is correct, but misleading. The LPSTR value returned is a "fabricated" LPSTR and cannot be considered a general-purpose string pointer. Consider the definition of the MAKEINTATOM macro:

```
   #define MAKEINTATOM(i)  (LPSTR) ((DWORD)((WORD)(i)))
```

This tells the compiler to "take the integer value 'i', think of it as a WORD, zero-extend this into a DWORD, then think of this as a LPSTR." Thus, MAKEINTATOM(1Ah) returns 001Ah. This obviously is not the same as "1A", which would be ASCII(1)+ASCII(A)+0.

The reason this psuedo-LPSTR works with AddAtom(), for example, is that AddAtom() looks to see if the HIWORD of the LPSTR parameter is 0 (zero). If so, AddAtom() knows that the LOWORD contains an actual integer value and it simply grabs that.

The following code samples show how problems can occur with these psuedo-LPSTRs returned from MAKEINTATOM.

Incorrect

```
ATOM AddIntAtom(int iAtom)
{
    LPSTR   szAtom;

    MessageBox(hWnd,
               (szAtom=MAKEINTATOM(iAtom)),
               "Adding Atom",
               MB_OK);
    return (AddAtom(szAtom));
```

```
}
```

The above code fragment will create and return a valid atom, but the message box will display an erroneous value.

Correct

```
ATOM AddIntAtom(int iAtom)
{
    LPSTR   szAtom;
    char    szBuf[10];

    szAtom=MAKEINTATOM(iAtom);
    sprintf(szBuf, "%d", LOWORD(szAtom));   /* Here's the trick */
    MessageBox(hWnd,
               szBuf,
               "Adding Atom",
               MB_OK);
    return (AddAtom(szAtom));
}
```

In the above example, we converted the integer value contained in the LOWORD of szAtom into a character string, then used this new character string in the MessageBox() call.

Although these code fragments illustrate the limitations of a MAKEINTATOM LPSTR, they are not very realistic because you really should use GetAtomName() to get the character string of an atom. If you have not yet created an atom out of an integer value, you could just format the integer into character string directly, as follows:

```
   sprintf (szBuf, "%d", iAtom);
   MessageBox (hWnd, szBuf,....);
```

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDde

Making a List Box Item Unavailable for Selection

PSS ID Number: Q74792
Authored 30-Jul-1991 Last modified 16-May-1995

-
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

-

SUMMARY

In the Microsoft Windows graphical environment, an application can use a list box to enumerate options. However, there are circumstances in which one or more options may not be appropriate. The application can change the appearance of items in a list box and prevent the user from selecting one of these items by using the techniques discussed below.

MORE INFORMATION

Changing the Appearance of a List Box Item

To dim (gray) a particular item in a list box, use an owner-draw list box as follows:

1. Create a list box that has the LBS_OWNERDRAW and LBS_HASSTRINGS styles.

2. Use the following code to process the WM_MEASUREITEM message:

```
case WM_MEASUREITEM:
   ((MEASUREITEMSTRUCT FAR *)(lParam))->itemHeight = wItemHeight;
   break;
```

   wItemHeight is the height of a character in the list box font.

3. Use the following code to process the WM_DRAWITEM message:

```
#define PHDC (pDIS->hDC)
#define PRC  (pDIS->rcItem)

DRAWITEMSTRUCT FAR *pDIS;

   ...

case WM_DRAWITEM:
   pDIS = (DRAWITEMSTRUCT FAR *)lParam;

   /* Draw the focus rectangle for an empty list box or an
```

```
   empty combo box to indicate that the control has the
   focus
 */
if ((int)(pDIS->itemID) < 0)
{
   switch(pDIS->CtlType)
   {
      case ODT_LISTBOX:
         if ((pDIS->itemAction) & (ODA_FOCUS))
            DrawFocusRect (PHDC, &PRC);
         break;

      case ODT_COMBOBOX:
         if ((pDIS->itemState) & (ODS_FOCUS))
            DrawFocusRect (PHDC, &PRC);
         break;
   }
   return TRUE;
}

/* Get the string */
switch(pDIS->CtlType)
{
   case ODT_LISTBOX:
      SendMessage ( pDIS->hwndItem,
                    LB_GETTEXT,
                    pDIS->itemID,
                    (LPARAM)(LPSTR)szBuf);
      break;

   case ODT_COMBOBOX:
      SendMessage ( pDIS->hwndItem,
                    CB_GETLBTEXT,
                    pDIS->itemID,
                    (LPARAM)(LPSTR)szBuf);
      break;
}

if (*szBuf == '!')   // This string is disabled
{
   hbrGray = CreateSolidBrush (GetSysColor
                                  (COLOR_GRAYTEXT));
   GrayString ( PHDC,
                hbrGray,
                NULL,
                (LPARAM)(LPSTR)(szBuf + 1),
                0,
                PRC.left,
                PRC.top,
                0,
                0);
   DeleteObject (hbrGray);

   /* SPECIAL CASE - Need to draw the focus rectangle if
      there is no current selection in the list box, the
```

```
      1st item in the list box is disabled, and the 1st
      item has gained or lost the focus
    */
   if (pDIS->CtlType == ODT_LISTBOX)
   {
      if (SendMessage ( pDIS->hwndItem,
                        LB_GETCURSEL,
                        0,
                        0L) == LB_ERR)
         if ( (pDIS->itemID == 0) &&
              ((pDIS->itemAction) & (ODA_FOCUS)))
            DrawFocusRect (PHDC, &PRC);
   }
}

else  // This string is enabled
{
   if ((pDIS->itemState) & (ODS_SELECTED))
   {
      /* Set background and text colors for selected
         item */
      crBack = GetSysColor (COLOR_HIGHLIGHT);
      crText = GetSysColor (COLOR_HIGHLIGHTTEXT);
   }
   else
   {
      /* Set background and text colors for unselected
         item */
      crBack = GetSysColor (COLOR_WINDOW);
      crText = GetSysColor (COLOR_WINDOWTEXT);
   }

   // Fill item rectangle with background color
   hbrBack = CreateSolidBrush (crBack);
   FillRect (PHDC, &PRC, hbrBack);
   DeleteObject (hbrBack);

   // Set current background and text colors
   SetBkColor (PHDC, crBack);
   SetTextColor (PHDC, crText);

   // TextOut uses current background and text colors
   TextOut ( PHDC,
             PRC.left,
             PRC.top,
             szBuf,
             lstrlen(szBuf));

   /* If enabled item has the input focus, call
      DrawFocusRect to set or clear the focus
      rectangle */
   if ((pDIS->itemState) & (ODS_FOCUS))
      DrawFocusRect (PHDC, &PRC);
}
```

```
      return TRUE;
```

Strings that start with "!" are displayed dimmed. The exclamation mark character is not displayed.

Preventing Selection

To prevent a dimmed string from being selected, create the list box with the LBS_NOTIFY style. Then use the following code in the list box's parent window procedure to process the LBN_SELCHANGE notification:

```
  case WM_COMMAND:

    switch (wParam)
      {

       ...

      case IDD_LISTBOX:
        if (LBN_SELCHANGE == HIWORD(lParam))
          {
          idx = (int)SendDlgItemMessage(hDlg, wParam,
                        LB_GETCURSEL, 0, 0L);
          SendDlgItemMessage(hDlg, wParam, LB_GETTEXT, idx,
                        (LONG)(LPSTR)szBuf);
          if ('!' == *szBuf)
            {
            // Calculate an alternate index here
            // (not shown in this example).

            // Then set the index.
            SendDlgItemMessage(hDlg, wParam, LB_SETCURSEL, idx, 0L);
            }
          }
        break;

       ...

      }
   break;
```

When the user attempts to select a dimmed item, the alternate index calculation moves the selection to an available item.

Additional reference words: 3.00 3.10 3.50 4.00 95 listbox
KBCategory: kbprg
KBSubcategory: UsrCtl

Managing Per-Window Accelerator Tables

PSS ID Number: Q82171
Authored 29-Mar-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

In the Windows environment, an application can have several windows, each with its own accelerator table. This article describes a simple technique requiring very little code that an application can use to translate and dispatch accelerator key strokes to several windows. The technique employs two global variables, ghActiveWindow and ghActiveAccelTable, to track the currently active window and its accelerator table, respectively. These two variables, which are used in the TranslateAccelerator function in the application's main message loop, achieve the desired result.

MORE INFORMATION

The key to implementing this technique is to know which window is currently active and which accelerator table, if any, is associated with the active window. To track this information, process the WM_ACTIVATE message that Windows sends each time an application gains or loses activation. When a window loses activation, set the two global variables to NULL to indicate that the window and its accelerator table are no longer active. When a window that has an accelerator table gains activation, set the global variables appropriately to indicate that the accelerator table is active. The following code illustrates how to process the WM_ACTIVATE message:

```
case WM_ACTIVATE:
   if (wParam == 0)  // indicates loss of activation
   {
      ghActiveWindow = ghActiveAccelTable = NULL;
   }
   else              // indicates gain of activation
   {
      ghActiveWindow = <this window>;
      ghActiveAccelTable = <this window's accelerator table>;
   }
   break;
```

The application's main message loop resembles the following:

```
while (GetMessage(&msg,     // message structure
                  NULL,     // handle of window receiving the msg
```

```
                  NULL,     // lowest message to examine
                  NULL))    // highest message to examine
{
   if (!TranslateAccelerator(ghActiveWindow, // active window
                             ghActiveAccelTable, // active accelerator
                             &msg))
   {
      TranslateMessage(&msg); // Translates virtual key codes
      DispatchMessage(&msg);  // Dispatches message to
                              //   window procedure
   }
}
```

Under Windows version 3.1, the WM_ACTIVATE message with the wParam set to WA_INACTIVE indicates loss of activation.

Under Win32, the low-order word of wParam set to WA_INACTIVE indicates deactivation.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMen

Mapping .INI File Entries to the Registry

PSS ID Number: Q104136
Authored 08-Sep-1993 Last modified 20-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

Under Windows NT, .INI file variables are mapped into the Registry as defined in the

```
\HKEY_LOCAL_MACHINE
   \Software\Microsoft\WindowsNT\CurrentVersion\IniFileMapping
```

mapping key. The Win32 Profile application programming interface (API) functions look for a mapping by looking up the filename extension portion of the profile file. If a match is found, then the search continues under that node for the specified application name. If a match is found, then the search continues for the variable name. If the variable name is not found, the value of the (NULL) variable name is a string that points to a node in the Registry, whose value keys are the variable names. If a specific mapping is found for the variable name, then its value points to the Registry value that contains the variable value.

The Profile API calls go to the Windows server to look for an actual .INI file, and read and write its contents, only if no mapping for either the application name or filename is found. If there is a mapping for the filename but not the application name, and there is a (NULL) application name, the value of the (NULL) variable will be used as the location in the Registry of the variable, after appending the application name to it.

In the string that points to a Registry node, there are several prefixes that change the behavior of the .INI file mapping:

! - This character forces all writes to go both to the Registry and to the .INI file on disk.

- This character causes the Registry value to be set to the value in the Windows 3.1 .INI file when a new user logs in for the first time after setup.

@ - This character prevents any reads from going to the .INI file on disk if the requested data is not found in the Registry.

USR: - This prefix stands for HKEY_CURRENT_USER, and the text after the prefix is relative to that key.

SYS: - This prefix stands for HKEY_LOCAL_MACHINE\Software, and the text after the prefix is relative to that key.

Additional reference words: 3.10 3.50 inifilemapping
KBCategory: kbprg

KBSubcategory: BseMisc

Mapping Modes and Round-Off Errors

PSS ID Number: Q89215
Authored 14-Sep-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

Mapping modes, window extents/origins, and viewport extents/origins allow for very powerful coordinate manipulation, such as scaling or moving objects. However, you should be aware that there are cases when using mapping modes other than MM_TEXT results in improper painting due to round-off errors.

Round-off errors occur when one logical unit does not equal one device unit, and an application requests the graphics device interface (GDI) to perform an action that would result in a nonintegral number of pixels needing to be drawn, scrolled, blt'd, and so on.

Round-off errors can manifest themselves in many ways, including unpainted portions of a client area when scrolling, gaps between objects that shouldn't have gaps (or vice versa), objects that shrink or grow one pixel depending on where they are on the screen, objects of unexpected sizes, and so on.

MORE INFORMATION

To better understand round-off errors, consider the following code:

```
SetMapMode (hDC, MM_ANISOTROPIC);
SetWindowExt (hDC, 2, 2);
SetViewportExt (hDC, 3, 3);
PatBlt (hDC, 0, 0, 5, 2, BLACKNESS);
```

This code tells the GDI to treat two logical units (the coordinates used by most GDI functions), in both the vertical and horizontal direction, as being equal to three device units (pixels). It then asks the GDI to draw what amounts to a black rectangle five logical units wide by three logical units tall starting at the logical point (0,0).

The GDI would translate this request into a request to draw a rectangle 7.5 (5 * 3/2 = 7.5) pixels wide by 3 (2 * 3/2 = 3) pixels tall. However, display cards cannot draw half a pixel, so the GDI would either have to round the width up to 8 or truncate it to 7. If an application relied on one behavior or the other, improper painting could occur.

Note that using mapping modes, window origins/extents, and viewport origins/extents does not mean that an application will have round-off errors. The occurrence of round-off errors depends on what these features are used for, the structure of the application, and other factors. Many applications take advantage of mapping modes, window origins/extents, and viewport origins/extents without ever encountering adverse round-off errors.

If an application exhibits round-off errors, there are a number of ways to prevent them, some which are described below.

Method 1

Only use MM_TEXT mapping mode, where one logical unit always equals one device unit. However, the application must do all its own scaling and moving of objects. The benefit of this approach is that the application has strict control over how objects are scaled and moved; you can look at your code to see the algebra that leads to round-off errors, and counter these errors appropriately. The drawback to this approach is that it makes the code more complicated and harder to read than it might be if the SetMapMode, SetWindowOrg, SetWindowExt, SetViewportOrg, and SetViewportExt functions were used.

Method 2

Mix MM_TEXT mapping mode with the mapping mode required. Sometimes applications only have round-off problems with certain types of objects. For example, in a graphing program, the application might want to set a certain mapping mode to draw a bar graph; this mapping mode might cause the fonts that the application draws to be of the wrong size.

To work around problems like this, mix MM_TEXT mapping mode with your mapping mode of choice. You could use MM_TEXT when dealing with objects that need exact sizes or placement and the other mapping mode for other drawing.

The benefits and drawbacks of this method are almost the same as those for method 1. However, with method 2, applications can take advantage of mapping modes for some of the scaling and moving of objects.

Method 3

If window/viewport origins/extents are set at compile time, be sure to only do operations that would result in no round-off errors. For example, take the fraction WindowExt over ViewportExt, and reduce this fraction. Then only do operations that involve multiples of the reduced WindowExt values. For example, given the following

```
WindowExt   = ( 6, 27)
ViewportExt = (50, 39)
```

turn this into a fraction and reduce it. It yields:

```
    in x direction:   6/50 = (2 * 3) / (2 * 5 * 5)  = 3/25
    in y direction:  27/39 = (3 * 3 * 3) / (3 * 13) = 9/13
```

Therefore, anything done in the x direction could be done using a multiple of three logical units; anything done in the y direction could be done using a multiple of nine logical units. For example, if the application wanted to scroll the window horizontally, it could scroll 3, 6, 9, 12, and so on logical units without having to deal with rounding errors. By using these values, the application will never have round-off errors.

One benefit of this method is that an application can take advantage of window origins/extents and viewport origins/extents. A disadvantage is that the application is limited to a certain set of origins/extents (that is, those built into the application at compile time).

Method 4

Applications can perform method 3 on-the-fly. This allows the application to deal with arbitrary window origins/extents and viewport origins/extents. To determine the minimum number of logical units an application could use given arbitrary extent values, the following code might prove useful (the code shown is for determining the value to use in the horizontal direction):

```
int GetMinWinXFactor (HDC hDC)
{
   int nMinX, xWinExt, xViewExt, nGCD;

   xWinExt  = LOWORD (GetWindowExt (hDC));
   xViewExt = LOWORD (GetViewportExt (hDC));
   while ((nGCD = GreatestCommonDivisor (xWinExt, xViewExt)) != 1)
      {
      xWinExt  /= nGCD;
      xViewExt /= nGCD;
      }
   return xWinExt;
}

int GreatestCommonDivisor (int i, int j)
{
   while (i != j)
      if (i > j)
         i -= j;
      else
         j -= i;
   return i;
}
```

The return value from the GetMinWinXFactor function above can then be used just like in method 3 (that is, the application can do all output based on multiples of this value).

Final Notes

The discussion above did not take into account the window origin, which can contribute to round-off errors. How origins and extents affect the coordinates that GDI uses is summed up in "Programmer's Reference.

Developers using mapping modes are encouraged to study the equations presented in the programmer's reference. The GDI uses these equations when converting between logical and device units. When round-off errors occur in an application, it is always a good idea to run the numbers through these equations to try to determine the cause of the errors.

Additional reference words: 3.00 3.10 3.50 4.00 95 rounding
KBCategory: kbprg
KBSubcategory: GdiDc

Maximum Brush Size

PSS ID Number: Q12071
Authored 01-Dec-1987 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows, versions 3.0 and 3.1
- Microsoft Win32 SDK, version 4.0

The maximum size of a brush is determined by the OEM driver. In theory, any brush size can be created. However, the brushes are realized by the OEM driver. The only requirement the driver must satisfy is that it must be able to handle 8 x 8 brushes.

If the driver realizes variable sizes, passing any bitmap is acceptable. The driver decides what to do with brushes larger than 8 x 8.

Currently, sample drivers use the top left 8 x 8 piece of the pattern; however, if the OEM wants to use the whole pattern, that also is proper. The limit of the current device drivers is 8 x 8 because of a performance trade-off.

The amount of pattern realized is a display driver-dependent issue.

Additional reference words: 3.00 3.10 4.00 95
KBCategory: kbprg
KBSubcategory: GdiPnbr

Memory Handle Allocation

PSS ID Number: Q91194
Authored 29-Oct-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

This article discusses the limitations that exist when allocating memory handles.

The minimum block you can reserve with each call to VirtualAlloc() is 64K. It is a good idea to confirm this number by checking the allocation granularity returned by GetSystemInfo().

With HeapAlloc(), there is no limit to the number of handles that can be allocated. GlobalAlloc() and LocalAlloc() (combined) are limited to 65536 total handles for GMEM_MOVEABLE and LMEM_MOVEABLE memory per process. Note that this limitation does not apply to GMEM_FIXED or LMEM_FIXED memory.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseMm

Memory Handles and Icons

PSS ID Number: Q99360
Authored 26-May-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, version 3.1

SUMMARY

Memory handles are not globally sharable among processes. Handles for icons, cursors, windows, and so forth, are not global handles but are handles into an index into the server-side handle table (the handle is actually an index into the server-side's handle table). Thus, some objects can be shared between processes -- but probably shouldn't be, for concurrency reasons.

GDI-related objects, however, are stored in a client-side handle table, which is translated to a handle value in a server-side table on every client-server transition. Thus, there are some objects that can be shared (USER-related objects) and some that can't be shared (BASE/KERNEL and GDI).

MORE INFORMATION

There are three types of handles in the system:

- Handles to objects that the executive (object manager) knows about. These are assigned on a per-process basis but each access to these objects goes through the executive.

- Handles that are maintained by the Win32 subsystem server (USER objects, including icons and cursors) and are therefore sharable. Please note that the allowed behavior of shared USER objects is subject to change in future releases of Windows NT. Thus, care should be taken when using these handles.

- Handles that are maintained by the Win32 subsystem client, and therefore are valid only in the context of the process that created it (GDI objects). These handles differ from the first type of handles listed in that you cannot call handle manipulation functions, such as DuplicateHandle() an WaitForSingleObject(), or use the security facilities on these objects.

Additional reference words: 3.10
KBCategory: kbprg
KBSubcategory: UsrMisc

Memory Requirements for a Win32 App vs. the Win16 Version

PSS ID Number: Q117892
Authored 13-Jul-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5 3.51, and 4.0
- Microsoft Win32s, versions 1.1, 1.15, and 1.2

SUMMARY

A Win32 port of a Windows-based application generally requires more virtual memory than the original Windows-based application. However, it is possible for the Win32 version of the application to have a smaller working set. The working set is the certain number of pages that the virtual memory manager must keep in memory for a process to execute efficiently. If you lower the working set of an application, it will use less RAM.

MORE INFORMATION

It can appear that the Win32-based version of an application running on Win32s requires more RAM than the Windows-based version of the application running on the same machine. This is because segments of a Windows-based application are loaded only as they are referenced, while the address space is reserved for the Win32-based application and its DLLs (dynamic-link libraries) at program load. Therefore, the memory count that is displayed by many "About" boxes is misleading: for the Windows-based application, the free memory reported is reduced by the number of segments actually loaded; for the Win32-based application, the free memory reported is reduced by the total address space required. However, this free memory represents only the virtual address space that all applications share, not the amount of RAM actually used.

You can use WMem to determine the address space used, the number of physical pages of RAM used, and to get an estimate of the working set. On a machine that has enough RAM to load the whole application without swapping, run only Program Manager and WMem. Use SHIFT+double-click in WMem and write down the available physical memory. Activate the application and use SHIFT+double-click again. The difference between the available physical memory before and after activating the application is a rough estimate of the working set. Test your application further and see how the working set changes during execution.

The working set of a Win32-based application can be decreased 30 percent or more with the use of the Working Set Tuner, included in the Win32 SDK. However, a Win32-based application may fail to load on Win32s even if its working set is significantly smaller than the free RAM (for example, 100K working set versus 1 megabyte free RAM). The entire application, DLLs included, must be mapped into the virtual address space.

The virtual memory size is set by the system at boot time, based on several factors. RAM is one factor, free disk space is another. The system must be able to allocate enough space for the swap file on disk. Windows, by default, allows the size of the swap file to be a maximum of 4 times larger than available RAM. This constant (4) can be modified by setting PageOverCommit in the 386enh section of the SYSTEM.INI file. Valid settings are between 1 and 20. Setting PageOverCommit to a value larger than 4 will result in less efficient usage of resources and slower execution, but it will allow you to run applications that otherwise are not able to run.

Additional reference words: 1.10 1.20 3.10 3.50 4.00 95 ProgMan
KBCategory: kbprg
KBSubcategory: BseMm

Menu Operations When MDI Child Maximized

PSS ID Number: Q71836
Authored 04-May-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

Pop-up menus added to an MDI application's menu using InsertMenu() with MF_BYPOSITION will be inserted one position further left than expected if the active MDI child window is maximized. This behavior occurs because the system menu of the active MDI child is inserted into the first position of the MDI frame window's menu bar.

To avoid this problem, if the active child is maximized when a new pop-up is inserted by position, add 1 (one) to the position value that would otherwise have been used. To determine that the currently active child window is maximized, send a WM_MDIGETACTIVE message to the MDI client window. In 16-bit Windows, if the high word of the return value from this message contains 1, the active child window is maximized. In Win32, you need to pass a pointer to a BOOL in the lParam. If the child window is maximized, then Windows will set the BOOL pointed to by the lParam to TRUE.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMdi

Message Retrieval in a DLL

PSS ID Number: Q96479
Authored 18-Mar-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

When a function in a dynamic-link library (DLL) retrieves messages on behalf of a calling application, the following must be addressed:

- The application and the DLL may be re-entered.
- The application can terminate while in the DLL's message retrieval loop.
- The DLL must allow the application to preprocess any messages that may be retrieved.

MORE INFORMATION

The following concerns arise when a function in a DLL retrieves messages by calling GetMessage or PeekMessage:

- When the DLL function retrieves, translates, and dispatches messages, the calling application and the DLL function may be re-entered. This is because message retrieval can cause the calling application to respond to user input while waiting for the DLL function to return. The DLL function can return a reentrancy error code if this happens. To prevent reentrancy, disable windows and menu-items, or use a filter in the GetMessage or PeekMessage call to retrieve specific messages.

- The application can terminate while execution is in the DLL function's message retrieval loop. The WM_QUIT message retrieved by the DLL must be re-posted and the DLL function must return to the calling application. This allows the calling application's message retrieval loop to retrieve WM_QUIT and terminate.

- When the DLL retrieves messages, it must allow the calling application to preprocess the messages (to call TranslateAccelerator, IsDialogMessage, and so forth) if required. This is be done by using CallMsgFilter to call any WH_MSGFILTER hook that the application may have installed.

The following code shows a message retrieval loop in a DLL function that waits for a PM_COMPLETE message to signal the end of processing:

```
while (notDone)
{
```

```
   GetMessage(&msg, NULL, 0, 0);

   // PM_COMPLETE is a WM_USER message that is posted when
   // the DLL function has completed.
   if (msg.message == PM_COMPLETE)
   {
      Clean up and set result variables;
      return COMPLETED_CODE;
   }
   else if (msg.message == WM_QUIT)  // If application has terminated...
   {
      // Repost WM_QUIT message and return so that calling
      // application's message retrieval loop can exit.
      PostQuitMessage(msg.wParam);
      return APP_QUIT_CODE;
   }

   // The calling application can install a WH_MSGFILTER hook and
   // preprocess messages when the nCode parameter of the hook
   // callback function is MSGF_MYHOOK. This allows the calling
   // application to call TranslateAccelerator, IsDialogMessage, etc.
   if (!CallMsgFilter(&msg, MSGF_MYHOOK))
   {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
   }

   :
   :
}
```

Define MSGF_HOOK to a value greater than or equal to MSGF_USER defined in WINDOWS.H to prevent collision with values used by Windows.

Preprocessing Messages in the Calling Application

The calling application can install a WH_MSGFILTER hook to preprocess messages retrieved by the DLL. It is not required for the calling application to install such a hook if it does not want to preprocess messages.

```
   lpfnMsgFilterProc = MakeProcInstance((FARPROC)MsgFilterHookFunc,
                                        ghInst);
   hookprocOld = SetWindowsHook(WH_MSGFILTER, lpfnMsgFilterProc);
   // Call the function in the DLL.
   DLLfunction();
   UnhookWindowsHook(WH_MSGFILTER, lpfnMsgFilterProc);
   FreeProcInstance(lpfnMsgFilterProc);
```

MsgFilterHookFunc is the hook callback function:

```
LRESULT CALLBACK MsgFilterHookFunc(int nCode, WPARAM wParam,
                                   LPARAM lParam)
{
```

```
   if (nCode < 0)
      return DefHookProc(nCode, wParam, lParam, &hookprocOld);

   // If CallMsgFilter is being called by the DLL.
   if (nCode == MSGF_MYHOOK)
   {
      Preprocess message (call TranslateAccelerator,
          IsDialogMessage etc.);
      return 0L if the DLL is to call TranslateMessage and
          DispatchMessage. Return 1L if TranslateMessage and
          DispatchMessage are not to be called.
   }
   else return 0L;
}
```

Additional reference words: 3.10 3.50 3.51 4.00 95 yield
KBCategory: kbprg
KBSubcategory: UsrMsg

Method for Sending Text to the Clipboard

PSS ID Number: Q35100
Authored 01-Sep-1988 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

Sending text to the Clipboard is usually a cumbersome process of allocating and locking global memory, copying the text to that memory, and sending the Clipboard the memory handle. This method involves many pointers and handles and makes the entire process difficult to use and understand.

Clipboard I/O is easily accomplished with an edit control. If a portion of text is highlighted, an application can send the edit control a WM_COPY or WM_CUT message to copy or cut the selected text to the Clipboard. In the same manner, text can be pasted from the Clipboard by sending a WM_PASTE message to an edit control.

The following example demonstrates how to use an edit control transparently within an application to simplify sending and retrieving text from the Clipboard. Note that this code will not be as fast as setting or getting the Clipboard data explicitly, but it is easier from a programming standpoint, especially if the text to be sent is already in an edit control. Note also that the presence of the edit window will occupy some additional memory.

MORE INFORMATION

For simplified Clipboard I/O, do the following:

1. Declare a global HWND, hEdit, which will be the handle to the edit control.

2. In WinMain, use CreateWindow() to create a child window edit control. Use the style WS_CHILD, and give the control dimensions large enough to hold the most text that may be sent to or received from the Clipboard. CreateWindow() returns the handle to the edit control that should be saved in hEdit.

3. When a Cut or Copy command is invoked, use SetWindowText() to place the desired string in the edit control, then use SendMessage() to select the text and copy or cut it to the Clipboard.

4. When a Paste command is invoked, use SetWindowText() to clear the edit control, then use SendMessage() to paste text from the Clipboard. Finally, use GetWindowText() to copy the text in the

edit control to a string buffer.

The actual coding for this procedure is as follows:

```
    .
    .
    .

    #define ID_ED    100
    HWND       hEdit;

    .
    .
    .
    /* In WinMain: hWnd is assumed to be the handle of the parent window,
*/
    /* hInstance is the instance handle of the parent.
*/
    /* The "EDIT" class name is required for this method to work. ID_ED
*/
    /* is an ID number for the control, used by Get/SetDlgItemText.
*/

    hEdit=CreateWindow("EDIT",
                       NULL,
                       WS_CHILD | BS_LEFTTEXT,
                       10, 15, 270, 10,
                       hWnd,
                       ID_ED,
                       hInstance,
                       NULL);


    .
    .
    .

    /* In the procedure receiving CUT, COPY, and PASTE commands:         */
    /* Note that the COPY and CUT cases perform the same actions, only   */
    /* the CUT case clears out the edit control.                         */

    /* Get the string length */
    short    nNumChars=strlen(szText);

    case CUT:
        /* First, set the text of the edit control to the desired string */
        SetWindowText(hEdit, szText);

        /* Send a message to the edit control to select the string */
        SendMessage(hEdit, EM_SETSEL, 0, MAKELONG(0, nNumChars));

        /* Cut the selected text to the clipboard */
        SendMessage(hEdit, WM_CUT, 0, 0L);
        break;
```

```
    case COPY:
        /* First, set the text of the edit control to the desired string */
        SetWindowText(hEdit, szText);

        /* Send a message to the edit control to select the string */
        SendMessage(hEdit, EM_SETSEL, 0, MAKELONG(0, nNumChars));

        /* Copy the text to the clipboard */
        SendMessage(hEdit, WM_COPY, 0, 0L);
        break;


    case IDM_PASTE:
        /* Check if there is text available */
        if (IsClipboardFormatAvailable(CF_TEXT))
            {
            /* Clear the edit control */
            SetWindowText(hEdit, "\0");

            /* Paste the text in the clipboard to the edit control */
            SendMessage(hEdit, WM_PASTE, 0, 0L);

            /* Get the test from the edit control into a string.    */
            /* nNumChars represents the number of characters to get */
            /* from the edit control.                               */
            GetWindowText(hEdit, szText, nNumChars);
            }
        else
            MessageBeep(0); /* Beep on illegal request */
        break;
```

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrClp

Microsoft Win32s Upgrade

PSS ID Number: Q122235
Authored 01-Nov-1994 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32s, version 1.25a

SUMMARY

Microsoft Win32s version 1.25a, the latest version of the software that allows you to run Win32-based applications on Windows version 3.1 or Windows for Workgroups versions 3.1 and later, is now available as a Microsoft Application Note.

The Application Note number is PW1118, and the title is "Microsoft Win32s Upgrade." It includes these files:

- W32S125.EXE, which is version 1.25a of Win32s.

- LICENSE.TXT, which has legal information regarding redistribution of Win32s files [please consult the Win32s "Programmer's Reference," which is part of the Microsoft Win32 Software Development Kit (SDK), for further information].

- README.TXT, which is the text of the Application Note.

The "Microsoft Win32s Upgrade" does not include OLE support. For OLE support for Win32-based applications under Win32s, obtain the OLE 2.02 appnote, number WW1116. For more information, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID: Q123087
TITLE : WW1116: OLE Version 2.02

The Win32s upgrade is intended as an end-user upgrade and is not intended for further redistribution.

MORE INFORMATION

To obtain this Application Note (number PW1118) and the files included with it, download PW1118.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for PW1118.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL

Download PW1118.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get PW1118.EXE

If you are unable to access the sources listed above, you can have this Application Note mailed to you by calling Microsoft Product Support Services Monday through Friday, 6:00 A.M. to 6:00 P.M. Pacific time. If you are outside the United States, contact the Microsoft subsidiary for your area. To locate your subsidiary, please call Microsoft International Customer Service at (206) 936-8661.

Additional reference words: 1.25a
KBCategory: kbsetup kbappnote kbfile
KBSubcategory: W32s

MIDL 1.0 and MIDL 2.0 Full Pointers Do Not Interoperate

PSS ID Number: Q115830
Authored 05-Jun-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, version 3.5

Microsoft Remote Procedure Call (RPC), version 1.0, has minimal support for full pointers, so the version of the MIDL compiler with Microsoft RPC version 1.0 (MIDL 1.0) treats full pointers (specified with the ptr attribute) as unique pointers (specified with the unique attribute).

The MIDL compiler with Microsoft RPC, version 2.0 (MIDL 2.0), supports full pointers. Because of the way Microsoft RPC, version 1.0, handles the on-wire representation of pointers, applications compiled using MIDL 2.0 full pointers cannot operate interactively with applications compiled using MIDL 1.0 full pointers.

The workaround is to recompile the MIDL 2.0 application to use unique pointers.

Additional reference words: 3.50
KBCategory: kbprg
KBSubcategory: NtwkRpc

Mirroring Main Menu with TrackPopupMenu()

PSS ID Number: Q99806
Authored 08-Jun-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

A developer may want to use TrackPopupMenu() to display the same menu that is used by the main window. TrackPopupMenu() takes a pop-up menu, while GetMenu() and LoadMenu() both return handles to top level menus, and therefore you cannot use the menus returned from these functions with TrackPopupMenu(). To "mirror" the main menu, you must create pop-up menus with the same strings, style, and IDs as the main menu. To do this, use the following Windows APIs:

GetMenu()
CreatePopupMenu()
GetMenuState()
GetMenuString()
GetSubMenu()
AppendMenu()

MORE INFORMATION

The following code displays the same menu as the main window when the right mouse button is clicked:

```
// In the main window procedure...

    case WM_RBUTTONDOWN:
    {

            HMENU hMenu;            // The handle to the main menu.
            int nMenu;              // The index of the menu item.
            POINT pt;              // The point to display the track menu.
            HMENU hMenuOurs;        // The pop-up menu that we are creating.
            UINT nID;              // The ID of the menu.
            UINT uMenuState;        // The menu state.
            HMENU hSubMenu;        // A submenu.
            char szBuf[128];        // A buffer to store the menu string.

// Get the main menu.
            hMenu = GetMenu(hWnd);
            nMenu = 0;

// Create a pop-up menu.
            hMenuOurs = CreatePopupMenu();
```

```
// Get menu state will return the style of the menu
// in the lobyte of the unsigned int. Return value
// of -1 indicates the menu does not exist, and we
// have finished creating our pop up.
           while ((uMenuState =
              GetMenuState(hMenu,nMenu,MF_BYPOSITION)) != -1)
           {
                if (uMenuState != -1)
                {
// Get the menu string.
              GetMenuString(hMenu,nMenu, szBuf,128,MF_BYPOSITION);
                     if (LOBYTE(uMenuState) & MF_POPUP) // It's a pop-up
menu.
                     {
                         hSubMenu = GetSubMenu(hMenu,nMenu);
                            AppendMenu(hMenuOurs,
                     LOBYTE(uMenuState),hSubMenu,szBuf);
                     }
                     else  // Is a menu item, get the ID.
                     {
                         nID = GetMenuItemID(hMenu,nMenu);
                         AppendMenu(hMenuOurs,LOBYTE(uMenuState),nID,szBuf);

                     }
                     nMenu++;  // Get the next item.
                }
        }
        pt = MAKEPOINT(lParam);
// TrackPopupMenu expects screen coordinates.
        ClientToScreen(hWnd,&pt);
           TrackPopupMenu(hMenuOurs,
          TPM_LEFTALIGN|TPM_RIGHTBUTTON,
          pt.x,pt.y,0,hWnd,NULL);

// Because we are using parts of the main menu in our
// pop-up menu, we can't just delete the pop-up menu, because
// that would also delete the main menu. So we must
// go through the pop-up menu and remove all the items.
        while (RemoveMenu(hMenuOurs,0,MF_BYPOSITION))
            ;

// Destroy the pop-up menu.
        DestroyMenu(hMenuOurs);
    }
    break;
```

If the menu is never dynamically modified, then the menu hMenuOurs could be made static and created inside the WM_CREATE message, and destroyed in the WM_DESTROY message.

To see how this function works, paste this code into the MENU sample application shipped with both Microsoft Visual C/C++ and Microsoft C/C++ version 7.0 in the file MENU.C in the MenuWndProc() function.

Additional reference words: 3.10 3.50 3.51 4.00 95 popup
KBCategory: kbprg
KBSubcategory: UsrMen

Missing Japanese Win32s-J Version 1.25 Files

PSS ID Number: Q130844
Authored 30-May-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Japanese Win32s-J version 1.25

SUMMARY

The files missing from the Japanese Win32s version 1.25 release included in MSDN (Microsoft Developers Network) Development Platform April '95 (CD 1) are now available.

Download WIN32SJ.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for WIN32SJ.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download WIN32SJ.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the SOFTLIB\MSLFILES directory
  Get WIN32SJ.EXE

MORE INFORMATION

The following files are missing from the MSDN Win32s-J v1.25 NODEBUG directory:

ole2conv.dll
ole2conv.sym
ole2disp.dll
ole2disp.sym
ole2nls.dll
ole2nls.sym
ole2prox.dll
ole2prox.sym
ole2thk.dll
ole32.dll
oleaut32.dll
oleaut32.sym
olecli.dll
olecli32.dll
olecli32.sym

olesvr32.dll
olesvr32.sym
sck16thk.dll
shell32.dll
shell32.sym
stdole.tlb
stdole32.tlb
storage.dll
typelib.dll
typelib.sym
version.dll
version.sym
w32s.386
w32scomb.dll
w32scomb.sym
w32skrnl.dll
w32sys.dll
win32s.exe
win32s16.dll
winmm.dll
winmm16.dll
winspool.drv
winspool.sym
wsock32.dll
wsock32.sym

All of these files are included in the self-extracting file (WIN32SJ.EXE) available from the MSL.

Additional reference words: 1.25 3.10 kbinf kbmsdn
KBCategory: kbother kbfile
KBSubcategory: wintldev

Mixer Manager Incorporated into NT 3.5 SDK and DDK

PSS ID Number: Q124504
Authored 03-Jan-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.5, 3.51, and 4.0
- Microsoft Win32 Device Development Kit (DDK) version 3.5

In Windows version 3.1, the Mixer Manager Application Programming Interface (API) and Mixer Manager Service Provider Interface (SPI) are provided by the Windows Sound System Version 2.0 Driver Development Kit (DDK). The Windows Sound System Version 2.0 DDK supports 16-bit application and driver development for digital audio mixers.

Starting with Windows NT version 3.5, the Mixer Manager API has been incorporated into the Win32 SDK, and the Mixer Manager SPI has been incorporated into the Win32 DDK. As a result, there is no 32-bit version of the Windows Sound System Version 2.0 DDK; all of its functions have been incorporated into the Win32 SDK and DDK.

Additional reference words: 3.50 4.00 95 WSS DDK Mixer Manager
KBCategory: kbmm
KBSubcategory: MMMixer

MoveFileEx Not Supported in Windows 95 But Functionality Is

PSS ID Number: Q129532
Authored 27-Apr-1995 Last modified 07-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SUMMARY

Win32 applications running on Windows NT use MoveFileEx() with the DELAY_UNTIL_REBOOT flag to move, replace, or delete the files currently being used. Examples of such files include device drivers and setup programs. To replace a file that is in use, you would do something like this:

```
MoveFileEx(szDstFile, NULL, MOVEFILE_DELAY_UNTIL_REBOOT);
MoveFileEx(szSrcFile, szDstFile, MOVEFILE_DELAY_UNTIL_REBOOT);
```

Windows 95 does not implement MoveFileEx() but does provide an alternate way for all Win32-, Win16-, and MS-DOS-based applications to move, replace, or delete files that are currently in use. To replace a file that is in use, you would do something like this:

```
GetWindowsDirectory(szTmpName, cchpFullPathBuf);
lstrcat(szTmpName, "\\WININIT.INI");
WritePrivateProfileString("Rename", "NUL", szDstFile, szTmpName);
WritePrivateProfileString("Rename", szDstFile, szSrcFile, szTmpname);
```

MORE INFORMATION

Although Windows 95 does not implement MoveFileEx(), it provides similar functionality to the DELAY_UNTIL_REBOOT flag through the [rename] section of WININIT.INI. If WININIT.INI is present in the Windows directory, WININIT.EXE processes it when the system boots. Once WININIT.INI has been processed, WININIT.EXE renames it to WININIT.BAK.

The syntax of the [rename] section is:

```
DestinationFileName=SourceFileName
```

NOTE: The source and destination filenames in WININIT.INI must be 8.3 file names. Long filenames are not processed.

The [rename] section can have multiple lines with one file per line. To delete a file, specify NUL as the DestinationFileName.

NOTE: DestinationFileName and SourceFileName cannot be long filenames, because WININIT.INI is processed before the protected mode disk system is loaded, and long filenames are only available when the protected mode disk system is running.

Here are some example entries:

```
[rename]
NUL=C:\TEMP.TXT
C:\NEW_DIR\EXISTING.TXT=C:\EXISTING.TXT
C:\NEW_DIR\NEWNAME.TXT=C:\OLDNAME.TXT
C:\EXISTING.TXT=C:\TEMP\NEWFILE.TXT
```

The first line causes TEMP.TXT to be deleted. The second line causes EXISTING.TXT to be moved to a new directory. The third line causes OLDNAME.TXT to be moved and renamed. The fourth line causes an existing file to be overwritten by NEWFILE.TXT.

Applications should not use WritePrivateProfileString() to write entries to the [rename] section because there can be multiple lines with the same DestinationFileName, especially if DestinationFileName is "NUL." Instead, they should add entries by parsing WININIT.INI and appending them to the end of the [rename] section.

Applications that use WININIT.INI should check for its existence in the Windows directory. If WININIT.INI is present, then another application has written to it since the system was last restarted. Therefore, the application should open it, and add entries to the [rename] section. If WININIT.INI isn't present, the application should create it and add to the [rename] section. Doing so ensures that entries from other applications won't be deleted accidentally by your application.

NOTE: WININIT.INI is not processed until Windows 95 is restarted.

Additional reference words: 4.00
KBCategory: kbprg
KBSubcategory: BseFileio

MS Setup Disklay2 Utility Calls COMPRESS.EXE Internally

PSS ID Number: Q103071
Authored 16-Aug-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

The Disklay2 utility that comes with the Microsoft Setup Toolkit calls the COMPRESS.EXE application internally to compress files. If this utility cannot be found during the execution of Disklay2, the infamous "Bad Command or Filename" error results. The screen dump of the results of Disklay2's progress will contain this error message. The solution is to ensure COMPRESS.EXE is available in the path.

COMPRESS.EXE is a component of the Windows Software Development Kit (SDK).

Additional reference words: 3.10 3.50 4.00 95 mssetup tool kit
KBCategory: kbtool
KBSubcategory: TlsMss

MultiByteToWideChar() Codepages CP_ACP/CP_OEMCP

PSS ID Number: Q108450
Authored 12-Dec-1993 Last modified 24-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

MultiByteToWideChar() maps a character string to a wide-character string. The declaration of this application programming interface (API) is as follows:

```
int MultiByteToWideChar(uCodePage, dwFlags, lpMultiByteStr,
   cchMultiByte, lpWideCharStr, cchWideChar)

UINT uCodePage;         /* codepage                         */
DWORD dwFlags;          /* character-type options            */
LPCSTR lpMultiByteStr;  /* address of string to map          */
int cchMultiByte;       /* number of characters in string    */
LPWSTR lpWideCharStr;   /* address of wide-character buffer */
int cchWideChar;        /* size of wide-character buffer     */
```

The first parameter, uCodePage, specifies the codepage to be used when performing the conversion. This discussion applies to the first parameter of WideCharToMultiByte() as well. The codepage can be any valid codepage number. It is a good idea to check this number with IsValidCodepage(), even though MultiByteToWideChar() returns an error if an invalid codepage is used. The codepage may also be one of the following values:

```
CP_ACP       ANSI codepage
CP_OEMCP     OEM (original equipment manufacturer) codepage
```

CP_ACP instructs the API to use the currently set default Windows ANSI codepage. CP_OEMCP instructs the API to use the currently set default OEM codepage.

If Win32 ANSI APIs are used to get filenames from a Windows NT system, use CP_ACP when converting the string. Windows NT retrieves the name from the physical device and translates the OEM name into Unicode. The Unicode name is translated into ANSI if an ANSI API is called, then it can be translated back into Unicode with MultiByteToWideChar().

If filenames are being retrieved from a file that is OEM encoded, use CP_OEMCP instead.

MORE INFORMATION

When an application calls an ANSI function, the FAT/HPFS file systems will

call AnsiToOem(); however, if an ANSI character does not exist in an OEM codepage, the filename will not be representable. In these cases, SetFileApisToOEM() should be called to prevent this problem by setting a group of the Win32 APIs to use the OEM codepage instead of the ANSI codepage.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: WIntlDev

Multicolumn List Boxes in Microsoft Windows

PSS ID Number: Q64504
Authored 06-Aug-1990 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

In the Microsoft Windows environment, a multicolumn list box is designed to contain homogeneous data. For example, all the data might be "first names." These first names could logically fall into the same column or be in multiple columns. This feature was added to Windows at version 3.0 to enable a list box to be shorter vertically by splitting the data into two or three columns.

MORE INFORMATION

To create a multicolumn list box, specify the LBS_MULTICOLUMN style when creating the list box. Then the application calls the SendMessage function to send an LB_SETCOLUMNWIDTH message to the list box to set the column width.

When an application sends an LB_SETCOLUMNWIDTH message to a multicolumn list box, Windows does not update the horizontal scroll bar until the a string is added to or deleted from the list box. An application can work around this situation by performing the following six steps when the column width changes:

1. Send the LB_SETCOLUMNWIDTH message to the list box.

2. Send a WM_SETREDRAW message to the list box to turn off redraw.

3. Add a string to the list box.

4. Delete the string from the list box.

5. Send a WM_SETREDRAW message to the list box to turn on redraw.

6. Call the InvalidateRect function to invalidate the list box.

In response, Windows paints the list box and updates the scroll bar.

Windows automatically manages the list box, including horizontal and vertical scrolling and distributing the entries into columns. The distribution is dependent on the dimensions of the list box. Windows fills Column 1 first, then Column 2, and so on. For example, if an application has a list box containing 13 ordered items and vertical space for 5 items, items 1-5 would be in the first column, items 5-10

in the second, and 11-13 in the last column, and item order would be maintained.

A multicolumn list box cannot have variable column widths.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Multiline Edit Control Does Not Show First Line

PSS ID Number: Q66668
Authored 01-Nov-1990 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.10 3.5, 3.51, and 4.0

When a multiline edit control is created that is less than one system character in height, the text in the edit control will not be displayed and subsequent attempts to enter text will cause the edit control to beep. This functionality is an invalid multiline edit control under Microsoft Windows versions 3.0 and later, even though this construct does work in Windows versions 2.x.

The multiline edit control also checks to see if the next line of text is displayable. If the next line of text is not displayable, it will beep to let you know that you have reached the limit of the edit control.

There is a similar situation with a control that overlaps another control in a dialog box. This construct is also considered invalid; thus, the second control will not be displayed.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Multiline Edit Control Limits in Windows NT

PSS ID Number: Q89712
Authored 28-Sep-1992 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)versions 3.1, 3.5, 3.51, and 4.0

The default maximum size for a multiline edit (MLE) control in both Windows and Windows NT is 30,000 characters. The EM_LIMITTEXT message allows an application to increase this value. Setting "cchmax" to 0 is a portable method of increasing this limit to the maximum in both Windows and Windows NT. When cchmax is set to 0, the maximum size for an MLE is 4GB-1 (4 gigabytes minus 1).

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Multiline Edit Control Wraps Text Different than DrawText

PSS ID Number: Q67722
Authored 12-Dec-1990 Last modified 23-Jun-1995

-
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

-

SUMMARY

Multiline edit controls will not wrap text in the same manner as the DrawText() function. This can be a problem when an application displays text that has been in an edit control because the text may wrap in a different location.

It is possible to obtain the text from the edit control and display it statically in a window with the same line breaks. To do this, the application must retrieve each line of text separately. This can be accomplished by sending the EM_GETLINE message to the control and displaying the retrieved text with the TextOut() function.

MORE INFORMATION

The following is a brief code fragment that demonstrates how to obtain the text of a multiline edit control line by line:

```
   ... /* other code */

   char  buf[80];         // Buffer for line storage
   HDC   hDC;             // Temporary display context
   HFONT hFont;           // Temporary font storage
   int   iNumEditLines;   // How much text
   TEXTMETRIC tm;         // Text metrics

   // Get number of lines in the edit control
   iNumEditLines = SendMessage(hEditCtl, EM_GETLINECOUNT, 0, 0L);

   hDC = GetDC(hWnd);

   // Get font currently selected into the control
   hFont = SendMessage(hEditCtl, WM_GETFONT, 0, 0L);

   // If it is not the system font, then select it into DC
   if (hFont)
      SelectObject(hDC, hFont);

   GetTextMetrics(hDC, &tm);
   iLine = 0;
```

```
   while (iNumEditLines--)
      {
      // First word of buffer contains max number of characters
      // to be copied
      buf[0] = 80;

      // Get the current line of text
      nCount = SendMessage(hEditCtl, EM_GETLINE, iLine, (LONG)buf);
      TextOut(hDC, x, y, buf, nCount);  // Output text to device
      y += tm.tmHeight;
      iLine++;
      }

   ReleaseDC(hWnd, hDC);
   ... /* other code */
```

The execution time of this code could be reduced by using the ExtTextOut() function instead of TextOut().

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Multimedia API Parameter Changes in the Win32 API

PSS ID Number: Q125864
Authored 07-Feb-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

This article discusses the following topics concerning multimedia APIs in the Win32 SDK:

- Some Windows version 3.1 APIs that require a device ID will also accept a properly cast device handle in the Win32 API if the caller is a 32-bit application.
- Some Windows version 3.1 APIs that require a device handle will also accept a properly cast device ID in the Win32 API if the caller is a 32-bit application.
- An explanation of the difference between using a device ID and using a device handle in conjunction with the above functions is given.
- A list of related APIs that are now obsolete but are provided for backwards compatibility with Windows version 3.1 is given.

MORE INFORMATION

Functions That Accept A Device Handle

Several MIDI and wave audio APIs have been revised in Win32 to provide greater flexibility to application developers. The following APIs require the calling application to provide a device ID for the target device under Windows 3.1, but under Win32 the APIs also accept a properly cast device handle if the caller is a 32-bit application:

```
midiInGetDevCaps
midiOutGetDevCaps
waveInGetDevCaps
waveOutGetDevCaps
```

Functions That Accept A Device ID

Conversely, the following APIs require the calling application to provide a device handle under Windows 3.1, but under Win32 the APIs also accept a properly cast device ID if the caller is a 32-bit application:

```
   midiOutGetVolume
   midiOutSetVolume
   waveOutGetVolume
   waveOutSetVolume
```

Device ID vs. Device Handle

A device ID has a one-to-one correspondence with the physical device it references and is determined by querying for the number of devices of a given type in the system and selecting the desired device. A device handle refers to a specific instance of a device, of which there may be more than one, and is obtained by opening a device. A device instance may be thought of as a logical copy of a physical device.

If a 32-bit application is querying a device's capabilities using one of the xxxGetDevCaps APIs listed above, the distinction between whether a device ID or device handle is used in the function call is unimportant because all instances of a device have the same capabilities. The result of one of these function calls will be the same whether or not a device ID or device handle was used.

However, if a 32-bit application uses one of the xxxVolume APIs listed above to get or set the output volume of a device, the distinction between using a device ID or device handle becomes important. If a device ID is used in a call to these xxxVolume APIs, then the result of the call and/or information returned applies to all instances of the device. If a device handle is used in a call to the xxxVolume APIs, then the result of the call and/or information returned by the call applies only to the instance of the device referenced by the device handle.

The revised versions of the above APIs are available to 32-bit applications only. For backwards compatibility reasons, 16-bit applications are subject to the API design of Windows 3.1.

Obsolete APIs

In addition to the above changes, the following related APIs are now obsolete, but are included in Win32 for backwards compatibility purposes:

```
   midiInGetID
   midiOutGetID
   waveInGetID
   waveOutGetID
```

For further information about all the above APIs and how to use device IDs and device handles, please consult the Win32 Multimedia Programmer's Reference.

Additional reference words: 4.00 95

KBCategory: kbmm
KBSubcategory: MmMisc

Multimedia Group System and API Design Guidelines

PSS ID Number: Q67692
Authored 11-Dec-1990 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The Microsoft Multimedia Systems Group is doing a large amount of system design and implementation. This article comments on the areas of system design.

MORE INFORMATION

Definitions

Term	Definition
Module	A module provides a set of functions and the interface to access those functions. The interface is called the API.
Client	A client uses a module. A client might be an application or a dynamic-link library (DLL).
Prefix	The initialization portion of the module that must be called before any of the other functions can be accessed.
Postfix	The closing portion of the module that must be called after the client has finished using the functions of the module.
Channel	A channel is created by calling a module's prefix function. It is used by the rest of the functions of the module (including the postfix function). Two notes about a channel: 1. A module can allocate resources to a channel. For example, a handle to a window is a channel, which has memory, a window procedure, and many other resources associated with it. 2. The channel uniquely identifies the user of the interface to the module. This allows the module to perform functions uniquely on each open channel. For

example, each file handle represents a different file that has been opened. Each file handle may have different attributes (read, write, read/write, binary, and so forth).

Separate Driver Interface from Module Interface

Separating the application from the hardware is one of the major tasks of system software. Layers within the operating system are separated from the hardware through the use of drivers. The system interface to the application should hide the mechanisms of the drivers as much as possible. This allows changing the mechanics of a driver in a later version of the system software. This also preserves the formality of the interface; applications are prevented from directly accessing the hardware.

Each application should also be separated from the drivers within the system. The driver API should not depend on where the driver is located, what it is named, or what form it takes (DLL, VxD, and so forth).

A driver should shield the application writer from its internal workings to enforce the principle of information hiding. If too much information is available, the application writer may choose to directly access the hardware, which jeopardizes the separation of functionality provided by the driver mechanism and system API.

Every Module Should Have an Initialization Function

Requiring the application to call a prefix function before using a system module and a postfix function after using a system module provides three major benefits, as follows:

1. Later versions of the driver can easily virtualize resources because the driver defies a channel for all communication. The driver can maintain separate resources for each communication channel, which is required for virtualization.

   Note that the complete virtualization does not need to be done in the first implementation of the module. As long as the interface requires the prefix and postfix calls, subsequent prefix calls can fail with an appropriate error message. Future versions of the system can properly handle multiple requests for the system resource.

2. The system module can allocate resources when the channel is allocated (when the prefix function is called), rather than when the system is initialized. Modules that are not called consume no resources. Resource allocation for all modules is postponed as long as possible.

3. The system can resolve conflicts between clients through the identification provided by the channels.

Interface Naming

During the process of designing a new module, use the interface naming conventions from an existing module with similar functionality. For example, an interface that deals with files would have a prefix interface containing the word "open," and a postfix interface containing the word "close." Extend naming conventions to similar functional areas. A stream interface would also have an "open" and a "close" function.

If the module has new functionality, then the functions can have unique names, such as the MIDI driver's midiOutStop function. If the module is similar to but not the same as another module, then use the function name to distinguish between modules. For example, the CreateWindow and CreateWindowEx functions in Windows create windows but CreateWindowEx also allows an application to specify other attributes.

The goal is to provide programmers familiar with existing modules a basis by which to quickly learn the new interface.

Prefix function names with the module name (abbreviated, if necessary). This allows the documentation to sort function names alphabetically, while keeping related functions together. More importantly, it allows easy identification of the module to which a function belongs.

Definition Naming

Prefix the names of constants and data structures with the module name (abbreviated, if necessary). For example, STRM_SEEK is a constant in the STRM group. This allows easy identification of the module to which a definition or data structure belongs.

As another example, MIDIOUTCAPS is the Device Capabilities structure for the MIDI output module. Using the naming convention developed here and symmetry (discussed in depth in the second part of this article), the MIDI Input module Device Capabilities structure should be called MIDIINCAPS.

Use additional prefixes as appropriate to identify the use of the definition. For example, MOERR_NODRIVER is a definition in the MIDI Output module to describe an error, that of no driver present.

Registering Drivers with Module

Most of the systems designed by the Multimedia group allow device drivers to be installed by the original equipment manufacturer (OEM) or even by the end user (given an appropriate setup program). There are two main ways for these drivers to communicate with the main module.

The first is to place an entry in the SYSTEM.INI file. When the parent module loads, it loads the child driver and initiates communication with the child.

The other method is for the child driver to call the parent to register itself as a client. This second method presumes that there is a suitable method available to load the child. Windows provides such a mechanism.

Requiring a driver to register itself with the handler module provides four benefits:

1. Drivers can be installed by adding them to the "modules to load" list. This is much easier than creating a line for the SYSTEM.INI file.

2. The handler module is more general because it does not assume the presence of certain drivers. This enhances system portability and reduces interdependencies between drivers and handlers. This advantage also applies to drivers loaded by a parent process.

3. A driver can pass information about itself, such as its name and entry points, to its parent during registration. This further separates the parent module from the driver. As long as the format of the interface data is fixed, independent changes may be made to both parent and driver.

4. Run-time installation of drivers is possible. The inherent nature of registration makes installing new drivers while the system is running much easier. This also simplifies implementing virtualization.

Symmetry of Function Names

- Every Open function should have a Close function and every Get function a Put function.

- Related functions in separate areas should work the same way. For example, if the MIDI output has an Open, the MIDI input should also have an Open. Additionally, the return values and parameters should be as similar as possible. This eases the programmer's task of learning the new APIs. This applies even if the current implementation doesn't use API symmetry. See "Designing for Implementation in Steps," below.

Symmetry in Naming Conventions

- Name defined constants and types for related areas should all be named using the same conventions. For example, LPMIDICALLBACK and LPWAVECALLBACK.

- If a naming convention already exists for a function type, adhere

to it. Example: use SEEK and TELL functions to move within a file system.

- If any part of an existing convention is used, little deviation from it is allowed. For example, a combination of SEEK and GET functions to move within a file system would not be the product of good design because it confuses an existing convention.

- If a convention does not already exist, create a new naming convention to avoid confusing things. Example: KNOCK and ANSWER.

Design for Implementation in Steps

Most implementations of any size must be done in incremental steps of functionality. More and more features are added to the modules until the entire design is completely implemented. For large or complex modules, this process may occur over several years. However, the original design must anticipate the complete, final functionality, not just the short-term goals. For example, even if allowing multiple users of a module will not be implemented in the first phase, this capability should be designed into the API. That way, the impact on users of the module will be minimal once implementation is complete.

Avoid placing arbitrary limits on functionality due to details of the current implementation. For example, even if only one user can have a resource allocated today, this may not always be true. Specifically, the Open function should return a handle to the resource that is then passed to functions that manipulate the resource. In the future, when multiple users of the resource is implemented, it will not be necessary to change other functions or applications.

In a message-based system, functions should return a "message not recognized" code for unexpected messages that is distinct from the "an error occurred" code. Then, when a future version of the driver contains extended functionality, an application can determine if the installed version of the driver supports the new features. If not, the application can take appropriate alternative action.

A project designed to be built in phases has well defined progress milestones. This makes it much easier to track progress while the module is under construction.

Building a module in phases also makes it easier to verify that the module is built correctly. Testing receives increments of functionality instead of the entire product toward the end of the development cycle.

Error Reporting

An function call can fail for many reasons. It is best if the call can return the specific cause of the error in addition to noting that the call failed. Functions that return a handle, structure, or other data cause particular problems because there is a limited set of values

that are always invalid.

Three approaches to error reporting are:

1. Ignore it (not recommended).

2. Provide a separate "what was that error?" call. This is more complicated than it sounds because, in a multitasking system, there can be multiple users of the module at the same "time." This makes determining what was the last error for a particular application difficult.

3. Return the handle or structure in a parameter and return the error code as the function return. This seems to be the best option, and is the approach used by OS/2.

Now that the error code is available, what should be done with it? To allow for internationalization and for additional error codes, the application should not associate the error code with a message. Instead, provide a function in each API that returns the text message for a specified error code. This function might be named GetTextErrorInformation, for example.

Client-Supplied Buffers

It is desirable for the client application to provide all buffers that it will access. If a system module allocates and maintains buffers, many implementation problems can arise when a buffer is made visible to the client application. Three advantages of client-supplied buffers are:

1. If the system software runs at a different privilege level or on a different CPU, or is otherwise separate from the client application, the system software can easily access the buffer. However, at the client's lower privilege level, or if the client and operating system are on different CPUs, it may be extremely difficult (if not impossible) to make a system-supplied buffer available to the client.
2. When the application supplies the buffers, the application has complete control over how much memory the system module uses.

3. The application is responsible for reporting an out-of-memory error. This removes an error condition from the system call.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbmm kbprg
KBSubcategory: MMMisc

Multiple Columns of Text in Windows Help Files

PSS ID Number: Q67895
Authored 28-Dec-1990 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

Microsoft Windows Help version 3.0 will support multiple columns of information in Help files. Unfortunately, the Table features of Word for Windows are not supported in Help version 3.0. In Windows Help version 3.1, however, Word for Windows tables are supported. Below is an outline of the supported techniques for creating multiple columns in Help files.

MORE INFORMATION

The best results can be achieved with Help version 3.0 if the text is organized such that one column has the bulk of the text to be presented and the other columns have relatively little text. The text might resemble the following:

```
   column1column1  column2  column3column3column3column3column3column3col
                            column3column3column3...
```

As the size of the Help window is decreased, the text in the third column will wrap to remain displayed for as long as possible. To achieve this effect, format the text as paragraphs with an "outdent," or negative indent, as shown in the following example:

```
   First line               Left margin and tab stop
   v                        v
   column1column1  column2  column3column3column3column3column3column3col
                            column3column3column3...
```

A typical format for paragraphs of this type is:

```
   Left Margin 2.5"
   First Line -2.5"
```

It is also necessary to define a tab stop at 2.5 inches.

Having two columns of text in the outdent is merely an example. You can define as many or as few columns in that space as necessary.

If two columns with similar amounts of text in each are required, you can use the side-by-side paragraph formatting of Word for the Macintosh or Word for MS-DOS. Only two paragraphs side-by-side are

supported by the Help compiler.

If neither of the tools mentioned above is available, or if more complicated tables are required, you can format the tables manually. Define appropriate tab stops and use them to align the columns of text. Format each physical line in the table as a separate paragraph. Select the entire table and format the paragraphs as "Keep Together." In the context of a Help system, this paragraph format disables word wrap.

The following is an example of a more complex table. In this example, (P*) is the paragraph mark at the end of a line:

```
   column1column1   column2column2  column3column3column3column3colu(P*)
   column1column1   column2coul     column3column3column3column3colu(P*)
   column1column1                   column3column3column3column3(P*)
   column1column1(P*)

   newcol1newcol1   newcol2newcol2  newcol3newcol3newcol3newcol3...(P*)
```

As the size of the Help window is reduced, the text will not wrap. Instead, you must use the horizontal scroll bar at the bottom of the window to view the remainder of the table.

Support for Word for Windows tables was added in Windows Help version 3.1.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsHlp

Multiple Desktops Under Windows NT

PSS ID Number: Q92505
Authored 08-Nov-1992 Last modified 23-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.51, 3.5, and 3.1

Windows NT does not support sharing the same desktop between two monitors on the same machine. An independent software vendor (ISV) that wants this capability must write a display device driver that behaves similar to a normal driver with a single monitor, but actually controls two monitors.

Support for multiple desktops is exposed and supported in Windows NT version 3.51.

Additional reference words: 3.10 3.50
KBCategory: kbother kbui
KBSubcategory: UsrMisc

Multiple References to the Same Resource

PSS ID Number: Q83808
Authored 20-Apr-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

Windows supports multiple references to a given resource. For example, suppose that an application has two top-level menus that each contain the same submenu. (An application can use the AppendMenu or SetMenu functions to add a submenu to another menu at run time.)

Normally, destroying a menu destroys all of its submenus. In the case above, however, when one menu is destroyed, the other menu has a lock on the common submenu. Therefore, the common submenu remains in memory and is not destroyed. The handle to the submenu remains valid until all references to the submenu are removed. The submenu either remains in memory or is discarded, while its handle remains valid.

MORE INFORMATION

Windows maintains a lock count for each resource, including menus. When the lock count falls to zero, Windows can free (destroy) the object. Each time an application loads a resource, its lock count is incremented. If a resource is loaded more than once, only one copy is created; subsequent loads only increment the lock count. Each call to free a resource decrements its lock count.

When the LoadResource function determines if a resource has already been loaded, it also determines if the resource has been discarded. If so, LoadResource loads the resource again. The resource is not necessarily present in memory at all times. However, if the lock count is not zero and the resource is discarded, Windows will automatically reload the resource. All resources are discardable and will be discarded if required to free memory.

Therefore, in the example above, the application's call to the DestroyMenu function calls FreeResource, which checks the lock count. This process is analogous to LoadMenu, which calls LoadResource.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrRsc

Multiprotocol Support for Windows Sockets

PSS ID Number: Q125704
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

The Windows Sockets (WinSock) API is based on the Berkeley Sockets progamming model, the standard interface for TCP/IP network programming on Windows. However, the WinSock implementations for Windows NT and Windows 95 include support for additional network transports. Here are the network transports supported by Microsoft WinSock implementations, listed by platform. For convenience, the headers and libraries required for application development are also listed.

Platform	Transport	Header*	Lib
Windows NT	TCP/IP	WINSOCK.H	WSOCK32.LIB
	IPX/SPX	WSIPX.H, WSNWLINK.H	""
	NetBEUI (via NetBIOS)	WSNETBS.H	""
	Appletalk	ATALKWSH.H	""
	ISO/TP4	WSHISOTP.H	""
Windows 95	TCP/IP	WINSOCK.H	""
	IPX/SPX	WSIPX.H, WSNWLINK.H	""
Windows for Workgroups version 3.11	TCP/IP	WINSOCK.H	WINSOCK.LIB

* WINSOCK.H is required for all platforms and transports, in addition to other header files.

MORE INFORMATION

The Windows Sockets API provides a uniform interface to multiple network transports and shields the programmer from most transport level ideosyncracies. However, WinSock does not eliminate the need to learn the basics of the transport protocol used. In particular, the programmer should be familiar with the following aspects of any transport protocol used with Windows Sockets:

1. Addressing.

   Each transport uses different address format. For example, the IP socket address structures looks like this:

   ```
   /*
   ```

```
 * Socket address, internet style.
 */
struct sockaddr_in
{
   short   sin_family;
   u_short sin_port;
   struct  in_addr sin_addr;
   char    sin_zero[8];
};

struct in_addr
{
   union
   {
      struct { u_char s_b1,s_b2,s_b3,s_b4; } S_un_b;
      struct { u_short s_w1,s_w2; } S_un_w;
      u_long S_addr;
   } S_un;
}
```

In contrast, the IPX address structure looks like this:

```
typedef struct sockaddr_ipx
{
   short sa_family;
   char  sa_netnum[4];
   char  sa_nodenum[6];
   unsigned short sa_socket;
} SOCKADDR_IPX;
```

2. Connection-Oriented vs. Connectionless Transport.

   In a connection-oriented transport protocol such as TCP or SPX, applications are required to establish a virtual circuit before data transfer can take place. The following sequences of WinSock functions are required at minimum to establish a virtual circuit:

Server	Client
socket	socket
bind	connect
listen	
accept	

   After the virtual circuit is established, the send() and recv() functions are used to transfer data.

   In a connectionless transport, a virtual circuit is not established. Both the client and server exchange data by binding a socket and calling sendto() or recvfrom().

3. Virtual circuit termination semantics.

4. Message Oriented vs. Stream-Oriented.

See the Win32 SDK online documentation for information on these topics.

5. Expedited data delivery.

   This is data that has been earmarked by the application as urgent data, and will be sent by the transport as quickly as possible. Not all transports support this feature. The Windows Sockets API provides the ability to request expedited data delivery via the MSG_OOB flag in the send() function.

6. Broadcasts.

   Here is an extract from the Win32 SDK online documentation:

   "Most connectionless transport protocols support broadcasts in the same fashion, in which any bound socket can send a broadcast if the SO_BROADCAST option is set, and broadcasts sent to the appropriate local endpoint are received without any additional work on the part of the application. NetBIOS transport protocols, however, handle broadcasts somewhat differently. In order to receive broadcasts, an application must bind to the NetBIOS broadcast address, which is an asterisk ("*") followed by 15 blank spaces (ASCII character 0x20). This means two things: A socket must be specially bound to receive broadcasts, and applications cannot depend on receiving broadcasts intended only for a specific application, since all NetBIOS broadcasts are delivered to this address. In other protocols such as UDP/IP and IPX, broadcasts are delivered to a socket only if the broadcast was sent to the same port to which the socket was bound."

For more information on any of the above topics, please see the Win32 SDK online documentation.

Additional reference words: 4.00
KBCategory: kbnetwork
KBSubcategory: NtwkWinsock

Mutex Wait Is FIFO But Can Be Interrupted

PSS ID Number: Q125657
Authored 01-Feb-1995 Last modified 03-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1 and 3.5

SUMMARY

A mutex is a synchronization object that is signalled when it is not owned by any thread. Only one thread at a time can own a mutex. Other threads requesting the mutex will have to wait until the mutex is signalled. This article discusses the order in which threads acquire the mutex.

MORE INFORMATION

Threads that are blocked on a mutex are handled in a first in, first out (FIFO) order. Therefore, the first to wait on the mutex will be the first to receive the mutex, regardless of thread priority.

It is important to remember that Windows NT can interrupt the wait and that this will change the order in which threads are queued. A kernel-mode asynchronous procedure call (APC) can interrupt a user-mode thread's execution at any time. Once the normal execution of the thread resumes, the thread will again wait on the mutex; however, the thread is placed at the end of the wait queue. For example, each time you enter the debugger (hit a breakpoint, execute OutputDebugString(), and so on), all application threads are suspended. Suspending a thread causes the thread to run a piece of code in kernel mode. When you continue from the debugger, the threads are resumed, causing them to resume their wait for the mutex, but possibly in a different order than before. In this case, it does not look like the mutex is acquired in FIFO order. Some threads may be unable to acquire the mutex when the application in run under the debugger.

NOTE: This implementation detail is subject to change. Windows 95 and other platforms that support the Win32 API may adopt different strategies.

Most programs do not usually need to make any assumption about this behavior. The only class of applications that should be sensitive to mutex claim and release throughput are realtime-class applications. If throughput is of importance to a program, critical sections should be used wherever possible.

Additional reference words: 3.10 3.50 windbg ntsd
KBCategory: kbprg
KBSubcategory: BseSync

Named Pipe Buffer Size

PSS ID Number: Q105531
Authored 20-Oct-1993 Last modified 03-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1, 3.5, and 3.51

The documentation for CreateNamedPipe() indicates that

> The input and output buffer sizes are advisory. The actual buffer size reserved for each end of the named pipe is either the system default, the system minimum or maximum, or the specified size rounded up to the next allocation boundary.

The buffer size specified should be a reasonable size so that your process will not run out of nonpaged pool, but it should also be large enough to accommodate typical requests.

Every time a named pipe is created, the system creates the inbound and/or outbound buffers using nonpaged pool, which is the physical memory used by the kernel. The number of pipe instances (as well as objects such as threads and processes) that you can create is limited by the available nonpaged pool. Each read or write request requires space in the buffer for the read or write data, plus additional space for the internal data structures.

Whenever a pipe write operation occurs, the system first tries to charge the memory against the pipe write quota. If the remaining pipe write quota is enough to fulfill the request, the write completes immediately.

If the remaining pipe write quota is too small to fulfill the request, the system will try to expand the buffers to accommodate the data using nonpaged pool reserved for the process. The write will block until the data is read from the pipe so that the additional buffer quota can be released. Therefore, if your specified buffer size is too small, the system will grow the buffer as needed, but the downside is that the operation will block. If the operation is overlapped, a system thread is blocked; otherwise, the application thread is blocked.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseIpc

Named Pipes Under WOW

PSS ID Number: Q94948
Authored 26-Jan-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

A 16-bit Windows-based application under Windows on Win32 (WOW) may use a named pipe that was created previously by a Win32-based application; however, REDIR.EXE must be included in the AUTOEXEC.BAT file for this to work properly.

Note that for local named pipes, networking doesn't need to be enabled.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: SubSys

Nesting Quotation Marks Inside Windows Help Macros

PSS ID Number: Q77748
Authored 24-Oct-1991 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

In Windows Help macros, strings may be delimited in two ways. The string can be opened and closed by double quotation marks or the string can be opened by a single opening quotation mark and closed by a single closing quotation mark.

Any quoted strings contained in a string delimited with double quotation marks must be enclosed in opening and closing single quotation marks.

The single opening quotation mark is different from the single closing quotation mark. The single opening quotation mark (`) is paired with the tilde (~) above the TAB key on extended keyboards; the single closing quotation mark (') is the same as the apostrophe. For example,

```
CreateButton("time_btn", "&Time", "ExecProgram("clock", 0)")
```

is illegal because the string "clock" uses double quotation marks within the double quotation marks used for the ExecProgram macro. The following example corrects the error by enclosing "clock" in single quotation marks:

```
CreateButton("time_btn", "&Time", "ExecProgram(`clock', 0)")
```

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsHlp

NetBIOS Name Table and NCBRESET

PSS ID Number: Q95944
Authored 02-Mar-1993 Last modified 23-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

The Windows NT NetBIOS implementation conforms to the IBM NetBIOS 3.0 specifications, with several enhancements discussed in this article.

MORE INFORMATION

Name Table

Under Windows NT, the name table is maintained on a per-process basis, which means that names added by one process are not visible by a different process. This also means that for two processes to establish a session, both processes must register two different NetBIOS names. However, sessions can be established by two threads in the same process using the same NetBIOS name.

NCBRESET

The IBM NetBIOS 3.0 specifications defines four basic NetBIOS environments under the NCBRESET command. Win32 follows the OS/2 Dynamic Link Routine (DLR) environment. This means that the first NCB issued by an application must be a NCBRESET, with the exception of NCBENUM. The Windows NT implementation differs from the IBM NetBIOS 3.0 specifications in the NCB_CALLNAME field.

In the "IBM Local Area Network Technical Reference," under the section on NetBIOS 3.0, the NCB_CALLNAME field is defined as the following:

> REQ_SESSIONS at NCB_CALLNAME+0 (1-byte field)
> The number of sessions requested by the application program.
> If zero, the default of 16 is used.
>
> REQ_COMMANDS at NCB_CALLNAME+1 (1-byte field)
> The number of commands requested by the application program.
> If zero, the default of 16 is used.
>
> REQ_NAMES at NCB_CALLNAME+2 (1-byte field)
> The number of names requested by the application program. This
> does not include a reservation for NAME_NUMBER_1.
> If zero, the default of 8 is used.

REQ_NAME_ONE at NCB_CALLNAME+3 (1-byte field)
A request to reserve NAME_NUMBER_1 for this application program.
If 0, NAME_NUMBER_1 is not requested.
If not 0, NAME_NUMBER_1 is desired to be reserved for this application.

Under the Windows NT implementation, the REQ_COMMANDS (NCB_CALLNAME+1) field is ignored. Instead, an application is bound by the amount of memory the process can allocate.

For more information on the differences between the Windows NT implementation and the IBM NetBIOS 3.0 specifications, see "The NetBIOS Function" in the "Win32 API Reference" Help file.

For more information on version 3.0 of NetBIOS, contact IBM and order the "IBM Local Area Network Technical Reference."

Additional reference words: 3.00 3.10 3.50 NCBRESET
KBCategory: kbprg
KBSubcategory: NtwkNetbios

Network DDE For 16-bit Windows-based Apps Under Windows NT

PSS ID Number: Q127861
Authored 19-Mar-1995 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 3.5

Network Dynamic Data Exchange (NetDDE) has limited support for 16-bit Windows application running under Windows NT. You can use DDE across the network, however, the NDde APIs are not supported.

The NDde APIs, such as NDdeShareAdd(), are used to create and manage the NetDDE shares, not for the actual communication. Therefore, for 16-bit applications to use NetDDE under Windows NT, you will need to use Generic Thunks to thunk to the 32-bit NDde APIs to create and trust the share. Once that is done, you can communicate using DDE or DDEML.

NOTE: You must be an administrator to add a DDE share.

Additional reference words: 3.50
KBCategory: kbprg
KBSubcategory: SubSys UsrNetDde

New Dialog Styles in Windows 95

PSS ID Number: Q125678
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

Windows 95 provides a few new dialog styles -- all listed in this article. All Windows version 3.1 dialog styles are still usable in Windows 95. However, DS_LOCALEDIT cannot be used in Win32-based applications because it does not apply. Although not documented, DS_ABSALIGN and DS_SETFONT exist in Windows version 3.1. They are documented in Windows 95.

MORE INFORMATION

Here is a list of the new dialog styles:

DS_3DLOOK	Gives the dialog box a nonbold font and draws three-dimensional borders around control windows in the dialog box.
DS_CENTER	Centers the dialog box in the working area -- the area not obscured by the tray.
DS_CENTERMOUSE	Centers the mouse cursor in the dialog box.
DS_CONTEXTHELP	Includes a question mark in the title bar of the dialog box. When the user clicks the question mark, the cursor changes to a question mark with a pointer. If the user then clicks a control in the dialog box, the control receives a WM_HELP message. The control should pass the message to the dialog procedure, which should call the WinHelp function using the HELP_WM_HELP command. The Help application displays a pop-up window that typically contains help for the control. Note that DS_CONTEXTHELP is just a placeholder. When the dialog box is created, the system checks for DS_CONTEXTHELP and, if it is there, adds WS_EX_CONTEXTHELP to the extended style of the dialog box.
DS_CONTROL	Creates a dialog box that works well as a child window of another dialog box, much like a page in a property sheet. This style allows the user to tab among the control windows of a child dialog box, use its accelerator keys, and so on.
DS_FIXEDSYS	Uses SYSTEM_FIXED_FONT instead of SYSTEM_FONT.

DS_NOFAILCREATE	Creates the dialog even if errors occur -- for example, if a child window cannot be created or if the system cannot create a special data segment for an edit control.
DS_SETFOREGROUND	Brings the dialog box to the foreground. Internally, Windows calls the SetForegroundWindow function for the dialog box.

Additional reference words: 4.00 DLGTEMPLATE kbinf
KBCategory: kbui kbprg
KBSubcategory: UsrDlgs

New Owner in Take-Ownership Operation

PSS ID Number: Q111541
Authored 14-Feb-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

When ownership of a file is taken, the user performing the operation becomes the new owner. The exception to this rule is when the user is a member of the "Administrators" group. In this situation, the ownership of the file is assigned to the Administrators group.

The reasoning for this behavior is that the administrators on a particular system work together. When one administrator takes ownership of a file, the others should also receive access.

MORE INFORMATION

When a take-ownership operation is performed, the system assigns the new owner SID based on the TOKEN_OWNER field of the user's access token.

When a user logs on to a Windows NT system, the logon process builds an access token to represent the user. Normally the TOKEN_OWNER field in the access token is set equal to TOKEN_USER (the user's SID). However, when the user is a member of the Administrators group, the system sets the TOKEN_OWNER field to the Administrators SID.

Although Windows NT does not provide a user interface for changing the TOKEN_OWNER field in the user's access token, it is possible to programatically change this value via the SetTokenInformation() Win32 API (application programming interface).

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

New User Heap Limits Under Windows 95

PSS ID Number: Q125676
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

In Windows version 3.1, window and menu data is maintained in two 16-bit heaps. This limits window and menu data to 64k each. Windows 95 uses 32-bit heaps for window and menu data, thus greatly expanding the limits placed on the number of items contained in these heaps.

MORE INFORMATION

Windows version 3.1 user and menu heaps are each limited to 64K of data. As a result, the number of window and menus in a system are each constrained to around 200. In Windows 95, the number of Windows and Menus that may exist in the system goes up to 32K each. This is possible because the Windows 95 user and menu heaps are each two megabytes in size.

The first 64K of the user heap looks exactly as it did in Windows version 3.1, except for the absense of WND structures. The Windows 95 WND structures populate the heap space above 64K, thus increasing the number of WND structures the heap can hold. This new arrangement also has the positive effect of freeing up space in the lower 64K space, making more space for class structures and other items that reside in the user heap. For the menu heap, there is nothing special about the low 64K, menus and their data may appear anywhere in the two-megabyte heap.

Additional reference words: 4.00
KBCategory: kbusage
KBSubcategory: UsrMisc

New Window Styles in Windows 95

PSS ID Number: Q125679
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

Windows 95 provides a number of new window styles that help make the user interface more attractive and intuitive. These new styles are listed in this article. Most of the new styles are extended styles, which are specified with the CreateWindowsEx() function. All Windows version 3.1 window styles are still usable in Windows 95.

MORE INFORMATION

Here are the new styles:

WS_EX_ABSPOSITION	Specifies that a window has an absolute position.
WS_EX_CLIENTEDGE	Specifies that a window has a 3D look -- that is a border with a sunken edge.
WS_EX_CONTEXTHELP	Includes a question mark in the title bar of the window. When the user clicks the question mark, the cursor changes to a question mark with a pointer. If the user then clicks a child window, the child receives a WM_HELP message.
WS_EX_CONTROLPARENT	Allows the user to navigate among the child windows of the window by using the TAB key.
WS_EX_LEFT	Gives window generic left-aligned properties. This is the default.
WS_EX_LEFTSCROLLBAR	Places vertical scroll bar to the left of the client area.
WS_EX_LTRREADING	Displays the window text using left-to-right reading order properties. This is the default.
WS_EX_MDICHILD	Creates an MDI child window.
WS_EX_OVERLAPPEDWINDOW	Combines the WS_EX_CLIENTEDGE and WS_EX_WINDOWEDGE styles.
WS_EX_PALETTEWINDOW	Combines the WS_EX_WINDOWEDGE, WS_EX_SMCAPTION, and WS_EX_TOPMOST styles.

WS_EX_RIGHT	Gives window generic right-aligned properties. This depends on the window class.
WS_EX_RIGHTSCROLLBAR	Places vertical scroll bar (if present) to the right of the client area. This is the default.
WS_EX_RTLREADING	Displays the window text using right-to-left reading order properties.
WS_EX_SMCAPTION	Creates a window that has a small title bar.
WS_EX_STATICEDGE	Creates a window with a three-dimensional border style intended to be used for items that do not accept user input.
WS_EX_TOOLWINDOW	Creates a tool window, which is a window intended to be used as a floating toolbar. A tool window has a title bar that is shorter than a normal title bar, and the window title is drawn using a smaller font. A tool window does not appear in the task bar or in the window that appears when the user presses ALT+TAB.
WS_EX_WINDOWEDGE	Specifies that a window has a border with a raised edge.

Additional reference words: 4.00 CreateWindowEx kbinf
KBCategory: kbui kbprg
KBSubcategory: UsrWndw

New Windows 95 Styles Make Attaching Bitmap to Button Easier

PSS ID Number: Q125673
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

In Windows 95, there are two new button styles (BS_BITMAP and BS_ICON). Using these styles makes attaching a bitmap or an icon to a button in Windows 95 easier than it was in Windows version 3.1.

MORE INFORMATION

In Windows version 3.1 you create the button by using the CreateWindow() function with the BS_OWNERDRAW style to attach a bitmap or an icon to a button. Each time the parent window receives the WM_DRAWITEM message the bitmap or the icon must be loaded and drawn on the button.

In Windows 95 you can create a button with style BS_BITMAP to display a bitmap instead of text on the button. After you create the button by using CreateWindow() function, assign the bitmap to the button by sending a WM_SETIMAGE message to the button with the wparam as IMAGE_BITMAP and lparam as a handle to the bitmap. Windows displays the specified bitmap on the button. The attached bitmap should not be deleted until the button uses it.

SAMPLE CODE

```
hwndButton = CreateWindow(
              "BUTTON",   // predefined class
              "OK",       // button text
              WS_VISIBLE| WS_CHILD | BS_DEFPUSHBUTTON |BS_BITMAP, //styles
              // Size and position values are given explicitly, because
              // the CW_USEDEFAULT constant gives zero values for buttons.
              5, // starting x position
              5, // starting y position
              30,// button width
              18, // button  height
              hWnd,// parent window
              NULL,// No menu
              (HINSTANCE) GetWindowLong(hWnd,
                          GWL_HINSTANCE), NULL); // pointer not needed

// load the bitmap, NOTE: delete it only when it's no longer being used.
hBitmap   = LoadBitmap(hInst,MAKEINTRESOURCE(IDB_BITMAP1));
// associate the bitmap with the button.
SendMessage(hwndButton,BM_SETIMAGE,(WPARAM)IMAGE_BITMAP,
```

```
            (LPARAM)(HANDLE)hBitmap);
```

In Windows 95, you can also create a button with style BS_ICON to display a icon instead of text on the button. After creating the button by using CreateWindow() function, assign the icon to the button by sending a WM_SETIMAGE message to the button with the wparam as IMAGE_ICON and the lparam as the handle to the icon. Windows displays the specified icon on the button.

NOTE: The system handles cleanup of icons or cursors loaded from resources. Therefore, the icon should not be deleted unless the icon was created at run time by using CreateIcon() function.

SAMPLE CODE

```
hwndButton = CreateWindow(
                  "BUTTON",   // predefined class
                  "OK",       // button text
                  WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON|BS_ICON,//styles
                // Size and position values are given explicitly, because
                // the CW_USEDEFAULT constant gives zero values for buttons.

                   5, // starting x position
                   5, // starting y position
                  30, // button width
                  18, // button height
                  hWnd,   // parent window
                  NULL,   // No menu
                  (HINSTANCE) GetWindowLong(hWnd,GWL_HINSTANCE),
                  NULL);      // pointer not needed

// load the icon, NOTE: you should delete it after assigning it.
hIcon = LoadIcon(hInst,MAKEINTRESOURCE(IDI_ICON1));
// associate the icon with the button.
SendMessage(hwndButton,BM_SETIMAGE,(WPARAM)IMAGE_ICON,
            (LPARAM)(HANDLE)hIcon);
```

If a bitmap or icon is not specified via the BM_SETIMAGE message and the BS_BITMAP or BS_ICON style is specified, a blank button with no text is displayed.

Additional reference words: 4.00
KBCategory: kbprg
KBSubcategory: UsrWnd

No Subsystem Developer Kit or Support for Such a Kit

PSS ID Number: Q89987
Authored 06-Oct-1992 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

There is not a kit available for developers to write subsystems, nor are there plans to provide such a kit. Furthermore, there is no documentation or support available to develop a subsystem for Windows NT.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: SubSys

Non-Addressable Range in Address Space

PSS ID Number: Q92764
Authored 15-Nov-1992 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

In Windows NT, each process has its own private address space. The process can use up to 2 gigabytes of virtual memory. This 2Gb is not necessarily contiguous. The system uses the other 2Gb.

The user-mode addresses extend from 0x00010000 to 0x7FFF0000. The following ranges are reserved as non-address space to ensure that the process does not walk on system-owned memory

   0x00000000 to 0x0000FFFF (first 64K of virtual space)

   0x7FFF0000 to 0x7FFFFFFF (last 64K of user virtual space)

These are effectively PAGE_NOACCESS ranges.

Additionally, Win32 DLLs will reserve other specific address ranges. For more information, see the file COFFBASE.TXT that comes with the DDK.

MORE INFORMATION

This range is not guaranteed to serve this purpose in the future. There could be good reasons in a future implementation to use these addresses. Code that is going to depend on this non-address range should verify its validity at run time with something like

```
BOOL IsFirst64kInvalid(void)
{
   BOOL bFirst64kInvalid = FALSE;

   try {
      *(char *)0x0000FFFF;
   }
   except (EXCEPTION_EXECUTE_HANDLER) {
      if (EXCEPTION_ACCESS_VIOLATION == GetExceptionCode())
         bFirst64kInvalid = TRUE;
   }

   return bFirst64kInvalid;
}
```

Additional reference words: 3.10 3.50

KBCategory: kbprg
KBSubcategory: BseMm

Noncontinuable Exceptions

PSS ID Number: Q98840
Authored 13-May-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

An exception is noncontinuable if the event isn't continuable in the hardware, or if continuation makes no sense. For example, if the caller's stack is corrupted while trying to post an exception, continuing from the bad stack exception would not be useful.

The noncontinuable exception does not terminate the application, and therefore an application that can succeed in catching the exception and running after a noncontinuable exception is free to do so. However, a noncontinuable exception typically arises as a result of a corrupted stack or other serious problem, making it very difficult to recover from the exception.

Additional reference words: 3.10 3.50 4.00 95 non-continuable
KBCategory: kbprg
KBSubcategory: BseExcept

Nonzero Return from SendMsg() with HWND_BROADCAST

PSS ID Number: Q102588
Authored 04-Aug-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

The SendMessage() function calls the window procedure for the specified window and does not return until that window has processed the message and returned a value. Applications can send messages to all top-level windows in the system by specifying HWND_BROADCAST as the first parameter to the SendMessage() function. In doing so, however, applications lose access to the return values resulting from the SendMessage() call to each of the top-level windows.

MORE INFORMATION

When a call to SendMessage() is made, the value returned by the window procedure that processed the message is the same value returned from the SendMessage() call.

Among other things, SendMessage() determines whether the first parameter is HWND_BROADCAST (defined as -1 in WINDOWS.H). If HWND_BROADCAST is the first parameter, SendMessage enumerates all top-level windows in the system and sends the message to all these windows. Because this one call to SendMessage() internally translates to a number of SendMessage() calls to the top-level windows, and because SendMessage() can return only one value, Windows ignores the individual return values from each of the top-level window procedures, and just returns a nonzero value to the application that broadcast the message. Thus, applications that want to broadcast a message to all top-level windows, and at the same time expect a return value from each SendMessage() call, should not specify HWND_BROADCAST as the first parameter.

There are a couple of ways to access the correct return value from messages sent to more than one window at a time:

- If the broadcasted message is a user-defined message, and only a few other applications respond to this message, then those applications that trap the broadcasted message must return the result by sending back another message to the application that broadcast the message. The return value can be encoded into the message's lParam.

- If the application does not have control over which application(s) will respond to the message, and it still expects a return value, then the application must enumerate all the windows in the system

using EnumWindows() function, and send the message separately to each window it obtained in the enumeration callback function.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMsg

Objects Inherited Through a CreateProcess Call

PSS ID Number: Q83298
Authored 08-Apr-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

The objects inherited by a process started by a call to CreateProcess() are those objects that you can get a handle to and on which you can use the CloseHandle() function. These objects include the following:

Processes
Events
Semaphores
Mutexes
Files (including file mappings)
Standard input, output, or error devices

However, the new process will only inherit objects that were marked inheritable by the old process.

These are duplicate handles. Each process maintains memory for its own handle table. If one of the processes modifies its handle (for example, closes it or changes the mode for the console handle), other processes will not be affected.

Processes will also inherit environment variables, the current directory, and priority class.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseProcThrd

Obsolete Macro functions in Japanese Windows Version 3.1

PSS ID Number: Q130059
Authored 10-May-1995 Last modified 16-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 Software Development Kit (SDK) version 3.5
- Microsoft Win32s version 1.2

SUMMARY

Some of the Japanese Windows version 3.1 functions are no longer supported in the Japanese version of Windows NT. These Macro functions are obsolete in the current C++ compiler.

MORE INFORMATION

The macro functions of Japanese Windows version 3.10 are no longer supported and are obsolete in current C++ compiler. They were remapped to other generic MBCS functions. Use the following as workaround functions. Include them in the current header file. T _ismbb* entries correspond to the replaced Windows version 3.1 macro.

```
#include <mbctype.h>
#define iskana       _ismbbkana
#define iskpun       _ismbbkpunct
#define iskmoji     _ismbbkalpha
#define isalkana    _ismbbalpha
#define ispnkana    _ismbbpunct
#define isalnmkana  _ismbbalnum
#define isprkana    _ismbbprint
#define isgrkana    _ismbbgraph
#define iskanji     _ismbblead
#define iskanji2    _ismbbtrail
```

Additional reference words: 3.10 1.20 3.50 kbinf
KBCategory: kbother
KBSubcategory: wintldev

Obtaining a Console Window Handle (HWND)

PSS ID Number: Q124103
Authored 19-Dec-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

It may be useful to manipulate a window associated with a console application. The Win32 API provides no direct method for obtaining the window handle associated with a console application. However, you can obtain the window handle by calling FindWindow(). This function retrieves a window handle based on a class name or window name.

Call GetConsoleTitle() to determine the current console title. Then supply the current console title to FindWindow().

MORE INFORMATION

Because multiple windows may have the same title, you should change the current console window title to a unique title. This will help prevent the wrong window handle from being returned. Use SetConsoleTitle() to change the current console window title. Here is the process:

1. Call GetConsoleTitle() to save the current console window title.

2. Call SetConsoleTitle() to change the console title to a unique title.

3. Call Sleep(40) to ensure the window title was updated.

4. Call FindWindow(NULL, uniquetitle), to obtain the HWND
   this call returns the HWND -- or NULL if the operation failed.

5. Call SetConsoleTitle() with the value retrieved from step 1, to
   restore the original window title.

You should test the resulting HWND. For example, you can test to see if the returned HWND corresponds with the current process by calling GetWindowText() on the HWND and comparing the result with GetConsoleTitle().

The resulting HWND is not guaranteed to be suitable for all window handle operations.

Sample Code

The following function retrieves the current console application window

handle (HWND). If the function succeeds, the return value is the handle of the console window. If the function fails, the return value is NULL. Some error checking is omitted, for brevity.

```
HWND GetConsoleHwnd(void)
{
    #define MY_BUFSIZE 1024 // buffer size for console window titles
    HWND hwndFound;         // this is what is returned to the caller
    char pszNewWindowTitle[MY_BUFSIZE]; // contains fabricated WindowTitle
    char pszOldWindowTitle[MY_BUFSIZE]; // contains original WindowTitle

    // fetch current window title

    GetConsoleTitle(pszOldWindowTitle, MY_BUFSIZE);

    // format a "unique" NewWindowTitle

    wsprintf(pszNewWindowTitle,"%d/%d",
                GetTickCount(),
                GetCurrentProcessId());

    // change current window title

    SetConsoleTitle(pszNewWindowTitle);

    // ensure window title has been updated

    Sleep(40);

    // look for NewWindowTitle

    hwndFound=FindWindow(NULL, pszNewWindowTitle);

    // restore original window title

    SetConsoleTitle(pszOldWindowTitle);

    return(hwndFound);
}
```

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg kbcode
KBSubcategory: BseCon UsrWndw

Obtaining Public Domain Information About Windows Sockets

PSS ID Number: Q115045
Authored 18-May-1994 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0
- Microsoft Windows SDK for Windows, version 3.1

SUMMARY

Anyone with FTP access to the Internet can obtain a vast amount of information pertinent to the Windows Sockets API. The information is available through anonymous FTP to sunsite.unc.edu in the /pub/micro/pc-stuff/ms-windows/winsock directory. In the winsock directory, there is a wealth of technical information, including specifications, applications, DLLs for testing, and developers' guides not only from Microsoft, but also from other vendors of Windows Sockets.

The following list gives a brief overview of the information contained in the directory. This information can also be obtained from the READ.ME file in the location mentioned above.

FAQ

A list of frequently asked questions, with companion answers. Please send additional questions or answers to towfiq@Microdyne.COM.

apps/

Applications that have been written to run on top of the Windows Sockets DLL. See the file READ.ME in the winsock directory for a description of each application.

./incoming/

A directory where submissions to the archive (such as sample programs) should be placed.

packages/

The winsock directory contains different vendors' WINSOCK.DLLs for testing purposes as well as Windows Sockets development packages. See the file READ.ME in this directory for a description of each package.

press-releases/

Some of the press releases about the Windows Sockets.

winsock-archive/

An archive of the winsock@Sunsite.UNC.Edu mailing list. (Send subscription requests to listserv@SunSite.UNC.Edu.)

wsguide.doc

The "Windows Sockets Guide," by Martin Hall, JSB (martinh@jsbus.com) in Microsoft Word for Windows format.

wsguide.ps

The "Windows Sockets Guide," by Martin Hall, JSB (martinh@jsbus.com) in PostScript format. In the subdirectories winsock-1.0 and winsock-1.1, you can find the specification in many different formats.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbref
KBSubCategory: NtwkMisc

Open File Dialog Box -- Pros and Cons

PSS ID Number: Q74612
Authored 24-Jul-1991 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

Applications can call the common dialog library (COMMDLG.DLL) function GetOpenFileName() to retrieve the name of a file that the user wants to manipulate. Using this DLL provides a common interface for opening files across applications and also eliminates dialog-box message processing within the application's code. However, the application must initialize a structure specific to the dialog box. This article discusses the benefits and costs of using the Open File common dialog, via GetOpenFileName().

MORE INFORMATION

When an application uses the Open File dialog box provided by the common dialog DLL, the primary benefit to that application's users will be a familiar interface. Once they learn how to open a file in one application that uses the DLL, they will know how to open a file in all applications that use it.

Features of the Open File dialog include:

- A list box of files, filtered by extension, in the current directory
- A list box of directories from root of current drive to current directory, plus subdirectories
- A combo box of file types to list in filename list box
- A combo box of drives available, distinguished by drive type
- An optional "Read Only" check box
- An optional "Help" button
- An optional application hook function to modify standard behavior
- An optional dialog template to add private application features

For the application's programming staff, the benefits of using the Open File common dialog will include:

- No dialog-box message processing necessary to implement the Open File dialog box
- Drive and directory listings are constructed by the DLL, not by the application
- A full pathname for the file to be opened is passed back to the application, and this name can be passed directly to the OpenFile function
- Offsets into the full pathname are also returned, giving the application access to the filename (sans pathname) and the file

extension without the need for parsing
- The application can pass in its own dialog box template, in which case the DLL will use that template instead of the standard template
- The application can provide a dialog hook function to extend the interface of the DLL or to change how events are handled
- The application can choose to have a single filename or multiple filenames returned from the dialog box

The cost for the programming staff could be in adapting previously written application code to handle the common dialog interface. While making this change is straightforward, it does require coding time. If the application only needs a filename with no path, the Open File common dialog is probably not appropriate.

For more information on using the Open File common dialog, query on GetOpenFileName().

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCmnDlg

OpenComm() and Related Flags Obsolete Under Win32

PSS ID Number: Q94990
Authored 28-Jan-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

OpenComm(), a Windows 3.1 application programming interface (API), is obsolete under Windows NT and is not in the Win32 API. Note that the flags, IE_BADID, IE_BAUDRATE, IE_BYTESIZE, IE_DEFAULT, IE_HARDWARE, IE_MEMORY, IE_NOPEN, and IE_OPEN are obsolete, but are still in the header files.

OpenComm() is provided for 16-bit Windows-based applications running under Windows on Win32 (WOW).

MORE INFORMATION

Under Win32, CreateFile() is used to create a handle to a communications resource (for example, COM1). The fdwShareMode parameter must be 0 (exclusive access), the fdwCreate parameter must be OPEN_EXISTING, and the hTemplate parameter must be NULL. Read, write, or read/write access can be specified and the handle can be opened for overlapped I/O.

ReadFile() and WriteFile() are used for communciations I/O. The TTY sample program shipped with the Win32 Software Development Kit (SDK) demonstrates how to do serial I/O in a Win32-based application.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseCommapi

Overlapping Controls Are Not Supported by Windows

PSS ID Number: Q79981
Authored 14-Jan-1992 Last modified 19-May-1995

The information in this article applies to:

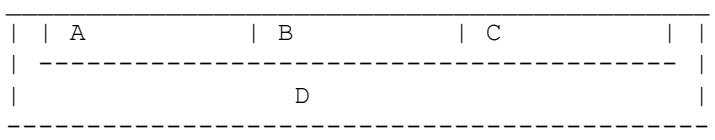
- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

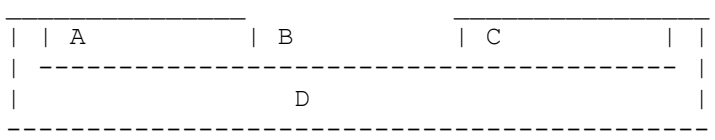
Child window controls should not be overlapped in applications for the Windows operating system. When one control overlaps another control, or another child window, the borders shared by the controls may not be drawn properly. Overlapping controls may confuse the user of the application because clicking the mouse in the common area may not activate the control that the user intended to activate. This behavior is a consequence of the way that Windows is designed.

MORE INFORMATION

The following example illustrates the painting problems caused by the ambiguity of overlapping borders. Consider three edit controls, called A, B and C, which overlap each other, and an enclosing child window D:



Assume that control B has the focus. If this set of controls is covered by another window, which is subsequently moved away, Windows will send a series of client and nonclient messages to each of the controls and to the enclosing child window. The result of these messages may appear as the illustration below, where the portion of window B's border that overlapped with part of window D's border is missing:



Repainting problems related to overlapping controls may vary depending on the version of Windows used.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95 CS_PARENTDC WS_CLIPCHILDREN
WS_CLIPSIBLINGS WM_NCPAINT WM_PAINT

KBCategory: kbprg
KBSubcategory: UsrDlgs

Overview of the Windows 95 Virtual Address Space Layout

PSS ID Number: Q125691
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

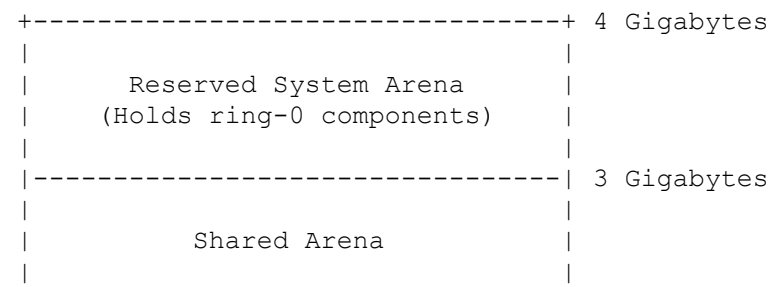
SUMMARY

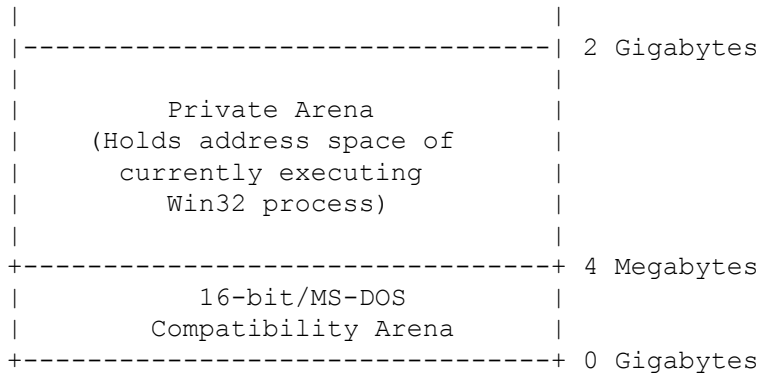
The virtual memory management mechanism in Microsoft Windows 95 makes it possible to execute Win32-based, 16-bit-based, and MS-DOS-based applications simultaneously. To accomplish this, the virtual memory manager uses a virtual address space layout that is considerably different from that used by Microsoft Windows version 3.x and that is slightly different from that used by Microsoft Windows NT. Although the differences from Windows NT are slight, they are important.

The memory manager in Windows 95 uses paging and 32-bit linear addressing to provide a full 32-bit virtual address space that has a maximum size of four gigabytes (GB). This four-GB address space is partitioned by the memory manager into four major sections, known as arenas, that are used for different types of applications and parts of the system. The first arena, from zero to four megabytes (MB) exists for compatibility with applications based on Windows version 3.1x and MS-DOS. The next arena, from four MB to two GB, is the private address space for each Win32 process. The third arena, from two to three GB, is a shared address space that contains memory mapped files and the 16-bit components. Finally, the fourth arena, from three to four GB, is reserved for the system's use.

MORE INFORMATION

The following diagram shows the overall virtual address space layout used in Windows 95. The Compatibility Arena holds the current virtual machine and other software. Each Win32 process gets its own private address space in which to execute. The Private Arena contains the currently executing Win32 process's private address space. All 16-bit-based applications and DLLs, including the 16-bit Windows system components, reside in the Shared Arena. Finally, the Reserved System Arena is used to store all ring-0 code such as the virtual machine manager and virtual device drivers. It is not accessible by either 16-bit-based or Win32-based applications.





Each arena has a specific purpose and is described in detail below.

16-bit/MS-DOS Compatibility Arena

The first four megabytes of the system's address space is reserved by the system and is accessible to 16-bit and MS-DOS software for compatibility. The current virtual machine occupies the lowest megabyte of this area. The remaining three megabytes are mostly empty space but may contain MS-DOS device drivers and Terminate & Stay Resident (TSR) programs.

The 16-bt/MS-DOS Compatibility Arena is not accessible to Win32 processes for reading or writing. This means Win32 processes may may not allocate memory, load DLLs, or be loaded below the four megabyte (MB) address.

Private Arena

The private arena holds the private address space of the currently executing Win32 process. Because every Win32 process gets its own address space, the contents of this arena will depend upon which process is currently executing. The memory manager maps the pages of a process's private address space so that other processes cannot access it and corrupt the process. The process's code, data, and dynamically-allocated memory all exist in the private address space.

With the exception of the system's shared DLLs (USER32.DLL, GDI32.DLL, and KERNEL32.DLL), all DLLs loaded by the process are mapped into the process's private address space. Windows extension DLLs such as SHELL32.DLL, COMCTL32.DLL, and COMDLG32.DLL are not system shared DLLs and are mapped into the process's private address space.

Because console applications are Win32-based applications without graphical user interfaces, they too get their own private address spaces, as do Win32 graphical user interface (GUI) applications.

The minimum load address for a Win32 process in Windows 95 is four MB because the first four megabytes are reserved for the Compatibility Arena.

Shared Arena

The shared arena is unique to Windows 95. This arena contains components that must be mapped into every process's address space. All of the pages in this arena are mapped identically in every process.

The 16-bit global heap, which contains all 16-bit-based applications, DLLs, and 16-bit system DLLs, resides in the shared arena. The Win32 shared system DLLs (USER32.DLL, GDI32.DLL, and KERNEL32.DLL) are also located in the shared arena.

Unlike the Reserved System Arena, the shared arena is readable and writable by Win32 and 16-bit processes alike. This doesn't mean they are free to get memory directly from this address space. All 16-bit-based applications and DLLs actually are located in the 16-bit global heap, so they allocate memory from the 16-bit global heap; when this heap needs to be grown, KRNL386.EXE gets the memory from the shared arena.

Win32 processes may not allocate memory directly from the shared arena, but they always use it for mapping views of file mappings. Unlike Windows NT, where views of file mappings always are placed in the private address space, Windows 95 holds views of file mappings in the shared arena.

The DOS Protected Mode Interface (DPMI) server's memory pool is located in the Shared Arena. Thus, calls to the DPMI server to allocate memory will result in memory that is globally accessible.

Sometimes, a virtual device driver (VxD) may need to map a buffer passed to it by a Win32 process into globally accessible memory so that the buffer can be accessed even if the process isn't in context. By calling _LinPageLock virtual machine manager service with the PAGEMAPGLOBAL flag, a VxD can obtain a linear address in the shared arena that corresponds to the buffer passed to it by the Win32 process.

Reserved System Arena

The reserved system contains the code and data of all ring-0 components such as the virtual machine manager, DOS extender, DPMI server, and virtual device drivers. This arena is used exclusively by ring-0 components and not addressable by ring 3 code, such as MS-DOS-based, 16-bit-based, and Win32-based applications and DLLs.

Additional reference words: 4.00 layout memory virtual
KBCategory: kbprg
KBSubcategory: BseMm

Owner-Draw Buttons with Bitmaps on Non-Standard Displays

PSS ID Number: Q67715
Authored 12-Dec-1990 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

If an application contains an owner-draw button that paints itself with a bitmap, the application's resources must contain a set of bitmaps appropriate to each display type on which the application might run.

If the application's resources do not contain bitmaps suitable for the display on which the application is running, the application can use the default 3-D button appearance by changing the button style to BS_PUSHBUTTON from BS_OWNERDRAW.

Under Windows 95, there is a new style BS_BITMAP which applications might find easier to use.

Changing the style of a button is possible in Windows version 3.0; however, this technique is not guaranteed to be supported in future releases of Windows.

MORE INFORMATION

An owner-draw button can use bitmaps to paint itself. When an application contains this type of owner-draw button, it must also contain a set of bitmaps appropriate for each display type on which the application might run. Each set of bitmaps has a normal "up" bitmap and a depressed "down" bitmap to implement the 3-D effects. The most common standard Windows display types are: CGA, EGA, VGA, 8514/a, and Hercules Monochrome. The dimensions and aspect ratio of the display affect the appearance of the bitmap. For example, a monochrome bitmap designed for VGA will display correctly on an 8514/a and any other display with a 1:1 aspect ratio.

If an application determines that it does not contain an appropriate set of bitmaps for the current display type, then it should change the button style from BS_OWNERDRAW to BS_PUSHBUTTON. After the style has been changed and the button has been redrawn, the button will appear as a normal 3-D push button.

The following code fragment demonstrates how to change the style of a push button from owner-draw to normal:

```
    ...

    /*
```

```
     * hWndButton is assumed to be the handle to the button.
     * Note that lParam has a nonzero value, which forces the button
     * to be redrawn. This assures that the normal button appearance
     * will show after this message is sent.
     */
    SendMessage(hWndButton, BM_SETSTYLE, BS_PUSHBUTTON, 1L);

    ...
```

Additional reference words: 3.00 3.10 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Owner-Draw: Overview and Sources of Information

PSS ID Number: Q64327
Authored 31-Jul-1990 Last modified 16-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

Owner-draw controls are a new feature of Windows version 3.0. Because your application does all the drawing of the contents of the controls, you can customize them any way you like. Owner-draw controls are similar to predefined controls in that Windows will handle the control's functionality and mouse and keyboard input processing. However, you are responsible for drawing the owner-draw control in its normal, selected, and focus states.

You can create owner-draw controls from the menu, button, and list-box classes. You can create owner-draw combo boxes, but they must have the CBS_DROPDOWNLIST style (equates to a static text item and a list box). The elements of an owner-draw control can be composed of strings, bitmaps, lines, rectangles, and other drawing functions in any combination, in your choice of colors.

MORE INFORMATION

The Windows SDK sample application MENU demonstrates owner-draw menu items. The SDK sample application OWNCOMBO is a fairly large example of owner-draw and predefined list boxes and combo boxes.

The Microsoft Software Library contains simplified examples of an owner-draw push button, owner-draw list boxes, and an owner-draw drop-down list style combo box. Each of these examples includes descriptive text in a related Knowledge Base article. For additional information, please see the following article(s) in the Microsoft Knowledge Base:

ARTICLE-ID: Q64326
TITLE : Owner-Draw: Handling WM_DRAWITEM for Drawing Controls

-or-

ARTICLE-ID: Q64328
TITLE : Owner-Draw: 3-D Push Button made from Bitmaps with Text

-or-

ARTICLE-ID: Q65792
TITLE : Owner-Draw Example: Right - and Decimal Alignment

To respond to a WM_MEASUREITEM message, you MUST specify the height of the appropriate item in your control. Optionally, you can specify the item's width as well.

If you want to do something special when a string is deleted from a list box, you should process WM_DELETEITEM messages. Windows's default action is to erase the deleted string and to redraw the list box.

If you want to have control over the sorting for the order of items in a list box or combo box that does not have the *_HASSTRINGS style, you should specify the appropriate *_SORT style and process WM_COMPAREITEM messages. If the *_SORT and *_HASSTRINGS styles are present, Windows will automatically do the sorting without sending WM_COMPAREITEM messages. If *_SORT is not specified, WM_COMPAREITEM messages will not be generated and items will be displayed in the list box in the order in which they were inserted.

The heart of owner-draw controls is the response to WM_DRAWITEM messages. During this processing is when you draw an entire button or each individual item in a menu, list box, or combo box. Because Windows does not interfere in the drawing of owner-draw controls, your application must draw the specified control item. The display of the control must indicate the state of the control. Common states are as follows:

1. Focus state (has the focus or not)

2. Selection state (selection or not)

3. Emphasis state (active, grayed, or disabled) (less common to process)

See the article titled "Owner-Draw: Handling WM_DRAWITEM for Drawing Controls" for information about drawing controls in their various states. Familiarity with the WM_DRAWITEM message and the various control states is extremely helpful before trying to follow the code examples.

Under Windows 95 and Windows 3.51, the BS_BITMAP style allows buttons to display bitmaps without using owner-draw.

Additional reference words: 3.00 3.10 3.50 4.00 owndraw od owner draw
KBCategory: kbprg
KBSubcategory: UsrCtl

Owners Have Special Access to Their Objects

PSS ID Number: Q130543
Authored 22-May-1995 Last modified 23-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5 and 3.51

The Windows NT operating system allows the owner of an object to determine what types of access are granted or denied for a given user. This is referred to as Discretionary Access Control (DAC). In addition to granting the generic read and write types of access, the owner of an object can also grant other users the right to modify the access allowed to the object.

The access right to view the access allowed on an object is called READ_CONTROL. This is often granted as part of a generic right. The access right that allows someone to change the access for an object is called WRITE_DAC.

The owner of an object can always request WRITE_DAC and READ_CONTROL access to the object. This prevents a situation where the owner of an object can not manipulate the object. This also allows owners of objects to restrict their own access to the object (to guard against accidents) without having to explicitly grant READ_CONTROL and WRITE_DAC access to their accounts.

Additional reference words: 3.10 3.50 AccessCheck
KBCategory: kbprg
KBSubcategory: BseSecurity

PAGE_READONLY May Be Used as Discardable Memory

PSS ID Number: Q94947
Authored 26-Jan-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

Virtual memory pages marked as PAGE_READONLY under Win32 may be used the way discardable segments of memory are used in Windows 3.1. These virtual memory pages are by default not "dirty," so the system may use them (zeroing them first if necessary) without having to first write their contents to disk.

From a system resource perspective, PAGE_READONLY is treated similar to discardable memory under Windows 3.1 when the system needs to free up resources. From a programming standpoint, the system automatically re-reads the memory when the data is next accessed (for example, we attempt to access our page when it has been "discarded," a page fault is generated, and the system reads it back in). Memory-mapped files are convenient for setting up this type of behavior.

If a PAGE_READONLY memory page becomes dirty [by changing the protection via VirtualProtect() to PAGE_READWRITE, changing the data, and restoring PAGE_READONLY], the page will be written to disk before the system uses it.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseMm

Panning and Scrolling in Windows

PSS ID Number: Q11619
Authored 02-Nov-1987 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

When using bitmaps, the mapping mode is ignored and physical units (in other words, MM_TEXT pixels) are used. It is not necessary to use the extent/origin routines to keep track of the logical origin.

If scrolling is desired and if there are no child windows in the client area, it is best to BitBlt the client area to scroll it, and PatBlt the uncovered area with the default brush.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrPnt

Passing a Pointer to a Member Function to the Win32 API

PSS ID Number: Q102352
Authored 03-Aug-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

Many of the Win32 application programming interfaces (APIs) call for a callback routine. One example is the lpStartAddr argument of CreateThread():

```
HANDLE CreateThread(lpsa, cbStack, lpStartAddr, lpvThreadParm,
   fdwCreate, lpIDThread)

LPSECURITY_ATTRIBUTES lpsa;    /* Address of thread security attrs */
DWORD cbStack;                    /* Initial thread stack size*/
LPTHREAD_START_ROUTINE lpStartAddr; /* Address of thread function */
LPVOID lpvThreadParm;       /* Argument for new thread*/
DWORD fdwCreate;               /* Creation flags*/
LPDWORD lpIDThread;               /* Address of returned thread ID */
```

When attempting to use a member function as the thread function, the following error is generated:

```
error C2643: illegal cast from pointer to member
```

The problem is that the function expects a C-style callback, not a pointer to a member function. A major difference is that member functions are called with a hidden argument called the "this" pointer. In addition, the format of the pointer isn't simply the address of the first machine instruction, as a C pointer is. This is particularly true for virtual functions.

If you want to use a member function as a callback, you can use a static member function. Static member functions do not receive the "this" pointer and their addresses correspond to an instruction to execute.

Static member functions can only access static data, and therefore to access nonstatic class members, the function needs an object or a pointer to an object. One solution is to pass in the "this" pointer as an argument to the member function.

MORE INFORMATION

This situation occurs with callback functions of other types as well, such as:

```
   DLGPROC                  GRAYSTRINGPROC
   EDITWORDBREAKPROC        LINEDDAPROC
   ENHMFENUMPROC            MFENUMPROC
   ENUMRESLANGPROC          PROPENUMPROC
   ENUMRESNAMEPROC          PROPENUMPROCEX
   ENUMRESTYPEPROC          TIMERPROC
   FONTENUMPROC             WNDENUMPROC
   GOBJENUMPROC
```

For more information on C++ callbacks, please see the May 1993 issue of the "Windows Tech Journal."

The following sample demonstrates how to use a static member function as a thread function, and pass in the "this" pointer as an argument.

Sample Code

```
#include <windows.h>

class A
{
public:
   int x;
   int y;

   A() { x = 0; y = 0; }

   static StartRoutine( A * );   // Compiles clean, includes "this" pointer
};

void main( )
{
   A a;

   DWORD dwThreadID;

   CreateThread( NULL,
      0,
      (LPTHREAD_START_ROUTINE)(a.StartRoutine),
      &a,                        // Pass "this" pointer to static member fn
      0,
      &dwThreadID
   );
}
```

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseMisc

Passing Security Information to SetFileSecurity()

PSS ID Number: Q102100
Authored 28-Jul-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

The SetFileSecurity() Win32 application programming interface (API) takes a pointer to a Security Descriptor. This is because SetFileSecurity() can set any of the following security information for a file:

   The owner identifier of the file
   The primary group identifier of the file
   The discretionary access-control list (DACL) of the file
   The system access-control list (SACL) of the file

When you pass the SD and SECURITY_INFORMATION structure to SetFileSecurity(), the SECURITY_INFORMATION structure identifies which security information is to be set. The SECURITY_INFORMATION structure is a DWORD that can be one of the following values:

   OWNER_SECURITY_INFORMATION
   GROUP_SECURITY_INFORMATION
   DACL_SECURITY_INFORMATION
   SACL_SECURITY_INFORMATION

Each of these values represents one of the security items listed above. The SD that is passed to SetFileSecurity() is simply a container for the security information being set for the specified file. SetFileSecurity() examines the value in the SECURITY_INFORMATION structure, extracts the appropriate information from the provided SD, and applies it to the specified file's SD.

Additional reference words: 3.10
KBCategory: kbprg
KBSubcategory: BseSecurity

PEN2CTL.VBX Custom Controls for Windows 95 Pen Services

PSS ID Number: Q130654
Authored 24-May-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) Version 4.0

SUMMARY

The PEN2CTL.VBX provides three pen edit custom controls to allow for quick development of pen-aware applications using Microsoft Windows 95 Pen Services. The application designer may substitute these custom controls for the standard input controls available in Visual Basic.

MORE INFORMATION

The PEN2CTL.VBX ships with version 4.0 of the Win32 SDK and is available in the Microsoft Software. It is installed along with the Pen 2.0 Samples Library.

To get PEN2CTL.VBX along with the PENVBX.HLP Help file and the sample application files discussed in this article, download PEN2CTL.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for PEN2CTL.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download PEN2CTL.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the SOFTLIB\MSLFILES directory
  Get PEN2CTL.EXE

The PEN2CTL.EXE self-extracting file contains:

- PENVBX.HLP, a help file that fully explains all the technical issues and describes how to interact with these pen-aware custom controls.

- Several files that comprise a sample Pen application that demonstrates how to use all the major features of the PEN2CTL.VBX.

- PEN2CTL.VBX, which is freely redistibutable and implements the three pen-aware custom edit controls: HEdit, BEdit, and lEdit.

- HEdit is the handwriting edit control. It accepts free-form input. This control is similar to the standard Visual Basic text box except data can be entered using a pen in addition to the normal keyboard input method.

- BEdit is the boxed edit control. It expands the properties of the handwriting edit control. It allows for additional manipulation of the writing area and provides the application with comb or box guides that accept pen input. Each segment or box accepts only a single character of input. This increases the accuracy of the recognition and in most cases is preferable to the handwriting edit control. The BEdit control also has the ability to provide a list of alternate words from which you may choose.

- lEdit is the pen ink edit control. It is similar to a picture box control in that it allows you to draw, erase, move, resize, manipulate, and format pen strokes (called ink) on the control. It also allows you to set background pictures and grid lines as well as create bitmaps of the background and/or ink.

While there are many similarities between the controls provided by the Pen 1.0 VBX (PENCNTRL.VBX) and those provided by PEN2CTL.VBX, there are also many differences. Those familiar with the Pen 1.0 VBX should refamiliarize themselves with PEN2CTL.VBX and its changes.

NOTE: Neither VBX is compatible with the other but both implement some of the same custom controls, so developers should make sure the correct VBX is installed in their development environment and is distributed with their application.

Additional reference words: 4.00 2.00
KBCategory: kbprg kbfile
KBSubcategory: WpenMisc WpenVB

Performing a Clear Screen (CLS) in a Console Application

PSS ID Number: Q99261
Authored 26-May-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

There is no Win32 application programming interface (API) that will clear the screen in a console application. However, it is fairly easy to write a function that will programmatically clear the screen.

MORE INFORMATION

The following function clears the screen:

```
void cls( HANDLE hConsole )
{
   COORD coordScreen = { 0, 0 };    /* here's where we'll home the
                                       cursor */
   BOOL bSuccess;
   DWORD cCharsWritten;
   CONSOLE_SCREEN_BUFFER_INFO csbi; /* to get buffer info */
   DWORD dwConSize;                 /* number of character cells in
                                       the current buffer */

   /* get the number of character cells in the current buffer */

   bSuccess = GetConsoleScreenBufferInfo( hConsole, &csbi );
   PERR( bSuccess, "GetConsoleScreenBufferInfo" );
   dwConSize = csbi.dwSize.X * csbi.dwSize.Y;

   /* fill the entire screen with blanks */

   bSuccess = FillConsoleOutputCharacter( hConsole, (TCHAR) ' ',
      dwConSize, coordScreen, &cCharsWritten );
   PERR( bSuccess, "FillConsoleOutputCharacter" );

   /* get the current text attribute */

   bSuccess = GetConsoleScreenBufferInfo( hConsole, &csbi );
   PERR( bSuccess, "ConsoleScreenBufferInfo" );

   /* now set the buffer's attributes accordingly */

   bSuccess = FillConsoleOutputAttribute( hConsole, csbi.wAttributes,
      dwConSize, coordScreen, &cCharsWritten );
   PERR( bSuccess, "FillConsoleOutputAttribute" );
```

```
   /* put the cursor at (0, 0) */

   bSuccess = SetConsoleCursorPosition( hConsole, coordScreen );
   PERR( bSuccess, "SetConsoleCursorPosition" );
   return;
}
```

Additional reference words: 3.10 3.50 4.00 95 clearscreen
KBCategory: kbprg
KBSubcategory: BseCon

Performing a Synchronous Spawn Under Win32s

PSS ID Number: Q125212
Authored 22-Jan-1995 Last modified 04-May-1995

The information in this article applies to:

- Microsoft Win32s, version 1.2

SUMMARY

Under Windows NT, you can synchronously spawn an application (that is, spawn an application and wait until the spawned application is terminated before continuing). To do so, call CreateProcess() to start the application, and pass the handle returned to WaitForSingleObject() to wait for the application to terminate. This is shown in the sample code in the "More Information" section in this article.

However, this method does not work under Win32s. Under Win32s, CreateProcess() does not return the process handle for 16-bit Windows-based applications, only for Win32-based application. Even if it did, the method described in the proceeding paragraph would not work under Win32s because WaitForSingleObject() returns TRUE immediately under Win32s.

In fact, there is no 32-bit only solution for this issue. The 32-bit WinExec() does not return an instance handle as the 16-bit WinExec() does. In addition, you cannot use GetExitCodeProcess() to find the exit status of 16-bit Windows-based applications in order to loop on their status. It is a limitation that GetExitCodeProcess() returns zero for 16-bit Windows-based applications on both Windows NT and Win32s.

The solution is to create a thunk to the 16-bit side and from the 16-bit side, solve the problem as you would normally solve it from a Windows-based application. Namely, start the application with WinExec() and use one of the Toolhelp APIs in a test loop to determine when the application is terminated. Alternatively, you can use EnumWindows() to determine when the application is terminated. The sample code below uses the Toolhelp APIs.

MORE INFORMATION

Sample code to perform a synchronous spawn is given below. The code is divided into three source files:

- The main application.
- The 32-bit side of the thunk.
- The 16-bit side of the thunk.

You can use the thunking code as is, calling SynchSpawn() in your own application as demonstrated in the main application below. For information on Universal thunks (including which header files and libraries to use), please see the "Win32s Programmer's Reference."

In all three modules, use the following header file SPAWN.H:

```
/*** Function Prototypes ****/

DWORD APIENTRY SynchSpawn( LPCSTR lpszCmdLine, UINT nCmdShow );

/*** Constants for Dispatcher ***/

#define SYNCHSPAWN     1
```

Main Application

This application attempts to synchronously spawn NOTEPAD under Windows NT and Win32s. NOTE: Under Win32s, NOTEPAD is a 16-bit application.

GetVersion() is used to detect the platform. Under Windows NT, CreateProcess() and WaitForSingleObject() perform the spawn. Under Win32s, the thunked routine SynchSpawn() is called.

```
/*** Main application code ***/

#include <windows.h>
#include "spawn.h"

void main()
{
   DWORD dwVersion;
   STARTUPINFO si = {0};
   PROCESS_INFORMATION pi = {0};

   dwVersion = GetVersion();

   if( !(dwVersion & 0x80000000) )  // Windows NT
   {
      si.cb = sizeof(STARTUPINFO);
      si.lpReserved = NULL;
      si.lpReserved2 = NULL;
      si.cbReserved2 = 0;
      si.lpDesktop = NULL;
      si.dwFlags = 0;

      CreateProcess( NULL,
                     "notepad",
                     NULL,
                     NULL,
                     TRUE,
                     NORMAL_PRIORITY_CLASS,
                     NULL,
                     NULL,
                     &si,
                     &pi );
      WaitForSingleObject( pi.hProcess, INFINITE );
   }
   else if( LOBYTE(LOWORD(dwVersion)) < 4 )  // Win32s
```

```
   {
      SynchSpawn( "notepad.exe", SW_SHOWNORMAL );
   }

   MessageBox( NULL, "Return from SynchSpawn", " ", MB_OK );

}
```

32-bit Side of Thunk

This DLL provides the 32-bit side of the thunk. If the DLL is loaded under Win32s, it initializes the thunk in its DllMain() by calling UTRegister(). The entry point SynchSpawn() packages up the arguments and calls the 16-bit side through the thunk.

```
/*** Code for 32-bit side of thunk ***/

#define W32SUT_32    // Needed for w32sut.h in 32-bit code

#include <windows.h>
#include "w32sut.h"
#include "spawn.h"

typedef BOOL (WINAPI * PUTREGISTER) ( HANDLE     hModule,
                  LPCSTR     lpsz16BitDLL,
                  LPCSTR     lpszInitName,
                  LPCSTR     lpszProcName,
                  UT32PROC * ppfn32Thunk,
                  FARPROC    pfnUT32Callback,
                  LPVOID     lpBuff
                );

typedef VOID (WINAPI * PUTUNREGISTER) (HANDLE hModule);

typedef DWORD (APIENTRY *PUT32CBPROC) (LPVOID lpBuff, DWORD dwUserDefined);

UT32PROC      pfnUTProc = NULL;
PUTREGISTER   pUTRegister = NULL;
PUTUNREGISTER pUTUnRegister = NULL;
PUT32CBPROC   pfnUT32CBProc = NULL;
int           cProcessesAttached = 0;
BOOL          fWin32s = FALSE;
HANDLE        hKernel32 = 0;

/*****************************************************************\
* Function: BOOL APIENTRY DllMain(HANDLE, DWORD, LPVOID)          *
*                                                                 *
*  Purpose: DLL entry point. Establishes thunk.                   *

BOOL APIENTRY DllMain(HANDLE hInst, DWORD fdwReason, LPVOID lpReserved)
{
   DWORD dwVersion;

   if ( fdwReason == DLL_PROCESS_ATTACH )
```

```
   {

    /*
     * Registration of UT need to be done only once for first
     * attaching process.  At that time set the fWin32s flag
     * to indicate if the DLL is executing under Win32s or not.
     */

      if( cProcessesAttached++ )
      {
         return(TRUE);         // Not the first initialization.
      }

      // Find out if we're running on Win32s
      dwVersion = GetVersion();
      fWin32s = (BOOL) (!(dwVersion < 0x80000000))
                       && (LOBYTE(LOWORD(dwVersion)) < 4);

      if( !fWin32s )
         return(TRUE);         // Win32s - no further initialization needed

      hKernel32 = LoadLibrary( "Kernel32.Dll" ); // Get Kernel32.Dll handle

      pUTRegister = (PUTREGISTER) GetProcAddress( hKernel32, "UTRegister"
);


      if( !pUTRegister )
         return(FALSE);        // Error- Win32s, but can't find UTRegister

      pUTUnRegister = (PUTUNREGISTER) GetProcAddress(hKernel32,
                                                     "UTUnRegister");

      if( !pUTUnRegister )
         return(FALSE);        // Error- Win32s, but can't find
UTUnRegister


      return (*pUTRegister)( hInst,           // Spawn32.DLL module handle
                             "SPAWN16.DLL",   // 16-bit thunk dll
                             "UTInit",        // init routine
              "UTProc",         // 16-bit dispatch routine
                             &pfnUTProc,      // Receives thunk address
                             pfnUT32CBProc,   // callback function
                             NULL );          // no shared memroy

   }
   if((fdwReason==DLL_PROCESS_DETACH)&&(0==--cProcessesAttached)&&fWin32s)
   {
      (*pUTUnRegister)( hInst );
      FreeLibrary( hKernel32 );
   }
} // DllMain()

/*****************************************************************\
```

```
* Function: DWORD APIENTRY SynchSpawn(LPTSTR, UINT)                    *
*                                                                      *
* Purpose: Thunk to 16-bit code                                        *

DWORD APIENTRY SynchSpawn( LPCSTR lpszCmdLine, UINT nCmdShow )
{
   DWORD Args[2];
   PVOID Translist[2];

   Args[0] = (DWORD) lpszCmdLine;
   Args[1] = (DWORD) nCmdShow;

   Translist[0] = &Args[0];
   Translist[1] = NULL;

   return( (* pfnUTProc)( Args, SYNCHSPAWN, Translist) );
}
```

16-bit Side of Thunk

This DLL provides the 16-bit side of the thunk. The LibMain() and WEP() of this 16-bit DLL perform no special initialization. The UTInit() function is called during thunk initialization; it stores the callback procedure address in a global variable. The UTProc() function is called with a code that indicates which thunk was called as its second parameter. In this example, the only thunk provided is for SynchSpawn(). The synchronous spawn is performed in the SYNCHSPAWN case of the switch statement in the UTProc().

NOTE: UTInit() and UTProc() must be exported. This can be done in the module definition (.DEF) file.

```
/* Code for 16-bit side of thunk.                                */
/* Requires linking with TOOLHELP.LIB, for ModuleFindHandle(). */

#ifndef APIENTRY
#define APIENTRY
#endif
#define W32SUT_16     // Needed for w32sut.h in 16-bit code

#include <windows.h>
#include <toolhelp.h>
#include <malloc.h>
#include "w32sut.h"
#include "spawn.h"

UT16CBPROC glpfnUT16CallBack;

/*********************************************************************\
* Function: LRESULT CALLBACK LibMain(HANDLE, WORD, WORD, LPSTR)      *
*                                                                    *
*  Purpose: DLL entry point                                          *

int FAR PASCAL LibMain( HANDLE hLibInst, WORD wDataSeg,
```

```
         WORD cbHeapSize, LPSTR lpszCmdLine)
{
   return (1);
} // LibMain()

/*****************************************************************\
* Function: DWORD FAR PASCAL UTInit(UT16CBPROC, LPVOID)           *
*                                                                 *
*  Purpose: Universal Thunk initialization procedure              *

DWORD FAR PASCAL UTInit( UT16CBPROC lpfnUT16CallBack, LPVOID lpBuf )
{
   glpfnUT16CallBack = lpfnUT16CallBack;
   return(1);   // Return Success
} // UTInit()


/*****************************************************************\
* Function: DWORD FAR PASCAL UTProc(LPVOID, DWORD)                *
*                                                                 *
*  Purpose: Dispatch routine called by 32-bit UT DLL              *

DWORD FAR PASCAL UTProc( LPVOID lpBuf, DWORD dwFunc)
{
   switch (dwFunc)
   {
      case SYNCHSPAWN:
      {
          HMODULE hMod;
          MODULEENTRY FAR *me;
          UINT hInst;
          LPCSTR lpszCmdLine;
          UINT nCmdShow;
          MSG msg;
          BOOL again=TRUE;

          /* Retrieve the command line arguments stored in buffer */

          lpszCmdLine = (LPSTR) ((LPDWORD)lpBuf)[0];
          nCmdShow = (UINT) ((LPDWORD)lpBuf)[1];

          /* Start the application with WinExec() */

          hInst = WinExec( lpszCmdLine, nCmdShow );
          if( hInst < 32 )
             return 0;

          /* Loop until the application is terminated. The Toolhelp API
           * ModuleFindHandle() returns NULL when the application is
           * terminated. NOTE: PeekMessage() is used to yield the
           * processor; otherwise, nothing else could execute on the
           * system.
           * /

          hMod = GetModuleHandle( lpszCmdLine );
```

```
            me = (MODULEENTRY FAR *) _fcalloc( 1, sizeof(MODULEENTRY) );
            me->dwSize = sizeof( MODULEENTRY );
            while( NULL != ModuleFindHandle( me, hMod ) && again )
            {
               while( PeekMessage( &msg, NULL, 0, 0, PM_REMOVE ) && again )
               {
                  if(msg.message == WM_QUIT)
                  {
                     PostQuitMessage(msg.wParam);
                     again=FALSE;
                  }
                  TranslateMessage(&msg);
                  DispatchMessage(&msg);
               }
            }
            return 1;
         }

      } // switch (dwFunc)

      return( (DWORD)-1L ); // We should never get here.
   } // UTProc()

   /*****************************************************************\
   * Function: int FAR PASCAL _WEP(int)                              *
   *                                                                 *
   *  Purpose: Windows exit procedure                                *

   int FAR PASCAL _WEP( int bSystemExit )
   {
      return (1);
   } // WEP()
```

Additional reference words: 1.20 win16
KBCategory: kbprg kbcode
KBSubcategory: W32s

PHONECAPS for Phones That Don't Report Button States

PSS ID Number: Q108308
Authored 08-Dec-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Telephony Software Development Kit (SDK) version 1.0 for Windows version 3.1
- Microsoft Win32 SDK, version 4.0

A Windows Telephony application can call phoneGetDevCaps to inquire about a phone's telephony capabilities. The resulting function in the TAPI (Telephony application programming interface) service provider is TSPI_phoneGetDevCaps. If the phone device does not report button states, it is acceptable for the service provider to set the dwButtonModeSize/Offset fields in the PHONECAPS structure to 0 (zero). Alternatively, the provider could describe the button(s) as PHONEBUTTONMODE_DUMMY.

As a clarification, setting the dwButtonModeSize to 0 or describing the buttons as PHONEBUTTON_DUMMY does not prohibit the service provider from setting and using the dwButtonFunctionsSize and dwButtonFunctionsOffset members of the PHONECAPS structure.

Additional reference words: 1.00 3.10 4.00 95
KBCategory: kbprg
KBSubcategory: MsgTapi

Physical Memory Limits Number of Processes/Threads

PSS ID Number: Q94840
Authored 19-Jan-1993 Last modified 14-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

Each time Windows NT creates an object, such as a process or a thread, it must allocate a certain amount of physical memory (nonpaged pool) for its support. The amount of memory that is needed for support of a process object is significantly higher than the memory requirement for support of a thread object. The amount of memory that is required for a thread object on a RISC machine is higher than the memory requirement for a thread on an x86 machine, due to the greater number and size of registers on the RISC machines.

Due to the physical memory requirement of processes and threads, programs that use the CreateProcess() and CreateThread() APIs should be careful to check their return codes to detect out-of-memory conditions.

On Windows NT 3.5, each time a process is created it reserves the minimum working set of memory. On a 32 MB system, the default minimum working set is 200 KB. Therefore, on a 32 MB system, you can create ~100 processes. You can lower your minimum working set to 80 KB (the lowest allowed) with the following call:

```
SetProcessWorkingSetSize( (HANDLE)(-1), 20*4096, 100*4096 );
```

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseProcThrd

Placing a Caret After Edit-Control Text

PSS ID Number: Q12190
Authored 16-Oct-1987 Last modified 25-May-1995

-
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

-

The EM_SETSEL message can be used to place a selected range of text in a Windows edit control. If the starting and ending positions of the range are set to the same position, no selection is made and a caret can be placed at that position. To place a caret at the end of the text in a Windows edit control and set the focus to the edit control, do the following:

```
   hEdit = GetDlgItem( hDlg, ID_EDIT );    // Get handle to control
   SetFocus( hEdit );
   SendMessage( hEdit, EM_SETSEL, 0, MAKELONG(0xffff,0xffff) );
```

It is also possible to force the caret to a desired position within the edit control. The following code fragment shows how to place the caret just to the right of the Nth character:

```
   hEdit = GetDlgItem( hDlg, ID_EDIT );    // Get handle to control
   SetFocus( hEdit );
   SendMessage( hEdit, EM_SETSEL, 0, MAKELONG(N,N) );
                                        // N is the character position
```

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Placing Captions on Control Windows

PSS ID Number: Q77750
Authored 24-Oct-1991 Last modified 16-May-1995

-
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

-

The SetWindowText() function can be used to place text into the caption bar specified for a control window. The control must have the WS_CAPTION style for the caption to be visible.

This technique does not work with edit controls because the SetWindowText() function specifies the contents of the edit control, not its caption.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Placing Double Quotation Mark Symbol in a Resource String

PSS ID Number: Q47674
Authored 03-Aug-1989 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

To specify a set of double quotation marks within a string in an application's resource (RC) file, use two double quotation mark characters in succession, as in the following example:

Specify the following string in the RC file:

```
"The letter ""Q"" is quoted."
```

The following string will appear in the compiled resource (RES) file:

```
The letter "Q" is quoted.
```

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrRsc

Placing Text in an Edit Control

PSS ID Number: Q32785
Authored 11-Jul-1988 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows, versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

Text is placed into an edit control by calling SetDlgItemText() or by sending the WM_SETTEXT message to the edit control window, with lParam being a pointer to a null-terminated string. This message can be sent in two ways:

1. SendMessage(hwndEditControl, WM_SETTEXT, ...

2. SendDlgItemMessage(hwndParent, ID_EDITCTL, WM_SETTEXT...

   NOTE: hwndParent is the window handle of the parent, which may be a dialog or window. ID_EDITCTL is the ID of the edit control.

Text is retrieved from an edit control by calling GetDlgItemText() or by sending the WM_GETTEXT message to the edit control window, with wParam being the maximum number of bytes to copy and lParam being a far pointer to a buffer to receive the text. This message can be sent in two ways:

1. SendMessage(hwndEditControl, WM_GETTEXT, ...

2. SendDlgItemMessage(hwndParent, ID_EDITCTL, WM_GETTEXT...

   NOTE: hwndParent is the window handle of the parent, which may be a dialog or window. ID_EDITCTL is the ID of the edit control.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Points to Remember When Writing a Debugger for Win32s

PSS ID Number: Q121093
Authored 26-Sep-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s, versions 1.1, 1.15, and 1.2

SUMMARY

This article is intended for developers of debugging tools for the Win32s environment. It covers the issues that should be to taken into consideration while writing debugging tools for the Win32s environment.

Overall, the code of a debugger for the Win32s environment is similar to the code of a debugger for the Windows NT environment. There are special points that must be considered before and while writing debugging tools for the Win32s environment. They are:

- Using the WaitForDebugEvent() API Function
- Debugging Shared Code
- Getting and Setting Thread Context
- Tracing Through Mixed 16- and 32-bit Code
- Using Asynchronous Stops
- Identifying System DLLs
- Understanding Linear and Virtual Addresses
- Reading and Writing Process Memory
- Accessing the Thread Local Storage (TLS)

Each of these is discussed in detail in the More Information section below.

MORE INFORMATION

The following information is specific to writing a debugger for the Win32s environment.

Using the WaitForDebugEvent() API Function

The WaitForDebugEvent() API function waits for a debugging event to occur in a process being debugged. Use it to trap debugging events.

Because of the non-preemptive nature of Windows version 3.1, it is not

possible to guarantee the timeout functionality. For this reason, the dwTimeout parameter was implemented differently in Win32s. In Win32s, if dwTimeout is zero, the WaitForDebugEvent() function behaves as documented in the "Win32 Programmer's Reference." Otherwise, the function waits indefinitely, until a debug event occurs or until a message is received for that process.

Make sure the function returns if a message is received, so that the calling process can respond to messages. If WaitForDebugEvent() returns because a debug event has occurred, the return value is TRUE. Otherwise, the return value is FALSE. In Win32, a FALSE return value means failure. Have the calling process use SetLastError() to set the error value to 0 before calling WaitForDebugEvent(). Then if the return value is FALSE and error value returned by GetLastError() is still zero, it means a message arrived.

The following code fragment demonstrates the use of WaitForDebugEvent() in the message loop:

```
   while( GetMessage(&msg, NULL, NULL, NULL) )
   {
      TranslateMessage(&msg); /* Translate virtual key codes */
      DispatchMessage(&msg);  /* Dispatch message to window */

      SetLastError( 0 );      /* Set error code to zero  */
      if( WaitForDebugEvent(&DebugEvent, INFINITE) )
      {
      /* Process the debug event */
         ProcessDebugEvents( &DebugEvent );
      }
      else
      {
         if( GetLastError() != 0 )
         {
         /* Handle error condition. */
         }
      }
   }
```

Debugging Shared Code

Under Win32s, all processes run in a single address space. For that reason, if a debugger sets a breakpoint in shared code, all processes will encounter this breakpoint, even those that are not being debugged. For these processes, the debugger should restore the code, let the process execute the restored instruction, and then reset the breakpoint. The problem is that in order to do these operations, the debugger needs a handle to the process thread.

The debugger does not have a handle for the process thread of a process it did not create. To get the handle, Win32s supports a new function, OpenThread(), which is not a part of the Win32 API.

```
   HANDLE OpenThread(dwThreadId);
```

```
DWORD dwThreadId;  /* The thread ID */

Parameter description:

   dwThreadId - Specifies the thread identifier of the thread to open.

Returns:

   If the function succeeds, the return value is an open handle of the
   specified thread; otherwise, it is NULL. To get extended error
   information, use the GetLastError() API.

Comments:

   The handle returned by OpenThread() can be used in any function that
   requires a handle to a thread.
```

OpenThread() is exported by KERNEL32.DLL, but is not included in any of the SDK import libraries.

To create an import library on a Windows NT development machine:

1. Place the following contents into a file named W32SOPTH.C:

```
#include <windows.h>

HANDLE WINAPI OpenThread(DWORD dwThreadId)
{
   return (HANDLE)NULL;
}
```

2. Place the following contents into a file named W32SOPTH.DEF:

```
LIBRARY kernel32

DESCRIPTION 'Win32s OpenThread library'

EXPORTS
   OpenThread
```

3. Place the following contents into a file named MAKEFILE:

```
w32sopth.lib: w32sopth.obj
   lib -out:w32sopth.lib -machine:i386 -def:w32sopth.def w32sopth.obj

w32sopth.obj: w32sopth.c
   cl /c w32sopth.c
```

4. Run the NMAKE utility from the directory that contains the files created in steps 1-3. This creates the W32SOPTH.LIB file.

The debugger should perform the following test: in the DEBUG_INFO structure returned by WaitForDebugEvent(), there is a thread ID. The debugger should check to see if this ID is one of the debugged processes. If it is not, the

debugger should call OpenThread() with the given thread ID as the parameter and receive a handle to the thread. Using this handle, the debugger should call GetThreadContext(), identify the breakpoint, restore the code, set the single step bit of EFlag, and resume the process by calling ContinueDebugEvent(). Then control returns to the debugger. The debugger restores the breakpoint. After dealing with the non-debugged process, the debugger must close the thread handle obtained from OpenThread() by using CloseHandle().

The following code fragment demonstrates how a debugger can handle breakpoints in the context of a non-debugged process:

```
LPDEBUG_EVENT lpEvent;  /* Pointer to the debug event structure */
HANDLE hProc;           /* Handle to process */
HANDLE hThread;         /* Handle to thread */
CONTEXT Context;        /* Context structure */;
BYTE bOrgByte;          /* Original byte in the place of BP */
DWORD cWritten;         /* Number of bytes written to memory */
static DWORD dwBPLoc;   /* Breakpoint location */

/*
 * Other debugger functions:
 *
 * LookupThreadHandle -
 *    Receives a thread ID and returns a handle to the thread, if
 *    the thread created by the debugger, else returns NULL.
 */
HANDLE LookupThreadHandle(DWORD);

/*
 * LookupOriginalBPByte -
 *    Receives an address of a breakpoint and returns the original
 *    contents of the memory in the place of the breakpoint.
 *    The memory contents is returned in the byte buffer passed as
 *    a parameter.
 * Return value - If the breakpoint was set by the debugger the
 *   return value is TRUE, else FALSE.
 */
BOOL LookupOriginalBPByte( LPVOID, LPBYTE );

/* Handle debug events according to event types */
switch( lpEvent->dwDebugEventCode )
{
/* ... */
   case EXCEPTION_DEBUG_EVENT:
   /* Handle exception debug events according to exception type */
      switch( lpEvent->u.Exception.ExceptionRecord.ExceptionCode )
      {
      /* ... */
      case EXCEPTION_BREAKPOINT:
         /* Breakpoint exception */
         /* Look for the thread handle in the debugger tables */
            hThread = LookupThreadHandle( lpEvent->dwThreadId );
            if( hThread == NULL )
            {
```

```
      /* Not a debuggee */
      /* Get process and thread handles */
         hProc = OpenProcess( 0, FALSE, lpEvent->dwProcessId );
         hThread = OpenThread( lpEvent->dwThreadId );

      /* Get the full context of the processor */
         Context.ContextFlags = CONTEXT_FULL;
         GetThreadContext( hThread, &Context );

      /* We get the exception after executing the INT 3 */
         dwBPLoc = --Context.Eip;

      /* Restore the original byte in memory in the */

      /* place of the breakpoint */
         if( !LookupOriginalBPByte((LPVOID)dwBPLoc, &bOrgByte) )
         {
         /* Handle unfamiliar breakpoint */
         }
         else
         {
         /* Restore memory contents */
            WriteProcessMemory( hProc, (LPVOID)dwBPLoc,
            &bOrgByte, 1, &cWritten );

         /* Set the Single Step bit in EFlags */
            Context.EFlags |= 0x0100;
            SetThreadContext( hThread, &Context );
         }

      /* Free Handles */
         CloseHandle( hProc );
         CloseHandle( hThread );

      /* Resume the interrupted process */
         ContinueDebugEvent( lpEvent->dwProcessId,
                             lpEvent->dwThreadId, DBG_CONTINUE );
      }
      else
      {
      /* Handle debuggee breakpoint. */
      }
      break;

   case STATUS_SINGLE_STEP:
      hThread = LookupThreadHandle( lpEvent->dwThreadId );
      if( hThread == NULL )
      {
      /* Not a debuggee, just executed the original instruction */
      /* and returned to the debugger. */

      /* Get process handle */
         hProc = OpenProcess( 0, FALSE, lpEvent->dwThreadId );

      /* Restore the INT 3 instruction in the place of the BP */
```

```
                bOrgByte = 0xCC;
                WriteProcessMemory( hProc, (LPVOID)dwBPLoc,
                &bOrgByte, 1, &cWritten );

             /* Free Handle */
                CloseHandle( hProc );

             /* Resume the process */
                ContinueDebugEvent( lpEvent->dwProcessId,
                                    lpEvent->dwThreadId, DBG_CONTINUE );
             }
             else
             {
             /* Handle debuggee single-step. */
             }
             break;
       /* .... */
    }
 /* .... */
 }
```

This sample code does not contain code to handle error checking and return values from APIs. The assumption is that a non-debugged process generates a single step exception only when it is executing the instruction in the place of the breakpoint. The code for handling the single step exception does not handle debug registers.

Getting and Setting Thread Context

Because of architectural differences between Windows NT and Win32s, there is a difference in the way GetThreadContext() and SetThreadContext() work in Win32s. These functions return successfully only if they are called after returning from WaitForDebugEvent() with the EXCEPTION_DEBUG_EVENT value in the dwDebugEventCode field of the DEBUG_INFO structure and before calling ContinueDebugEvent(). At any other point, these APIs fail and GetLastError() returns ERROR_CAN_NOT_COMPLETE.

Tracing Through Mixed 16- and 32-bit Code

Occasionally, Win32-based applications switch to 16-bit mode and then go back to 32-bit mode. For example, part of the Windows API is implemented in Win32s by using thunks to connect to Windows version 3.1. That means that in order to call the API, Win32s switches to 16-bit mode, calls the corresponding API on the Windows version 3.1 side, and then returns to 32-bit mode.

Most debuggers do not allow tracing through 16-bit code. So when the code is about to switch to 16-bit mode, the debugger should trace over this code. To do so, Win32s supplies the DbgBackTo32 label. All calls to 16-bit code return through this address. The DbgBackTo32 label is exported by W32SKRNL.DLL. At this label, there is a RET instruction. After executing this RET instruction and immediately another following RET instruction, Win32s resumes execution at the application code, at the instruction

following the call to the thunked function. So if the debugger determines that the next call is into a thunk function, it can set a breakpoint at DbgBackTo32 and trace over this call.

Using Asynchronous Stops

The asynchronous stop key combination was set to CTRL+ALT+F11 in Win32s. It allows a 16-bit debugger to run at the same time as a 32-bit debugger. Each debugger can synchronously stop the other.

If the user presses CTRL+ALT+F11 when the executing code is 16-bit code, execution will not be interrupted until it returns to 32-bit code. This way, the debugger does not have to handle 16-bit code. If the user presses CTRL+ALT+F11 when the executing code is 32-bit code, execution is interrupted immediately.

Execution is interrupted by generating a single step exception. To handle the case where the user presses CTRL+ALT+F11 while 16-bit code is executing, the address of the exception is at a special Win32s label (W32S_BackTo32). This label is exported by W32SKRNL.DLL and is located a few instructions before DbgBackTo32. For more information on this see the "Tracing Through Mixed 16- and 32-bit Code" above.

The code at W32S_BackTo32 is system code and usually debuggers should not allow tracing through system code. But between W32S_BackTo32 and DbgBackTo32, the debugger may allow tracing through this specific code and also through the two following RET instructions. This will bring the user to the point in the application at which CTRL+ALT+F11 was pressed.

Identifying System DLLs

When tracing through application code, it is not desirable to trace into system DLL code. The main reason for this is that in many cases the code goes to 16-bit code. To enable the debugger to distinguish between system and user DLLs, all Win32s system DLLs contain an extra exported symbol called WIN32SYSDLL. The address of this symbol is meaningless. The existence of such a symbol indicates that this is a system DLL.

Understanding Linear and Virtual Addresses

Win32s uses flat memory address space as does Windows NT, but unlike Windows NT, the base of the code and data segments is not at zero. You must consider this when dealing with linear addresses -- such as hardware debug registers when setting a hardware breakpoint. When setting a hardware breakpoint, you need to add the base of the selector to the virtual address of the breakpoint and set the debug register with this value. If you do not do so, the code will run on Windows NT but not on Win32s.

The debugger needs to get the base address of the selectors by using the GetThreadSelectorEntry() function.

Similarly, when the hardware breakpoint is encountered, you must subtract

the selector base address from the contents of the debug register in order to read the process memory at the breakpoint location.

Reading and Writing Process Memory

When reading from or writing to process memory, all hardware breakpoints must be disabled. If you do not do so, accessing the memory locations pointed to by the debug registers will trigger the hardware breakpoints.

The following code demonstrates how a debugger can read process memory at the location of a read memory hardware breakpoint:

```
CONTEXT Context;
LDT_ENTRY SelEntry;
DWORD dwDsBase;
DWORD DR7;
BYTE Buffer[4];

/* Get Context */
Context.ContextFlags = CONTEXT_FULL | CONTEXT_DEBUG_REGISTERS;
GetThreadContext( hThread, &Context );

/* Calculate the base address of DS */
GetThreadSelectorEntry(hThread, Context.SegDs, &SelEntry);
dwDsBase = ( SelEntry.HighWord.Bits.BaseHi << 24) |
             (SelEntry.HighWord.Bits.BaseMid << 16) |
             SelEntry.BaseLow;

/*
 * Disable all hardware breakpoints before reading the process
 * memory. Not doing so will lead to nested breapoints.
 */
DR7 = Context.Dr7;
Context.Dr7 &= ~0x3FF;
SetThreadContext( hThread, &Context );

/* Read DWORD at the location of DR0 */
ReadProcessMemory( hProcess,
                   (LPVOID)((DWORD)Context.Dr0-dwDsBase),
                   Buffer, sizeof(Buffer), NULL);

/* Restore hardware breakpoints */
Context.Dr7 = DR7;
SetThreadContext( hThread, &Context );
```

Accessing the Thread Local Storage (TLS)

The lpThreadLocalBase field of the CREATE_PROCESS_DEBUG_INFO structure in Windows NT specifies the base address of a per-thread data block. At offset 0x2C within this block, there exists a pointer to an array of LPVOIDs. There is one LPVOID for each DLL/EXE loaded at process initialization, and that LPVOID points to Thread Local Storage (TLS). This gives a debugger

access to per-thread data in its debuggee's threads using the same algorithms that a compiler would use.

On the other hand, in Win32s, lpThreadLocalBase contains a pointer directly to the array of LPVOIDs, not the pointer to the per-thread data block.

Additional reference words: 1.10 1.15 1.20
KBCategory: kbprg kbcode
KBSubCategory: W32s

Possible Cause for ERROR_INVALID_FUNCTION

PSS ID Number: Q111838
Authored 20-Feb-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

The Win32 Tape API (application programming interface) functions may return an error code of ERROR_INVALID_FUNCTION if the operation being attempted is not supported by the tape device.

MORE INFORMATION

You can determine what operations are valid for a tape device by calling GetTapeParameters() to get the tape drive parameters. See the code fragment in the Sample Code section of this article for an example of how to call GetTapeParameters() for this information.

Once you have the TAPE_GET_DRIVE_PARAMETERS structure, you can check for a specific operation in the FeaturesLow and FeaturesHigh members of the TAPE_GET_DRIVE_PARAMETERS structure. This is also demonstrated in the Sample Code section.

Sample Code

```
/*
** This is a code fragment only and will not compile and run as is.
*/

DWORD dwRes, dwSize;
TAPE_GET_DRIVE_PARAMETERS parmDrive;

...

dwSize = sizeof(parmDrive);
dwRes = GetTapeParameters(hTape, GET_TAPE_DRIVE_INFORMATION,
                &dwSize, &parmDrive);
if (dwRes != NO_ERROR) {
    /* place error handling code here */
    exit(-1);
}

if (parmDrive.FeaturesLow & TAPE_DRIVE_ERASE_LONG)
    printf("Device supports the long erase technique.\n");

...
```

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseFileio

Possible Serial Baud Rates on Various Machines

PSS ID Number: Q99026
Authored 19-May-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

Computers running Windows NT may be unable to set the same serial baud rates due to differences in serial port hardware on various platforms and machines. These differences may be important to note when writing a serial communications application that runs on different Windows NT platforms.

The simplest way to determine what baud rates are available on a particular serial port is to call the GetCommProperties() application programming interface (API) and examine the COMMPROP.dwSettableBaud bitmask to determine what baud rates are supported on that serial port.

MORE INFORMATION

Some baud rates may be available on one machine and not on another because of differences in the serial port hardware used on the two machines. Most Intel 80x86 machines use a standard 1.8432 megahertz (MHz) clock speed on serial port hardware, and therefore most Intel machines can set the same baud rates. However, on other platforms, such as MIPS, there is no standard serial port clock speed. MIPS serial ports are known to exist with 1.8432 MHz, 3.072 MHz, 4.2336 MHz, and 8.0 MHz serial port clock chips. Future NT implementations on other platforms may have different serial port clock speeds as well.

Furthermore, certain requested baud rates are special-cased in the Windows NT serial driver so that they will work. The following are these special cases:

MHz	Requested Baud	Divisor	Resulting Baud Rate (+/- 1)
1.8432	56000	2	57600
3.072	14400	13	14769
4.2336	9600	28	9450
4.2336	14400	18	14700
4.2336	19200	14	18900
4.2336	38400	7	37800
4.2336	56000	5	52920
8.0	14400	35	14286
8.0	56000	9	55556

The actual baud rate can be calculated by dividing the divisor multiplied

by 16 into the clock rate. For example, for a 1.8432 MHz clock and a divisor of 2, the baud rate would be:

```
1843200 Hz / (2 * 16) = 57600
```

For all other cases, as long as the requested baud rate is within 1 percent of the nearest baud rate that can be found with an integer divisor, the baud rate request will succeed.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseCommapi

Posting Frequent Messages Within an Application

PSS ID Number: Q40669
Authored 26-Jan-1989 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

The object-oriented nature of Windows programming can create a situation in which an application posts a message to itself. When such an application is designed, care must be taken to avoid posting messages so frequently that system messages to the application are not processed. This article discusses two methods of using the PeekMessage() function to combat this situation.

MORE INFORMATION

In the first method, a PeekMessage() loop is used to check for system messages to the application. If none are pending, the SendMessage() function is used from within the PeekMessage() loop to send a message to the appropriate window. The following code demonstrates this technique:

```
while (fProcessing)
     {
     if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
          {
          if (msg.message == WM_QUIT)
               break;
          /* process system messages */
          }
     else
          {
          /* perform other processing */
          ...
          /* send WM_USER message to window procedure */
          SendMessage(hWnd, WM_USER, wParam, lParam);
          }
     }
```

In the second method, two PeekMessage() loops are used, one to look for system messages and one to look for application messages. PostMessage() can be used from anywhere in the application to send the messages to the appropriate window. The following code demonstrates this technique:

```
while (fProcessing)
     {
     if (PeekMessage(&msg, NULL, 0, WM_USER-1, PM_REMOVE))
          {
```

```
          if (msg.message == WM_QUIT)
               break;
          /* process system messages */
          }
     else if (PeekMessage(&msg, NULL, WM_USER, WM_USER, PM_REMOVE))
          /* process application messages */
     }
```

An application should use a PeekMessage() loop for as little time as possible. To be compatible with battery-powered computers and to optimize system performance, every Windows-based application should inform Windows that it is idle as soon and as often as possible. An application is idle when the GetMessage() or WaitMessage() function is called and no messages are waiting in the application's message queue.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMsg

PRB: "Out of Memory Error" in the Win32 SDK Setup Sample

PSS ID Number: Q114610
Authored 08-May-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SYMPTOMS

When a dialog box is shown using UIStartDlg() or a billboard is switched during the file copy operation, you may receive an "out of memory" error. The error will also occur in any setup program based on a modified version of the SDK sample.

CAUSE

The dialog box and billboard templates are stored as resources in MSCUISTF.DLL. This DLL (Dynamic Link Library) is not loaded at the beginning of the setup program but is rather loaded and unloaded [using LoadLibrary() and FreeLibrary()] around each call to UIStratDlg() and when billboards are switched. Hence, each time a dialog or billboard is displayed, floppy disk #1 has the potential of being accessed. If disks have been swapped due to the installation process such that disk #1 is no longer in the drive, you will receive an out of memory error when LoadLibrary() is called on MSCUISTF.DLL.

RESOLUTION

To solve the problem, call LoadLibrary() at the beginning of WinMain() and call FreeLibrary() and the end of WinMain(). This way the DLL is always in use and will never be unloaded until the setup is done.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool kbprb
KBSubcategory: TlsMss

PRB: "Routine Not Found" Errors in Windows Help

PSS ID Number: Q108722
Authored 16-Dec-1993 Last modified 28-Nov-1994

The information in this article applies to:

- Microsoft Win32s versions 1.0, 1.1, and 1.2

SYMPTOMS

When Freecell is running on Win32s, you may get some errors when you try to use Freecell Help. This has nothing to do with whether or not Win32s is installed correctly.

When you first open Help in Freecell running on Win32s, you will get a message box that says, "Routine Not Found." You will get that same message box when you choose the Find button.

When you choose any term in Freecell Help that has a dotted underline (a pop-up hot key), a window pops up but then quickly disappears. You will then get a message box that reads, "Help Topic Does Not Exist."

CAUSE

FREECELL.HLP was meant to be used with WINHLP32.EXE. FREECELL.HLP uses the advanced features of WINHLP32.EXE for full-text searching. WINHELP.EXE, which runs on Windows 3.1, does not support full-text searching. WINHLP32.EXE does not run on Win32s or Windows 3.1. As a result, everytime FREECELL.HLP tries to bind the Find button to the full-text searching functions, it fails, and WinHelp pops up an error message box.

RESOLUTION

You can still read the information in the help file. Although you cannot use the Find button to do full-text searches, you can use the Search button to do keyword searches. WINHLP32.EXE will not work on a MS-DOS + Windows + Win32s operating system. If you need to read the pop-up definitions or use the Find button, you can do so if you run Freecell Help on the Windows NT operating system.

Additional reference words: 1.00 1.10 1.20
KBCategory: kbprg kbprb
KBSubcategory: W32s

PRB: _getdcwd() Returns the Root Directory Under Win32s

PSS ID Number: Q98286
Authored 02-May-1993 Last modified 28-Nov-1994

The information in this article applies to:

- Microsoft Win32s versions 1.1 and 1.2

SYMPTOMS

In the following code segment, _getdcwd() always returns the root:

```
_getdcwd( 3, cBuf, MAX_PATH );
MessageBox( hWnd, cBuf, "Drive 3 <C drive>", MB_OK );
```

Also, in the following code segment, _chdrive() and _getcwd() always return the root:

```
_chdrive( 3 );
_getcwd( cBuf, MAX_PATH );
MessageBox( hWnd, cbuf, "Drive 3 <C drive>", MB_OK );
```

CAUSE

When a Win32-based application starts on Win32s, the root is set as the current directory for any drive except the default drive.

RESOLUTION

The following code fragments work as expected under Win32s:

```
_getdcwd( 0, cBuf, MAX_PATH );
MessageBox( hWnd, cBuf, "Drive 0 <default drive>", MB_OK );

-or-

GetCurrentDirectory( sizeof (cBuf), cBuf );
MessageBox( hWnd, cBuf, "SCD", MB_OK );
```

MORE INFORMATION

Windows NT uses the current directory of a process as the initial current directory for the current drive of a child process. So for example, if the current directory in the command prompt (CMD.EXE) is C:\WINNT then the current directory of the child process will be C:\WINNT.

However, on Win32s, the current directory for any drive except the default drive is set to the root and not the current directory of the parent process. A Win32-based application running on Win32s calling _chdrive() or

SetCurrentDirectory() to change the drive GetCurrentDirectory or _getcwd() will then return the root. The _getdcwd() function is a composite of changing drives, getting the current directory of that drive, and change back to the original drive. Therefore, _getdcwd() will always return the root on Win32s.

Running the following sample to display the current directory of drives C and D under Windows NT properly displays the full path of the drive. Running the sample under Win32s always displays the root ("C:\", "D:\").

Sample Code

```
#include <direct.h>

...

   status =  _getdcwd(3, szPath, MAX_PATH);    // drive 3 == C:
   if( status != NULL )
   {
      MessageBox( hWnd, szPath, "Current working directory on C:", MB_OK );
   }

   status =  _getdcwd( 4, szPath, MAX_PATH );  // drive 4 == D:
   if( status != NULL )
   {
      MessageBox( hWnd, szPath, "Current working directory on D:", MB_OK );
   }

...
```

Additional reference words: 1.10 1.20
KBCategory: kbprg kbprb
KBSubcategory: W32s

PRB: Access Denied When Opening a Named Pipe from a Service

PSS ID Number: Q126645
Authored 27-Feb-1995 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.5 and 3.51

SYMPTOMS

If a service running in the Local System account attempts to open a named pipe on a computer running Windows NT version 3.5, the operation may fail with an Access Denied error (error 5). This can happen even if the pipe was created with a NULL DACL.

NOTE: For more information about placing a NULL DACL on a named pipe, please see the following article in the Microsoft Knowledge Base:

```
ARTICLE-ID: Q102798
TITLE     : Security Attributes on Named Pipes
```

CAUSE

In Windows NT version 3.1, a process running in the Local System account could connect to a resource using a Null Session. For security reasons, use of the Null Session is restricted by default on Windows NT version 3.5.

RESOLUTION

You can allow access to a named pipe using the Null Session by adding the pipe name to the following registry entry on the machine that creates the named pipe:

```
\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\LanmanServer\
   Parameters\NullSessionPipes
```

The pipe name added to this entry is the name after the last backslash in the string used to open the pipe. For example, if you use the following string to open the pipe:

```
\\hardknox\pipe\mypipe
```

you would add mypipe to the NullSessionPipes entry on the computer named hardknox.

You must either reboot or restart (stop and then start) the Server service for changes in this entry to take effect. Also, the named pipe will still need to have a NULL DACL.

In Windows NT 3.51, by customer request, it is no longer necessary to

reboot. Once a named pipe is added to the key listed above, null-session connections to that pipe will immediately be accessible.

This new functionality allows programs to permit null session access to named pipes that do not have names known prior to booting the system.

MORE INFORMATION

Usually, when a session is established between a computer supplying a resource (server) and a computer that wants to use the resource (client), the client is identified and credentials are verified. When a Null Session is used, there is no validation of the client; everyone is allowed access.

If you allow a pipe to be used by a Null Session, you should either:

- Verify that the data supplied by the pipe is truly public.

  -or-

- Use an alternative method for verifying clients.

REFERENCES

The "Windows NT Registry Entries" help file in the Windows NT version 3.5 Resource Kit.

Additional reference words: 3.50
KBCategory: kbprg kbprb
KBSubcategory: BseSecurity

PRB: AccessCheck() Returns ERROR_INVALID_SECURITY_DESCR

PSS ID Number: Q115946
Authored 07-Jun-1994 Last modified 28-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SYMPTOMS

In certain cases, the AccessCheck() API fails and GetLastError() returns the message "ERROR_INVALID_SECURITY_DESCR". This error message indicates that the security descriptor passed to AccessCheck() was in an invalid format.

CAUSE

This is expected behavior for the AccessCheck() function. AccessCheck() was designed for use by programs that create and maintain their own security descriptors. These security descriptors would always have the owner, DACL, and group information.

RESOLUTION

If the security descriptor is indeed valid, you can eliminate the error by ensuring that the security descriptor has been opened for access to the following types of security information:

```
OWNER_SECURITY_INFORMATION
GROUP_SECURITY_INFORMATION
DACL_SECURITY_INFORMATION
```

You can double check the validity of the security descriptor by calling the IsValidSecurityDescriptor() API.

Additional reference words: 3.10 3.50
KBCategory: kbprg kbprb
KBSubcategory: BseSecurity

PRB: After CreateService() with UNC Name, Service Start Fails

PSS ID Number: Q127862
Authored 19-Mar-1995 Last modified 04-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 3.5

SYMPTOMS

When giving a Universal Naming Convention (UNC) name to CreateService(), the call succeeds, but service start fails with ERROR_ACCESSS_DENIED (5). The access denied message is given, even though the service runs under an account that has logon as service rights and has full access to the service image path.

CAUSE

The service control manager calls CreateProcess() to start the service and the service controller runs in the LocalSystem security context. When you call CreateProcess() with a UNC name from a process running in the LocalSystem context, you get ERROR_ACCESS_DENIED. This is because LocalSystem has restricted (less than guest) access to remote machines. A null session is set up for LocalSystem remote access, which has reduced rights (less in Windows NT version 3.5 than in Windows NT version 3.1).

RESOLUTION

There are two possible solutions:

- When specifying the fully qualified path to the service binary file, do not use a UNC name. It may be desirable to copy the service binary file to the local machine. This has the added benefit that the service will no longer be dependent on network operations.

  -or-

- If the service binary is on \\MACHINEA\SHARENAME, add SHARENAME to

  HKEY_LOCAL_MACHINE\SYSTEM\
  CurrentControlSet\
  Services\
  LanmanServer\
  Parameters\
  NullSessionShares

  on MACHINEA. This will let requests to access this share from null sessions succeed.

  WARNING: This will allow everyone access to the share. If you want to

maintain security for the share, create an account with the access required.

Additional reference words: 3.50
KBCategory: kbprg kbprb
KBSubcategory: BseService

PRB: Applications Cannot Change the Desktop Bitmap

PSS ID Number: Q74366
Authored 16-Jul-1991 Last modified 13-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SYMPTOMS

The desktop bitmap is not updated when an application updates the Wallpaper entry of the [Desktop] section of WIN.INI and then sends a WM_WININICHANGE message to the desktop window.

RESOLUTION

By design, there is no supported method for an application to dynamically change the desktop bitmap under Windows 3.0 and 3.1.

MORE INFORMATION

Please note that an application could accidentally (or maliciously) reference a desktop bitmap in a format that would GP fault the system. For additional information, please see the following article(s) in the Microsoft Knowledge Base:

```
ARTICLE-ID: Q69292
TITLE     : PRB: Video Driver GP Faults When Handling Large Bitmaps
```

Because the entry in WIN.INI has changed, this means that Windows will GP fault every time the user tries to start it in the future, making Windows no longer available.

In Windows 3.1, the application can call

```
SystemParametersInfo(SPI_SETDESKWALLPAPER,....)
```

which has safety checks built in.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95 WM_WININICHANGE
KBCategory: kbprg kbprb
KBSubCategory: UsrIni

PRB: Area Around Text and Remainder of Window Different

PSS ID Number: Q22242
Authored 17-Dec-1987 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SYMPTOMS

When text is painted into a window, an area around the text is a different color than the remainder of the window.

CAUSE

The area around the text is painted with a solid color while the remainder of the window is painted using a dithered color.

RESOLUTION

To make the area around the text and the remainder of the window have the same color, perform one of the following two steps:

- Use a solid color for the window background, and use the same color for the text background. To ensure that a color is a solid color, use the GetNearestColor function. This function returns the nearest solid color available to represent the specified color.

  -or-

- Call the SetBkMode function to specify TRANSPARENT mode for the text. Doing so prevents Windows from painting the area behind the text. The window background color shows through instead.

MORE INFORMATION

By default, when an application paints text into its window, Windows fills the area around the character with the current background color. Windows always uses a solid color for this purpose.

When an application registers a window class, it specifies a handle to a brush that Windows uses to paint the window background. On some output devices, brushes can create dithered colors. On one of these devices, the area behind painted text might have a different color than the remainder of the window.

The following code specifies the window background color:

```
#define ELANGREEN  0x003FFF00
pTemplateClass->hbrBackground = CreateSolidBrush((DWORD)ELANGREEN);
```

The following code specifies the color used to paint around text and draws some text into a device context:

```
#define SZ -1
SetBkColor(hDC, (DWORD)ELANGREEN);
DrawText(hDC, (LPSTR)"text", SZ, (LPRECT)&Rect, DT_BOTTOM);
```

The color used to paint the area around the text has a yellow cast, which gives it a slightly different appearance than the window background color.

A brush may be able to represent a wider color range than the solid colors because a brush covers an area while a solid color may be used to paint nominal-width lines (for example, lines that are one device unit wide) that must be the same color at all locations and angles. Therefore, the device-driver writer has the option of providing dithered colors for brushes, but has no such freedom when it comes to the solid colors for drawing lines.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: GdiDrw

PRB: AttachThreadInput() Resets Keyboard State

PSS ID Number: Q100486
Authored 22-Jun-1993 Last modified 29-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, version 3.1

SYMPTOMS

Start with a program that calls AttachThreadInput() to a thread in another process. Call GetKeyboardState() to get the current key. Call SetKeyboardState() to set the keystate. This call returns TRUE, indicating success, but the keystate is not successfully set.

If the thread is in the same process, calling SetKeyboardState() works as expected.

CAUSE

When attaching to another thread, a temporary message queue is created. This queue contains a copy of the keystate information from the queue to which you are attaching. When the keystate is set, the temporary queue keystate is updated and the application programming interface (API) succeeds. However, when the detach occurs, the keystate change information is lost and reverts to what it was before the attach.

RESOLUTION

To work around the problem, either:

- Stay attached

  -or-

- Use hooks

STATUS

This problem will not be resolved in the release of Windows NT version 3.1; however, a resolution is being considered for a future release.

Additional reference words: 3.10
KBCategory: kbprg kbprb
KBSubcategory: UsrMisc

PRB: Average & Maximum Char Widths Different for TT Fixed

PSS ID Number: Q92410
Authored 05-Nov-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SYMPTOMS

The tmAveCharWidth and tmMaxCharWidth fields of the TEXTMETRIC structure are not equal for a fixed-pitch TrueType (TT) font such as Courier New.

RESOLUTION

This is by TrueType design. The tmMaxCharWidth denotes the maximum possible ink width of the font rather than maximum cell width of the font.

MORE INFORMATION

TrueType fonts use ABC character spacing. The "A" width is the distance that is added to the current position before placing the TrueType glyph. The "B" width is the width of the black part (ink width) of the TT glyph. The "C" width is the distance from the end of the "B" width to the beginning of the next character. Advanced width (cell width) is equal to A+B+C.

The physical TT fonts that are passed to drivers have just the "B" part of the characters, so all fonts at the level of the driver appear to be proportional width fonts. The tmMaxCharWidth is the least width in which the "B" part of all characters will fit. The tmAveCharWidth is the average advance width (A+B+C). For a fixed-pitch TT font such as Courier New, the A+B+C width is the same for all characters; however, the maximum width as defined above can be different.

tmMaxCharWidth is greater than tmAveCharWidth only if A+C is negative. This is possible for a fixed-pitch font.

Please see section 18.2.4.1 of the Windows 3.1 SDK "Guide to Programming" for more information about ABC character spacing.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: GdiTt

PRB: Building SDK SNMP Samples Results in Unresolved

PSS ID Number: Q129240
Authored 23-Apr-1995 Last modified 24-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5

SYMPTOMS

Building the SNMP samples in the Win32 SDK for Windows NT version 3.5 (TESTDLL.DLL or SNMPUTIL.EXE) using Visual C++ version 2.0 results in the the linker complaining of unresolved externals _mb_cur_max_dll and _pctype_dll.

CAUSE

The application was built to use the C run-time in a DLL. In the Win32 SDK, the import library is CRTDLL.LIB, and in Visual C++, the import library is MSVCRT.LIB. The "___mb_cur_max_dll" and "__pctype_dll" symbols are defined in the CRTDLL.LIB file, but not in the MSVCRT.LIB file.

For more information, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID: Q119503
TITLE : PRB: Unresolved ___mb_cur_max_dll and __pctype_dll

RESOLUTION

Choose Settings from the Project menu. Then select C/C++, and go to the Code Generation category. For the run-time library listed, use Multithreaded using DLL.

Additional reference words: 3.50
KBCategory: kbnetwork kberrmsg kbprb
KBSubcategory: NtwkSnmp

PRB: Byte-Range File Locking Deadlock Condition

PSS ID Number: Q117223
Authored 22-Jun-1994 Last modified 19-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1 and 3.5

SYMPTOMS

It is possible for a thread in a multithreaded Win32-based application to block while doing a LockFile() or LockFileEx() API call, even when the area of memory the thread has requested is not locked by another thread.

NOTE: If performance-monitoring tools (such as PERFMON) are used to examine the status of existing threads and the Thread State indicates that the thread is waiting and the Thread Wait Reason shows that it is the executive that the thread is waiting on, this is probably not an indication that a deadlock has occurred, because threads are often in this state for other reasons. Also, if the Thread State indicates that the thread is not waiting, then a deadlock has probably not occurred.

CAUSE

There is a small window of time during which a multithreaded application is vulnerable to this condition. Specifically, if one thread (call it Thread1) is in the process of unlocking a currently locked byte range within a file while a second thread (Thread2) is in the process of obtaining a lock on that same byte range using the same file handle and without specifying the flag LOCKFILE_FAIL_IMMEDIATELY, Thread1 can block, waiting for the region to become available. Ordinarily, when unlocking takes place, blocked threads are released; but in this critical window of time, it is possible for Thread2 to unlock the byte range without Thread1 being released. Thus, Thread1 never resumes operation despite the fact that there is no apparent fault in the logic of the program.

RESOLUTION

The deadlock condition described above can only come about if multiple threads are concurrently doing synchronous I/O using the same file handle.

To avoid the problem, you have three options:

- Each thread can obtain its own handle to the file either through the use of the DuplicateHandle() API or through multiple calls to the CreateFile() API.

  -or-

- The threads can perform asynchronous I/O. This also requires the

application developer to provide some form of explicit synchronization to coordinate access to the file by the threads.

-or-

- The threads can specify LOCKFILE_FAIL_IMMEDIATELY and then loop until a retry succeeds if the lock request fails. This option is the least desirable because it incurs significant CPU use overhead.

REFERENCES

For more information about threads, files, and file handles, see the following sections in the "Win32 SDK Programmer's Reference," volume 2, part 3, "System Services":

- Chapter 43.1.6, "Synchronizing Execution of Multiple Threads"

- Chapter 45.2.2, "Reading, Writing, and Locking Files"

- Chapter 48.3, "Kernel Objects"

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubCategory: BseMisc

PRB: C2371 BSTR Redefinition When VFW.H Included in MFC App

PSS ID Number: Q129953
Authored 08-May-1995 Last modified 16-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 3.5
- Microsoft Visual C++, 32-bit Edition, versions 2.0 and 2.1

SYMPTOMS

The following error message occurs when compiling an application with Microsoft Visual C++, version 2.0 or 2.1, which uses the Microsoft Video for Windows header file, VFW.H, and the Microsoft Foundation Classes (MFC):

```
c:\msvc20\include\oleauto.h(214) : error C2371: 'BSTR' : redefinition;
different basic types
```

This error message does not occur when using Microsoft Visual C++ version 1.5 for Windows with the Microsoft Video for Windows 1.1 DK installed while running under Microsoft Windows operating system version 3.1.

RESOLUTION

If the application does not require binary string (BSTR) support you can eliminate this error message by defining "_AFX_NO_BSTR_SUPPORT" before the MFC include files. For example, place the code below at the beginning of the STDAFX.H file:

```
#define _AFX_NO_BSTR_SUPPORT
```

If the application does require BSTR support, then you can eliminate this error message by including the code below before including the VFW.H file:

```
#define OLE2ANSI
```

MORE INFORMATION

This error message occurs by design. MFC versions 3.0 and 3.1 require either that "OLE2ANSI" is defined when including the object linking and embedding (OLE) headers or that "_AFX_NO_BSTR_SUPPORT" is defined. You cannot use both ANSI-BSTRs (which is the default) and Unicode-BSTRs; you must use one or the other.

The AFX.H file defines the BSTR type in order to allow CStrings to support BSTRs. The VFW.H file unconditionally includes the OLE2.H file, which eventually includes the OLEAUTO.H file, which also defines the BSTR type. "_AFX_NO_BSTR_SUPPORT" disables CString support for BSTRs.

Steps to Reproduce Problem

1. Generate a default application with AppWizard.
2. Add the statement "#include vfw.h" to the end of the STDAFX.H file.
3. Compile the application.

The VFW.H file is included with the Microsoft Visual C++, 32-bit Edition, and the Microsoft Win32 SDK for Windows NT.

Additional reference words: 3.10 3.50 2.00 2.10 CString port porting VfWDK
KBCategory: kbmm kbprb kberrmsg
KBSubcategory: MMVIDEO

PRB: Calling LoadMenuIndirect() with Invalid Data Hangs

PSS ID Number: Q131281
Authored 07-Jun-1995 Last modified 10-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SYMPTOMS

Under Windows 95 if you call LoadMenuIndirect() with invalid data, the system hangs (stops responding).

CAUSE

Invalid data anywhere in the array of MENUITEMTEMPLATE structures passed to the LoadMenuIndirect() function may cause this problem. An example of this might be an extra NULL byte after a MENUITEMTEMPLATE structure.

Under Windows NT version 3.5, passing the same invalid data in the MENUITEMTEMPLATE structure causes LoadMenuIndirect() to return NULL, with GetLastError() reporting an ERROR_INVALID_DATA value.

STATUS

This behavior is by design.

Additional reference words: 4.00 freeze lock up Menus
KBCategory: kbprg kbprb
KBSubcategory: UsrMen

PRB: Can't Disable CTRL+ESC and ALT+TAB Under Windows NT

PSS ID Number: Q125614
Authored 31-Jan-1995 Last modified 11-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 3.5

SYMPTOMS

CTRL+ESC and ALT+TAB task switching cannot be disabled by an application running under Windows NT.

Capturing WM_SYSCOMMAND messages and not sending them on for processing by DefWindowProc() allowed task switching to be disabled in Windows version 3.1, but it doesn't work under Windows NT.

CAUSE

A primary reason for this change is to avoid dependence on an application for processing of these key combinations. This way a hung application can be switched away from by using either CTRL+ESC or ALT+TAB.

RESOLUTION

CTRL+ESC may disabled on a system-wide basis by replacing the NT Task Manager. This is not recommended.

ALT+TAB and ALT+ESC may be disabled while a particular application is running if that application registers hotkeys for those combinations with Register HotKey().

STATUS

This behavior is by design.

REFERENCES

The first reference below describes the steps that must be taken to replace TASKMAN.EXE. The two additional references provide more information on the Windows NT Task Manager and its relationship to the Program Manager.

ARTICLE-ID:Q89373
TITLE :Replacing the Windows NT Task Manager

ARTICLE-ID:Q100328
TITLE :Replacing the Shell (Program Manager)

ARTICLE-ID:Q101659
TITLE :How Windows NT, 16-Bit Windows 3.1 Task Managers Differ

Additional reference words: 3.50
KBCategory: kbusage kbprb
KBSubcategory: UsrMisc

PRB: Can't Increase Process Priority

PSS ID Number: Q110853
Authored 31-Jan-1994 Last modified 02-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SYMPTOMS

An attempt to increase process priority to the real-time priority REALTIME_PRIORITY_CLASS may not set the PriorityClass to the expected level.

CAUSE

Only accounts with the "Increase Scheduling Priority" permission can increase the priority to real-time. Only Administrators and "Power Users" have this permission by default.

RESOLUTION

Either run the program from an Administrator or Power User account, or grant the Increase Scheduling Priority permission to any user group that must run the program.

STATUS

This behavior is by design.

MORE INFORMATION

The Increase Scheduling Priority permission can be granted to a user or group through the User Manager. To do this:

1. Open the Administrative Tools group in Program Manager.
2. Run the User Manager application.
3. Choose User Rights from the Policies menu.
4. Select the Show Advanced User Rights check box.
5. Select Increase Scheduling Priority from the drop-down list box.
6. Choose the Add button to add users or groups to the list of entities that have this right.

Note that the call to SetPriorityClass() may return success even though the priority was not set to REALTIME_PRIORITY_CLASS, because if you don't have the Increase Scheduling Priority permission, a request for REALTIME_PRIORITY_CLASS is interpreted as a request for the highest

priority class allowed in the current account. If it is important to know the actual priority class that was set, use the GetPriorityClass() function.

Additional reference words: 3.10 3.50
KBCategory: kbprg kbprb
KBSubcategory: BseProcThrd

PRB: Can't Remove Minimize or Maximize Button from Caption

PSS ID Number: Q130760
Authored 25-May-1995 Last modified 22-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SYMPTOMS

Under Windows 95, applications cannot create an overlapped or a popup window that contains only the Minimize or the Maximize button.

CAUSE

This is by design. Windows can have both buttons or none depending on the styles specified while creating the window, but specifying just one style (either the WS_MAXIMZIEBOX or WS_MINIMIZEBOX) creates both buttons on the caption, with the other one disabled.

STATUS

This behavior is by design.

MORE INFORMATION

Under Windows version 3.1 or Window NT, applications could remove either the Maximize or Minimize buttons on the caption bar. (This was usually done when the appliction removed a corresponding menu item from the system menu of a window.)

Applications running under Windows 95 that try to remove one of the buttons (not both), will not succeed. The system displays both buttons and disable the one whose style was not specified during creation. This is by design, and there is no way to work around it, unless the application draws its own caption bar.

Applications that removed the maximize or minimize menu items under Windows version 3.1, should just gray them out (disable them) under Windows 95 to maintain a uniform user interface.

Under Windows 95, applications can create a window (overlapped or popup) with just the Close button (the X button) by creating the window without specifying the WS_MAXIMIZEBOX or WS_MINIMIZEBOX styles. Calling SetWindowLong(GWL_STYLE) to change or remove the minimize of the maxmize buttons dynamically still displays both buttons with one of them disabled.

Additional reference words: 4.00 user
KBCategory: kbprg kbui kbprb

KBSubcategory: UsrWndw

PRB: Can't Use TAB to Move from Standard Controls to Custom

PSS ID Number: Q131283
Authored 07-Jun-1995 Last modified 10-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SYMPTOMS

As you attempt to customize the new Explorer-style File Open or Save As common dialog, pressing the TAB key fails to move the focus from the standard dialog's controls to the new controls specified in the custom template.

CAUSE

The custom dialog box template failed to specify the DS_CONTROL style.

RESOLUTION

When creating a custom dialog box template to be used with the new FileOpen or Save As common dialog, you must specify the DS_CONTROL style.

STATUS

This behavior is by design.

MORE INFORMATION

With the new way of customizing the File Open or Save As common dialog in Windows 95, you can customize applications to provide a custom dialog box template that adds controls to the default Open or Save As dialog box. This custom dialog box is created as a child of the default dialog box and thus requires a WS_CHILD style.

DS_CONTROL is a new style provided in Windows 95. It makes a modal dialog created as a child of another dialog interact well with its parent dialog. This style ensures that the user can move between the controls of the parent dialog and the child dialog - or in this case between the standard Open or Save As common dialog and the custom dialog provided by the application.

NOTE: This article does not apply to Windows NT. This information applies only to the new way of customizing the Explorer-style File Open or Save As common dialog in Windows 95.

Additional reference words: 4.00
KBCategory: kbprg kbprb
KBSubcategory: UsrCmnDlg

PRB: Cannot Alter Messages with WH_KEYBOARD Hook

PSS ID Number: Q33690
Authored 29-Jul-1988 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SYMPTOMS

After creating a program that uses the WH_KEYBOARD hook function to intercept a user's keystrokes and changing the wParam value before passing the hook code on to DefHookProc(), whichever application currently has the focus still receives the original character typed in rather than the translated character.

CAUSE

Keyboard messages cannot be altered with the WH_KEYBOARD hook. All that can be done is to "swallow" the message (return TRUE) or have the message passed on (FALSE). In a keyboard hook function, when you return DefHookProc(), you are passing the event to the next hook procedure in the potential hook chain, and giving it a chance to look at the event to decide whether or not to discard it. You are not passing the message to the system as if you had called DefWindowProc() from a Window procedure.

RESOLUTION

NOTE: In the discussion below, ignore the references to the WH_CALLWNDPROC hook for Win32-based applications. Win32 does not allow an application to change the message in a CallWndProc, as 16-bit Windows does.

To change the value of wParam (and hence the character message that is received by the window with the focus), you must install the WM_GETMESSAGE and WH_CALLWNDPROC hooks. The WH_GETMESSAGE hook traps all messages retrieved via GetMessage() or PeekMessage(). This is the way actual keyboard events are received: the message is placed in the queue by Windows and the application retrieves it via GetMessage() or PeekMessage(). However, because applications can send keyboard messages with SendMessage(), it is best to also install the WH_CALLWNDPROC hook. This hook traps messages sent to a window via SendMessage().

These hooks pass you the address of the message structure so you can change it. In Windows 3.0, when you return DefHookProc() within a WH_GETMESSAGE or WH_CALLWNDPROC hook procedure, you are passing the address of the (potentially altered) contents of the message structure on to the next hook function in the chain. In Windows 3.1, you should use the CallNextHookEx() function to pass the hook information to the next hook function. If you

alter the wParam before passing on the message, this will change the character message eventually received by the application with the focus.

NOTE: For Windows 3.0, keep in mind that the hook callback procedure must be placed in a DLL with fixed code so that it will be below the EMS line and thus will always be present. If the hook callback procedure is not in a fixed code segment, it could be banked out when it is called, and this would crash the system. System-wide hooks in Windows 3.1, however, must still reside in a DLL.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: UsrHks

PRB: CBS_SIMPLE ComboBox Repainting Problem

PSS ID Number: Q128110
Authored 27-Mar-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK version 3.5
- Microsoft Win32s version 1.2

SYMPTOMS

When a CBS_SIMPLE combo box has a WS_CLIPCHILDREN parent, the area below the edit control and left to the list box is not repainted correctly. This problem exists for 16-bit as well as 32-bit applications.

RESOLUTION

To work around this problem, subclass the combo box, calculate the blank area, and then repaint to the desired color.

The following ComboBox subclass procedure is written for a 16-bit application, but you can use the same idea in 32-bit applications.

Sample Code

```
LRESULT CALLBACK NewComboProc(
      HWND hWnd,
      UINT uMessage,
      WPARAM uParam,
      LPARAM lParam)
{
HDC myDC;
HPEN   hPen, hOldPen;
HBRUSH hBrush;
HBRUSH hOldBrush;
COLORREF myColor=RGB(255,255,255); //It can be any color. Here
               //the area is painted white.

HWND hEdit, hList;
RECT comboRect, editRect, listRect;
char   *wndClassName="Edit";

if (uMessage == WM_PAINT)
  {
  CallWindowProc(lpfnOldComboProc, hWnd, uMessage, uParam,
                 lParam);
  myDC = GetDC(hWnd);
  hBrush = CreateSolidBrush(myColor);
  hPen   = CreatePen (PS_SOLID, 1, myColor);
```

```
  hOldBrush = SelectObject(myDC, hBrush) ;
  hOldPen   = SelectObject(myDC, hPen);

  //This code obtains the handle to the edit control of the
  //combobox.

  hEdit = GetWindow(hWnd, GW_CHILD);
  GetClassName (hEdit, wndClassName, 10);
  if (!lstrcmp (wndClassName, "Edit"))

    hList=GetWindow(hEdit, GW_HWNDNEXT);

  else
    {
    hList=hEdit;
    hEdit=GetWindow(hList, GW_HWNDNEXT);
    }

  //The dimensions of the Edit Control, ListBox control and
  //the Combobox are  calculated and then used
  //as the base dimensions for the Rectangle() routine.

  GetClientRect (hWnd, &comboRect);
  GetClientRect (hEdit, &editRect);
  GetClientRect (hList, &listRect);
  Rectangle (myDC,
             comboRect.left,
             editRect.bottom,
             comboRect.right-listRect.right,
             comboRect.bottom);
  //Also paint the gap, if any exists, between the bottom
  //of the listbox and the bottom of the ComboBox rectangle.
  Rectangle (myDC,
             comboRect.right-listRect.right,
        editRect.bottom +
             listRect.bottom,
        comboRect.right,
        comboRect.bottom);

  DeleteObject(SelectObject(myDC, hOldBrush)) ;
  DeleteObject(SelectObject(myDC, hOldPen)) ;
  ReleaseDC(hWnd, myDC);
  return TRUE;
  }

return CallWindowProc(lpfnOldComboProc, hWnd, uMessage, uParam,
 lParam);
}
```

STATUS

This behavior is by design.

MORE INFORMAITON

Steps to Reproduce Behavior

To reproduce this behavior, use AppStudio to create a dialog with the WS_CLIPCHILDREN style, put a CBS_SIMPLE combobox in the dialog, and click the test button so you can test the dialog. Then move something on top of the dialog, and move the object on top of the combobox away. You can then see that area to the left of the listbox is not repainted correctly.

Additional reference words: 3.10 3.50 1.20
KBCategory: kbprg kbcode kbprb
KBSubcategory: UsrCtl

PRB: CBT_CREATEWND Struct Returns Invalid Class Name

PSS ID Number: Q106079
Authored 31-Oct-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

The CBT_CREATEWND structure contains information passed to a WH_CBT hook callback function before a window is created in the system. The lpszClass field of this CBT_CREATEWND structure may return an invalid class name, particularly for windows created internally by the system.

SYMPTOMS

The code to get to the window class name via the CBT_CREATEWND structure in a CBT callback function may look like the following:

```
   int FAR PASCAL CBTProc (int nCode, WPARAM wParam, LPARAM lParam)
    {
         LPCBT_CREATEWND  CreateWndParam;

           if (nCode >= 0)
           {
                switch (nCode)
                {
                   case HCBT_CREATWND:
                        CreateWndParam = (LPCBT_CREATEWND)lParam;
                        OutputDebugString ("ClassName = ");
                        OutputDebugString (lParam->lpcs->lpszClass);
                        OutputDebugString ("\r\n");
                        break;
                     :
                     :
                }
           }

           return (int)CallNextHookEx (hHook, nCode, wParam, lParam);
    }
```

However, this code may or may not output the correct class name, depending on what the call to CreateWindow() (for the window about to be created) looks like.

RESOLUTION

Windows provides the GetClassName() function to allow applications to

retrieve a window's class name. This function takes the handle to the window as a parameter, as well as a buffer to receive the null-terminated class name string. This function is a more reliable and a more recommended means to obtain window class name information than the CBT callback function's CBT_CREATEWND structure.

MORE INFORMATION

Whenever a window is about to be created, Windows checks to see if a WH_CBT hook is installed anywhere in the system. If it finds one, it calls the CBTProc() callback function with hook code set to HCBT_CREATEWND, and with the lParam parameter containing a long pointer to a CBT_CREATEWND structure.

The CBT_CREATEWND structure is defined in WINDOWS.H as follows

```
typedef struct tagCBT_CREATEWND {
    CREATESTRUCT FAR* lpcs;
    HWND hwndInsertAfter;
} CBT_CREATEWND;
```

with the CREATESTRUCT structure defined in the same file as:

```
typedef struct tagCREATESTRUCT {
   void FAR* lpCreateParams;
      :
   LPCSTR  lpszClass;    // Null-terminated string specifying window
                         // class name
   DWORD  dwExStyle;
} CREATESTRUCT;
```

When Windows internally creates windows (such as predefined controls in a dialog box, for example), it uses atoms for lpszClass instead of the actual string to minimize overhead. This makes it a little less straightforward (sometimes impossible) to get to the actual class name directly from the lpszClass field of the CREATESTRUCT structure.

To cite one particular example, when an application is minimized in Windows, Windows calls CreateWindow() for the icon title, specifying a class name of

```
(LPSTR)MAKEINTATOM (ICONTITLECLASS == 0x8004)
```

where MAKEINTATOM() is defined in WINDOWS.H as:

```
#define MAKEINTATOM (i)    ((LPCSTR) MAKELP (NULL, i))
```

Given this, to get to the actual class name, GetAtomName() must be called in this manner:

```
char szBuf [10];

GetAtomName (LOWORD (lpszClass), szBuf, 10);
OutputDebugString (szBuf);
```

This outputs a class name of #32772 for the IconTitleClass.

For predefined controls, such as "static", "button", and so forth, in a dialog box, the Dialog Manager calls CreateWindow() on each control, similarly using atoms for lpszClass. The Dialog Manager, however, creates these atoms in a local atom table in USER's default data segment (DS), thus making it impossible for other applications to get to these class names.

[Note the difference between local and global atom tables, where global atom tables are stored in a shared DS, and are therefore accessible to all applications, while local atom tables are stored in USER's default DS. DDE uses global atom tables so that Excel, for example, can use GlobalAddAtom() to add a string to the atom table, and another program could use GlobalGetAtomName() to obtain the character string for that atom.]

More information on atoms can be found by searching on the following word in the Microsoft Knowledge Base:

   atoms

More Information can also be found in the Windows version 3.1 online Help file under the heading Function Groups, under the Atom Functions subheading.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbprb kbcode
KBSubcategory: UsrHks

PRB: CFileDialog::DoModal() Does Not Display FileOpen Dialog

PSS ID Number: Q131225
Authored 06-Jun-1995 Last modified 07-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SYMPTOMS

Calling CFileDialog::DoModal() returns without displaying the FileOpen common dialog.

CAUSE

The CFileDialog class will automatically use the new Explorer-style FileOpen common dialog under Windows 95. This can break existing code which customizes these dialogs with custom templates, because the mechanism has changed in Windows 95.

NOTE: This does not apply to Windows NT 3.51, as this version of Windows NT will not display the new Explorer-style dialog.

RESOLUTION

An application that depends on the old behavior of customizing the File Open common dialogs will need to reset the OFN_EXPLORER bit in the Flags member of the OPENFILENAME structure before calling CFileDialog::DoModal.

MORE INFORMATION

The DIRPKR sample in particular, exhibits the symptoms described above, and will need to be modified to display the dialog box correctly in Windows 95. It works as is under Windows NT 3.51.

Sample Code

```
    CMyFileDlg  cfdlg(FALSE, NULL, NULL, OFN_SHOWHELP | OFN_HIDEREADONLY |
                      OFN_OVERWRITEPROMPT | OFN_ENABLETEMPLATE,
                      NULL, m_pMainWnd);

    cfdlg.m_ofn.hInstance      = AfxGetInstanceHandle();
    cfdlg.m_ofn.lpTemplateName = MAKEINTRESOURCE(FILEOPENORD);
    cfdlg.m_ofn.Flags         &= ~OFN_EXPLORER;

    if (IDOK==cfdlg.DoModal())
    {
      :
```

```
    :
}
```

REFERENCES

This information was derived from Visual C++ 2.1 Technical Note 52:
"Writing Windows 95 Applications with MFC 3.1"

Additional reference words: 2.10 4.00
KBCategory: kbprg
KBSubcategory: UsrCmnDlg

PRB: COMM (TTY) Sample Does Not Work on Windows 95

PSS ID Number: Q128787
Authored 10-Apr-1995 Last modified 11-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SYMPTOMS

The Win32 COMM (old TTY) sample that ships with Visual C++ version 2.x and the Windows 95 SDK (pre-release) does not work correctly under Windows 95 M8 builds and later. The problem involves assigning values to the Offset member of the OVERLAPPED structure which is one of the arguments to the WriteFile function call.

The observed behavior is that the COMM sample writes only one byte to the serial port. No other data is transmitted after the first byte.

CAUSE

The documentation for the OVERLAPPED structure explicitly states that the Offset and OffsetHigh members must be set to 0 when reading from or writing to a named pipe or communications device. This was not done in the sample.

RESOLUTION

1. Delete the following line from the WriteCommByte() function in the sample:

   ```
   WRITE_OS( npTTYInfo ).Offset += dwBytesWritten ;
   ```

2. Add the following lines to the CreateTTYInfo() function:

   ```
   WRITE_OS( npTTYInfo ).Offset = 0 ;
   WRITE_OS( npTTYInfo ).OffsetHigh = 0 ;
   ```

STATUS

This is a problem with the sample, not with Windows 95. Windows 95 correctly implements WriteFile() and use of the OVERLAPPED structure.

Additional reference words: 4.00
KBCategory: kbprg kbprb
KBSubcategory: BseComm

PRB: Controls Do Not Receive Expected Messages

PSS ID Number: Q121094
Authored 26-Sep-1994 Last modified 28-Nov-1994

The information in this article applies to:

- Microsoft Win32s, versions 1.1, 1.15, and 1.2

SYMPTOMS

A Win32-based application works as expected under Windows NT, but under Win32s, control messages are not received by the application as expected.

Here are some possible scenarios symptomatic of this problem:

- Sending LB_GETSELITEMS to a superclassed list box does not work.

- The custom control procedure does not receive messages such as CB_ADDSTRING, CB_INSERTSTRING, or CB_SHOWDROPDOWN.

- The wrong control message is received.

CAUSE

This is a Win32s limitation. When a Win32-based application sends a message, Win32s processes the message and passes it to Windows. Windows is responsible for delivering the message. Win32s truncates wParam from a DWORD to a WORD. Most messages do not use the high word of wParam, however, if you do use the high word of wParam, be aware that it will be lost under Win32s.

In the course of processing known messages (messages that are not user-defined), Win32s translates any pointers in lParam. In addition, if a message is new to Win32, the corresponding Windows message is used instead. For example, the Windows WM_CTLCOLOR was replaced in Win32 with WM_CTLCOLORBTN, WM_CTRLCOLORDLG, WM_CTLCOLOREDIT, and so on. Therefore, if a Win32-based application uses WM_CTRLCOLORBTN, Win32s passes WM_CTLCOLOR to Windows with a type of CTLCOLOR_BTN.

Control messages are not unique on Windows version 3.1 as they are on Windows NT. Control messages on Windows have values above WM_USER, however, messages for one control may share the same number as a message of another control. For example, both CB_ADDSTRING and LB_DELETESTRING are defined as WM_USER+3. Therefore, when Win32s receives the WM_USER+3 message, it needs to determine the correct control message. Win32s looks at the window class of the window that will receive the message and maps the message accordingly. If the window class is not a recognized control class, as in the case of a superclassed control, the message is not mapped, which results in unexpected behavior.

RESOLUTION

In order to get the desired behavior under Win32s, make the custom control use a recognized control class (such as "combobox") and subclass the window procedure instead of superclassing. If you need to subclass before the WM_CREATE/WM_NCCREATE messages, use a CBT hook. You will not be able to change the class icon and cursor, but the messages will be handled correctly.

If you need to use a custom control, create and use user-defined messages instead of the control messages.

STATUS

This behavior is by design.

Additional reference words: 1.10 1.15 1.20
KBCategory: kbprg kbprb
KBSubCategory: W32s

PRB: Corruption of the Perflib Registry Values

PSS ID Number: Q128404
Authored 02-Apr-1995 Last modified 11-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, version 3.5

SYMPTOMS

The Performance Monitoring tool can be affected by a corruption problem in the Perflib key. The following symptoms are evidence of this corruption problem:

- The explain text is not displayed.

  -or-

- The objects and counters are not displayed.

  -or-

- The explain text, objects, and counters are not displayed.

CAUSE

When an .INI file, which contains an extra language entry not associated with any specific objects and counters, is passed as a parameter to LODCTR.EXE, the values of 'Last Counter' and 'Last Help' are set to zero. This in turn causes the Performance Monitoring tool to fail.

The following is an example of a problem-causing .INI file:

```
[info]
drivername=SampPerf
symbolfile=sampperf.h

[languages]
009=English
011=OtherLanguage  <-- problem area

[text]
SampObj_009_Name=SampPerf
SampObj_009_Help=A sample performance object.

Count_1_009_Name=Tests/sec
Count_1_009_Help=The number of tests completed.
```

A further complication arises when a user attempts to rectify the situation by executing UNLODCTR.EXE. At this point, because Last Counter and Last

Help are set to zero, UNLODCTR.EXE only resets '\009\HELP' and '\009\COUNTER' to NULL.

RESOLUTION

To resolve this corruption problem, you must restore the Registry to its former state by using one of the following five methods:

- Backup and restore the local copy of the Registry by using Windows NT Backup.

  -or-

- Copy and restore the data located in the %WINNT%\SYSTEM32\CONFIG directory. This can be done only if Windows NT was installed on a FAT partition.

  -or-

- Save and restore the SOFTWARE registry hive.

  -or-

- Save and restore the Registry values First Counter, First Help, Last Counter, and Last Help.

  -or-

- Save and restore the Registry by using REGBACK.EXE and REGREST.EXE. Both programs are available with the Windows NT Resource Kit (RESKIT).

If the user ran UNLODCTR.EXE to attempt to fix the problem, you will also need to restore \009\HELP and \009\COUNTER, which you can do by simply rebooting the computer running Windows NT.

WARNING: The Registry is a vital part of Windows NT; improper modification of its keys and values can cause Windows NT to malfunction.

Additional reference words: 3.50
KBCategory: kbprg kbprb
KBSubcategory: BseMisc

PRB: CreateEllipticRgn() and Ellipse() Shapes Not Identical

PSS ID Number: Q83807
Authored 20-Apr-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SYMPTOMS

When CreateEllipticRgn() is used to create a region in the shape of an ellipse and Ellipse() is called with the same parameters to draw an ellipse on the screen, the calculated region does not match the drawn ellipse identically.

CAUSE

Ellipse() includes the lower-right point of the bounding rectangle in its calculations, while the CreateEllipticRgn function excludes the lower-right point.

RESOLUTION

To draw a filled ellipse on the screen that matches an elliptic region, create the region with CreateEllipticRgn() and call FillRgn() to fill the region with the currently selected brush.

MORE INFORMATION

The region created by the CreateEllipticRgn() is slightly smaller than the elliptical area created by Ellipse(). Unfortunately, decreasing the width and height of the bounding rectangle by 1 pixel does not solve the problem. Although changing the parameters of the Ellipse() API in this way produces a smaller ellipse, the new ellipse does not match the region created with CreateEllipticRgn().

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: GdiDrw

PRB: CreateFile Fails on Win32s w/ UNC Name of Local Machine

PSS ID Number: Q129543
Authored 27-Apr-1995 Last modified 29-Apr-1995

The information in this article applies to:

- Microsoft Win32s version 1.2

SYMPTOMS

CreateFile() fails on Win32s if the file name is a UNC name that refers to the local machine.

CAUSE

This is a limitation of Windows for Workgroups version 3.11. The same problem occurs with 16-bit Windows-based applications using OpenFile().

RESOLUTION

Do not use a UNC name to open a file on the local machine.

STATUS

This behavior is by design.

MORE INFORMATION

There is a similar limitation with Windows version 3.1 and LAN Manager. If you create a network drive and try to open a file on the same network share using a UNC name, it will fail. This also happens with 16-bit Windows-based applications.

Additional reference words: 1.20
KBCategory: kbprg kbprb
KBSubcategory: W32s

PRB: CreateFile() Does Not Handle OEM Chars as Expected

PSS ID Number: Q129544
Authored 27-Apr-1995 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Win32s, versions 1.2 and 1.25a

SYMPTOMS

When CreateFile() is passed a filename string that contains an OEM character, the name of the file created is not as expected. In particular, a file created with CreateFile() cannot be opened with OpenFile() when OpenFile() is passed the same filename string.

For example, suppose that the ANSI filename string contains a lower-case e accent character (0x0e9). The file created by CreateFile() contains an upper-case E and the file created by OpenFile() contains an upper-case E accent character.

CAUSE

CreateFile() is implemented with calls to MS-DOS. MS-DOS converts the given file names to upper-case letters. OpenFile() is implemented with a thunk to the 16-bit Windows OpenFile(). OpenFile() converts the file name to upper-case before calling MS-DOS. The conversion that the 16-bit OpenFile() is doing is different from the conversion performed by MS-DOS for the OEM characters. The result is that different filenames are created for the same string passed to CreateFile() and OpenFile() if the name contains OEM characters.

RESOLUTION

To make the file name created by CreateFile() consistent with the file name created by OpenFile(), call AnsiUpper() on the file name string before calling CreateFile().

STATUS

This behavior is by design. This will not be changed in Win32s now, because it may break existing Win32-based applications.

MORE INFORMATION

The 16-bit OpenFile() will fail to find an existing file if the file contains OEM characters but not an explicit full path. This also occurs with the 32-bit OpenFile(), because it is thunked to the 16-bit OpenFile().

SearchFile() will also fail to find files if the file name contains OEM characters or if the search path (lpszpath != NULL) contains OEM characters.

NOTE: In 16-bit Windows and Win32s 1.2 and earlier, OpenFile() returns the filename in OEM characters. However, the Win32 API documentation states that OpenFile() should return an ANSI string. Starting with the next version of Win32s, OpenFile() will return an ANSI string, as it does under Windows NT.

Additional reference words: 1.20
KBCategory: kbprg kbprb
KBSubcategory: W32s

PRB: CreateProcess() of Windows-Based Application Fails

PSS ID Number: Q127860
Authored 19-Mar-1995 Last modified 21-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 3.5

SYMPTOMS

When you spawn a 16-bit Windows-based application using CreateProcess() where neither lpApplicationName and lpCommandLine are NULL, WOW gives a popup saying:

> Cannot find file (or one of its components). Check to ensure the path and filename are correct and that all required libraries are available.

CAUSE

NTVDM expects the first token in the command line (lpCommandLine) to be the program name, although the Win32 subsystem does not. The current design will not be changed.

RESOLUTION

Make lpApplicationName NULL and put the full command line in lpCommandLine.

STATUS

This behavior is by design.

MORE INFORMATION

The documentation for CreateProcess() states:

> If the process to be created is an MS-DOS-based or Windows-based application, lpCommandLine should be a full command line in which the first element is the application name.

In this case (lpApplicationName is not NULL), lpCommandLine not only should be a full command line, but it must be a full command line.

Additional reference words: 3.50
KBCategory: kbprg kbprb
KBSubcategory: BseProcThrd

PRB: Data Section Names Limited to Eight Characters

PSS ID Number: Q100292
Authored 17-Jun-1993 Last modified 18-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SYMPTOMS

Data sections can be named by using #pragma data_seg. This method is commonly used so that the named data sections can be shared using the SECTIONS statement in the DEF file. However, if the length of the name specified in the pragma exceeds eight characters, then the section is not properly shared.

CAUSE

The linker does not support sections with longer names. The longer names require use of the COFF strings table, so the rewrite is not trivial.

MORE INFORMATION

Note that in addition, the first character of a section name must be a period. Therefore, the section name, as specified in both the pragma and the DEF file, can be a maximum of a period followed by seven characters.

For more information on the shared named-data section, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID: Q89817
TITLE : How to Specify Shared and Nonshared Data in a DLL

Additional reference words: 3.10 3.50
KBCategory: kbtool kbprb
KBSubcategory: TlsMisc

PRB: DDEML Fails to Call TranslateMessage() in its Modal Loop

PSS ID Number: Q102576
Authored 04-Aug-1993 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

During a synchronous transaction, DDEML causes the client to enter a modal loop until the transaction is processed. While DDEML dispatches messages appropriately, it fails to call TranslateMessage() while inside this modal loop. This problem does not apply to asynchronous transactions, where no such modal loop is entered.

SYMPTOMS

A common symptom of this problem is seen as the client processes user input while inside DDEML's modal loop in a synchronous transaction. WM_KEYDOWN and WM_KEYUP messages are received, with no corresponding WM_CHAR message for the typed character.

CAUSE

No WM_CHAR message is received because the WM_KEYDOWN message is never translated. For this to take place, a call to TranslateMessage() must be made inside the modal loop.

RESOLUTION

This limitation is by design. DDEML applications can work around this limitation by installing a WH_MSGFILTER hook, watching out for code == MSGF_DDEMGR.

The WH_MSGFILTER hook allows an application to filter messages while

the system enters a modal loop, such as when a modal dialog box (code

similarly, when DDEML enters a modal loop in a synchronous transaction (code == MSGF_DDEMGR).

The Windows 3.1 Software Development Kit (SDK) DDEML\CLIENT sample demonstrates how to do this in DDEML.C's MyMsgFilterProc() function:

```
/**************************************************************************
 *
 *  FUNCTION: MyMsgFilterProc
```

```
 *
 *  PURPOSE:  This filter proc gets called for each message we handle.
 *            This allows our application to properly dispatch messages
 *            that we might not otherwise see because of DDEMLs modal
 *            loop that is used while processing synchronous transactions.
 *
*************************************************************************/

   DWORD FAR PASCAL MyMsgFilterProc( int nCode, WORD wParam,

                DWORD lParam)
   {
       wParam; // not used

    #define lpmsg ((LPMSG)lParam)

    if (nCode == MSGF_DDEMGR) {

           /* If a keyboard message is for the MDI, let the MDI client
            * take care of it. Otherwise, check to see if it's a normal
            * accelerator key. Otherwise, just handle the message as usual.
            */

           if ( !TranslateMDISysAccel (hwndMDIClient, lpmsg) &&
                !TranslateAccelerator (hwndFrame, hAccel, lpmsg)){
               TranslateMessage (lpmsg);
               DispatchMessage (lpmsg);
           }
           return(1);
       }
       return(0);
   #undef lpmsg
   }
```

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbprb kbcode
KBSubcategory: UsrDde

PRB: DDEML with Excel Error: Remote Data Not Accessible

PSS ID Number: Q95982
Authored 03-Mar-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SYMPTOMS

When data is linked between a Microsoft Excel spreadsheet and a DDEML server application, with both applications open, the following error message may appear:

Remote data not accessible; start application <SERVER FILENAME>.EXE?

CAUSE

Excel broadcasts an initiate to all windows, for every service/topic name pair it finds, in an attempt to initiate a conversation with the server application. (This can be easily verified by running DDESPY and watching Excel broadcast its initiates.) If Excel can't get a response, or gets a NACK (negative ACK), Excel attempts to EXEC() a new instance of the server application.

RESOLUTION

A DDEML server application should return TRUE, in response to the XTYP_CONNECT transaction it receives, for every service/topic name pair it supports. Refer to page 518 of the Microsoft Windows Software Development Kit (SDK) version 3.1 "Programmer's Reference, Volume 3: Messages, Structures, and Functions" for more information on the XTYP_CONNECT transaction.

MORE INFORMATION

The DDEML server application responds to Excel's initiate by sending the XTYP_CONNECT transaction to the DDE callback function of each server application, passing the service and topic names to the server.

If the server application fails to return TRUE to the service/topic name it supports, Excel concludes that it is trying to initiate a DDE link to an application that is not available, and brings up the message box above, thus giving the user an option to start the application.

Additional reference words: 3.50 3.51 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: UsrDde

PRB: DDESpy Track Conversations Strings Window Empty

PSS ID Number: Q88197
Authored 19-Aug-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SYMPTOMS

In the DDESpy application, provided with version 3.1 of the Microsoft Windows Software Development Kit (SDK), DDE messages appear in the main window, while the Track Conversation Strings window remains empty.

CAUSE

Either the DDESpy application was started after the client and server applications, or neither application uses functions in the Dynamic Data Exchange Management Library (DDEML) to conduct DDE.

RESOLUTION

Start the DDESpy application before starting the client and server applications. Make sure that the client or the server (or both) uses the DDEML.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool kbprg kbprb
KBSubcategory: TlsSpy

PRB: Debugging an Application Driven by MS-TEST

PSS ID Number: Q100957
Authored 01-Jul-1993 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SYMPTOMS

When an application driven by MS-TEST is being debugged by WinDbg or NTSD (for example, after an exception has occurred), both the application and the debugger hang.

CAUSE

The debugger is hooked and ends up hanging.

RESOLUTION

It is not possible to use NTSD or WinDbg to debug an application that is driven by MS-TEST. Use Dr. Watson (drwtsn32) instead. Note that you must turn off Visual Notification.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool kbprb
KBSubcategory: TlsWindbg

PRB: Debugging the Open Common Dialog Box in WinDbg

PSS ID Number: Q99952
Authored 10-Jun-1993 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SYMPTOMS

When debugging an application that uses the Open common dialog box (created by the GetOpenFileName() function) in WinDbg, the program stops and the following information is displayed in the Command window:

```
Thread Terminate: Process=0, Thread=2, Exit Code=1
```

CAUSE

The Open common dialog box causes a thread to be created. At this point in the debugging, that thread has terminated. By default, WinDbg halts whenever a thread terminates.

RESOLUTION

Execute the go command (type "g" at the command prompt). Execution will continue.

MORE INFORMATION

To prevent WinDbg from halting when a thread is terminated, select Debug from the Options menu and check "Go on thread terminate."

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool kbprb
KBSubcategory: TlsWindbg

PRB: Description for Event ID Could Not Be Found

PSS ID Number: Q129003
Authored 17-Apr-1995 Last modified 18-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5
- Microsoft Visual C++, 32-bit Edition, version 2.0

SYMPTOMS

After building and executing the Win32 LOGGING sample that accompanies Visual C++ version 2.0 and the Microsoft Developer's Network Library CD, you find that the descriptions for the logged events contain the following:

> The Description for Event ID (#) in Source (application name) could not be found. It contains the following string(s):

CAUSE

The resource file containing the descritions that correspond to the Event IDs are not bound to the MESSAGES.DLL.

RESOLUTION

Add the MESSAGES.RC file to the Visual C++ version 2.0 project:

1. Run MAKEMC.BAT first. MAKEMC.BAT creates the MSG00001.BIN, MESSAGES.RC, and MESSAGES.H files.

2. Open Visual C++ version 2.0 and add MESSAGES.RC to the MESSAGES project (MESSAGES.MAK).

3. Rebuild both the MESSAGES and the LOG projects.

NOTE: If you ran the sample a couple of times from different places, you may need to delete the LOG registry key if it contains an incorrect location for the MESSAGES.DLL file.

Additional reference words: 3.50 3.51 Event Logging Log MSDN
KBCategory: kbprg kbprb
KBSubcategory: BseMisc

PRB: Device and TrueType Fonts Rotate Inconsistently

PSS ID Number: Q82932
Authored 01-Apr-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SYMPTOMS

In an application for the Microsoft Windows graphical environment that uses a mapping mode other than MM_TEXT and a text alignment other than the default (TA_LEFT, TA_TOP), device fonts and TrueType fonts rotate in the opposite directions. Device fonts may exhibit other unusual behaviors. Differences in rotated text may appear on the screen or in printed output.

RESOLUTION

To create a font that rotates based on the coordinate system in which the font is used, the application must set the CLIP_LH_ANGLES (0x10) bit in the lfClipPrecision field of the LOGFONT data structure. This technique is backward-compatible with Windows 3.0 because the CLIP_LH_ANGLES bit is ignored in version 3.0.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: GdiTt

PRB: Dialog Box and Parent Window Disabled

PSS ID Number: Q11337
Authored 01-Dec-1987 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SYMPTOMS

When an application uses one of the DialogBox family of functions to create a modal dialog box, both the parent window and the dialog box are disabled (unable to accept keyboard or mouse input).

CAUSE

In the application's resource file, the dialog box resource has the WS_CHILD style.

RESOLUTION

To avoid this problem, use the WS_POPUP style instead of the WS_CHILD style.

MORE INFORMATION

When an application creates a modal dialog box using one of the DialogBox family of functions, Windows disables the dialog box's parent window. If the parent window has any child windows, the child windows are also disabled.

An application can use the WS_CHILD style for dialog boxes created by one of the CreateDialog family of functions. However, problems and inconsistencies arise if the application uses the IsDialogMessage function to process dialog box input for either the parent or the child.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
CreateDialogIndirect CreateDialogIndirectParam CreateDialogParam
DialogBoxIndirect DialogBoxIndirectParam DialogBoxParam
KBCategory: kbprg kbprb
KBSubcategory: UsrDlgs

PRB: Dialog Editor Does Not Modify RC File Dialog Box

PSS ID Number: Q32019
Authored 22-Jun-1988 Last modified 22-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.1 and 3.5

SYMPTOMS

If the Windows Dialog Editor (DIALOG.EXE) is used to edit a dialog box, when the associated application is built, the previous version of the dialog box is used.

CAUSE

The dialog resource used by the program is stored in the application's resource file (which has the extension RC). The Dialog Editor stores the updated dialog box resource in a file with the .DLG extension.

RESOLUTION

Modify the resource file to use the RCINCLUDE statement to include the updated dialog box resource, as follows:

```
RCINCLUDE DIALOG.DLG
```

MORE INFORMATION

Suppose an application is written that uses a dialog box. The developer creates a dialog box template manually in the application's resource file. After building the application, the developer decides to modify the dialog box using the Dialog Editor.

When the Dialog Editor edits an existing dialog box, it reads the application's compiled resources stored in a compiled resource file (with the extension .RES). However, when the developer saves any changes, the Dialog Editor creates a file with the .DLG extension.

When the application is built for a second time, the Resource Compiler uses the original definition for the dialog box stored in the resource file. The resolution provided above avoids this problem because the Resource Compiler always includes the latest version of the dialog box template.

For more information about this topic, please see the following article in

the Microsoft Knowledge Base:

   ARTICLE-ID: Q40958
   TITLE   : PRB: DIALOG.EXE Reads Compiled .RES Files, Not .DLG Files

Additional reference words: 3.00 3.10 3.50
KBCategory: kbtool kbprg kbprb
KBSubcategory: TlsDlg

PRB: Dialog Editor Does Not Retain Unsupported Styles

PSS ID Number: Q74264
Authored 15-Jul-1991 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SYMPTOMS

The Windows Dialog Editor does not retain control/dialog styles that it does not support.

RESOLUTION

The styles in question must be manually added to the dialog script before using the resource compiler to compile the resource script.

MORE INFORMATION

Any style that cannot be explicitly set in the Dialog Editor Styles dialog box will not be present in the script file generated by the the Dialog Editor. For example, the edit control style, ES_OEMCONVERT is not available in the Dialog Editor Styles dialog box. If this style is manually added to the dialog script, compiled, and then loaded into the Dialog Editor, the ES_OEMCONVERT style will be removed.

Any styles that are not supported by the Dialog Editor must be manually added to the dialog script. To modify the dialog script, use a text editor to combine the desired styles using the bitwise OR operator("|"). This modified script should then be included into the application's resource script. If the application's .RES file is later loaded into the Dialog Editor, the unsupported styles will be removed and will need to be added again as described above.

The following is a list of styles that are known not to be supported by the Dialog Editor:

Style
CBS_OEMCONVERT
ES_OEMCONVERT
SBS_BOTTOMALIGN
SBS_LEFTALIGN
SBS_RIGHTALIGN
SBS_SIZEBOX
SBS_SIZEBOXBOTTOMRIGHTALIGN

SBS_SIZEBOXTOPLEFTALIGN
SBS_TOPALIGN
SS_LEFTNOWORDWRAP
SS_NOPREFIX
SS_USERITEM

In addition to the above control styles, any menu definitions added to the dialog script via the MENU resource statement will be deleted by the Dialog Editor.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbtool kbprb
KBSubcategory: TlsDlg

PRB: DIALOG.EXE Reads Compiled .RES Files, Not .DLG Files

PSS ID Number: Q40958
Authored 07-Feb-1989 Last modified 22-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.1 and 3.5

SYMPTOMS

Create a dialog box using DIALOG.EXE, the Dialog Editor. From MS-DOS, change an attribute (for example, position of control) of the dialog box by modifying the .DLG file produced by DIALOG.EXE. Invoke DIALOG.EXE again, the changes that were made are not evident.

CAUSE

The Dialog Editor produces .DLG and .RES files for the dialog box created. When using the Dialog Editor to modify an existing dialog box, the Dialog Editor will look at the .RES file for information about the makeup of the dialog box. The .DLG file is completely ignored at this point. This is why the Dialog Editor does not recognize any .DLG changes.

RESOLUTION

For the modifications to be recognized, the Resource Compiler (RC) must be used with the -r switch to compile the .DLG file. The new .RES file can then be loaded into the Dialog Editor and the changes will be recognized.

Additional reference words: 3.00 3.10 3.50
KBCategory: kbtool kbprg kbprb
KBSubcategory: TlsDlg

PRB: DialogBox() Call w/ Desktop Window Handle Disables Mouse

PSS ID Number: Q129597
Authored 30-Apr-1995 Last modified 01-May-1995

The information in this article applies to:

- Microsoft Win32s, version 1.2

SYMPTOMS

Under Windows NT and Windows 95, you can call DialogBox() using the return from GetDesktopWindow() as the parent window handle. However, doing the same thing under Win32s will disable the mouse. Clicking any mouse button results in a beep, and you must reboot the computer to enable the mouse again.

CAUSE

This is a bug in Windows, not Win32s. The same thing occurs from a 16-bit Windows-based application. Internally, Windows disables the parent window by calling EnableWindow(hwndParent, FALSE). If the handle is the desktop window handle, the desktop window is disabled. This disables the mouse as well, due to the bug in the Windows code.

RESOLUTION

Do not call DialogBox() with the desktop window handle as the parent window handle.

Additional reference words: 1.20 HWND_DESKTOP
KBCategory: kprg kbprb
KBSubcategory: W32s

PRB: Display Problems with Win32s and the S3 Driver

PSS ID Number: Q117153
Authored 21-Jun-1994 Last modified 04-Apr-1995

The information in this article applies to:

- Microsoft Win32s versions 1.1, 1.15, and 1.2

SYMPTOMS

Applications running on Windows 3.1 exhibit no problems when using an S3 driver, with a resolution of 1024x768 and 256 colors. However, the display is corrupted when Win32-based applications, such as FreeCell, run on Win32s in the same configuration.

CAUSE

This is a known problem with the S3 driver.

A general protection (GP) fault occurs when the display driver is performing a bit-block transfer (BitBlt). This happens whether or not Win32s is running. However, when Win32s is not running, Windows recovers from the faults. When a Win32-based application is running, Win32s catches all exceptions and transfers control to the nearest try/except frame. As a result, the BitBlt is interrupted.

RESOLUTION

Certain S3 drivers which exhibit these problems can be made to work with Win32s by making the following edit to your SYSTEM.INI file before running any Win32-based applications:

In the SYSTEM.INI file, you will find an entry in the [display] section

```
aperture-base=100
```

Change this entry to

```
aperture-base=0
```

Restart Windows and the display problems will no longer occur.

If this does not help, obtain the latest S3 drivers. It is reported that S3 driver version 1.3 does not have this problem.

Additional reference words: 1.10 1.20
KBCategory: kbprg kbprb
KBSubcategory: W32s

PRB: DLL Load Fails Under Win32s When Apps Share a DLL

PSS ID Number: Q131224
Authored 06-Jun-1995 Last modified 07-Jun-1995

The information in this article applies to:

- Microsoft Win32s versions 1.2 and 1.25a

SYMPTOMS

LoadLibrary fails under Win32s in the following situation:

1. App1 is executed and loads MYDLL.DLL.

2. App2 is executed and loads MYDLL.DLL as well.

3. App1 is terminated and unloads MYDLL.DLL.

App2 can GP fault when MYDLL.DLL is accessed. In addition, if App1 is restarted and attempts to load MYDLL.DLL, the call to LoadLibrary fails and GetLastError reports ERROR_DLL_INIT_FAILED.

This problem does not occur under Windows NT or Windows 95.

CAUSE

By default, Visual C++ create a DLL that statically links to the C Run-time (CRT). This version of the CRT is not compatible with Win32s. Problems occur when global data for the CRT is initialized, because of the shared address space under Windows.

The static CRT library uses global variables to manage memory allocations. In the scenario above, App2 can fault when allocating memory, if the global variables used to access memory refer to memory that was allocated App1 which has since terminated, because the allocated memory was returned to the heap.

RESOLUTION

Use the /MD (Multithreaded using CRT in a DLL) option when compiling the DLL and add the MSVCRT.LIB import library to the library list. Include the Win32s version of MSVCRT20.DLL which is included in Visual C++ 2.x (it is redistributable) in your project. Then the DLL will use the DLL version of the CRT libraries that is compatible with Win32s and the problem will not occur.

MORE INFORMATION

When a DLL that uses the CRT is loaded into memory, all global variables for the DLL and for the CRT libraries are initialized. Under Windows NT and Windows 95, the application is given its own copy of the global data for the DLL. When other applications use the same DLL, they each receive their own copy of the global variables as well. This eliminates conflicts, because the data is not shared.

Under Win32s, DLLs are loaded into the same shared memory space and all global variables for a DLL are shared. This means that the CRT global data is also shared. The version of MSVCRT20.DLL that targets Win32s was written to take this into account and avoid conflicts.

The Windows NT/Windows 95 version of MSVCRT20.DLL can be found in the MSVC20\REDIST directory of the CD. The Win32s version can be found in the WIN32S\REDIST directory.

Additional reference words: 1.20 2.00
KBCategory: kbprg kbprb
KBSubcategory: W32s

PRB: Dotted Line Displays as Solid Line

PSS ID Number: Q24179
Authored 16-Dec-1987 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows, versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SYMPTOMS

When an application creates a dotted line using a dotted pen and the R2_NOT mode, a solid line is drawn on the screen.

CAUSE

The background mode is OPAQUE. This is the default background mode. In this mode, the line is painted with the background color first, followed by the pen.

RESOLUTION

Use the SetBkMode() function to set the mode to TRANSPARENT. In this mode, only the pen is used; the background is not disturbed.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: GdiPnbr

PRB: Double Quotes Not in Help Files Compiled From Word 6 RTF

PSS ID Number: Q114604
Authored 08-May-1994 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SYMPTOMS

When using the Microsoft Windows Help Compiler (any version) to compile a .RTF file generated by Microsoft Word for Windows version 6.0, any double quotation marks (") in the text may not show up in the compiled help file.

CAUSE

Microsoft Word for Windows 6.0 automatically replaces double quotation marks with "smart quotes". This causes your .RTF file to contain the \ldblquote and \rdblquote tokens rather than the double quotation mark characters. The Help Compiler does not recognize these tokens and therefore ignores them.

RESOLUTION

1. Replace existing \ldblquote and \rdblquote tokens with quotation mark characters by loading your .RTF file into a text editor (such as notepad) and performing a search and replace. That is, search for \ldblquote and replace with the quote character (") (without the parenthesis).

2. Stop Word for Windows from replacing quotation mark characters in the future by unchecking the "Replace Simple Quotes with Smart Quotes" check box under both of the following dialog boxes:

   a. Choose Options from the Tools menu, then choose Auto Format.

   b. Choose Auto Correct from the Tools menu.

Additional reference words: 3.10 3.50 4.00 95 WINHELP HC HC31 HC30 HCP
KBCategory: kbtool kbprg kbprb
KBSubcategory: TlsHlp

PRB: DSKLAYT2 May Create Too Many Files on a Disk Image

PSS ID Number: Q114605
Authored 08-May-1994 Last modified 22-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, version 3.1

SYMPTOMS

The DSKLAYT2 program may create a disk image with more than 224 files on a single disk when many small files are part of your product. When trying to copy this image to a floppy disk, MS-DOS gives an error message indicating it cannot create all of the files.

CAUSE

MS-DOS allows aproximantly 224 files to be placed in the root directory of a floppy disk. Therefore, if DSKLAYT2 creates a disk image with more than 224 files, MS-DOS will generate an error message when trying to copy this disk image to an actual floppy disk. DSKLAYT2, however, will not provide any warnings about the potential problem.

RESOLUTION

The MS-DOS 224-file limit only applies to the root directory of the floppy disk, and therefore the solution involves creating a subdirectory on the floppy disk and copying some of the files to the subdirectory. The .INF file must also be modified to reflect the new locations of the files.

Perform the following steps:

1. Create your disk images as normal.

2. When copying the problem disk image to a floppy disk, create a subdirectory on the floppy disk to receive most of the files. For example, suppose your target disk is in drive A:. Use

   ```
   md a:\files
   ```

   to create the subdirectory. Then, when copying the image files to the floppy disk, be sure to copy all the Setup Toolkit files (for example, SETUP.EXE, _MSTEST.EXE, SETUPAPI.INC, and so forth) to the root directory of the floppy disk and all the other files to the "files" subdirectory.

3. Modify the .INF file and place the new copy in the root of disk 1. Your

.INF file must be modified to reference the new subdirectory as follows:

Before

```
[Files]
    1, myfile1.exe,,,,1992-01-30,,,,,,,ROOT,,,13833,,6,,,
```

After

```
[Files]
    1, files\myfile1.exe,,,,1992-01-30,,,,,,,ROOT,,,13833,,6,,,
```

This modification is easy using the global search and replace capabilities of a good editor. For example, if disk 2 is the problem disk, search for all occurrences of "2, " and replace them with "2, files\".

4. Change the "STF_ROOT" line in the [Default File Settings] section of your .INF to read:

   "STF_ROOT" = "YES"

   If your .LST file specifies a compressed .INF file, you must use COMPRESS.EXE to compress your modified .INF file before copying it to disk 1.

Additional reference words: 3.10 MSSETUP
KBCategory: kbtool kbprg kbprb
KBSubcategory: TlsMss

PRB: Editing Labels in a TreeView Gives WM_COMMAND|IDOK

PSS ID Number: Q130692
Authored 24-May-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.51, 4.0
- Microsoft Win32s version 1.3

SYMPTOMS

WM_COMMAND|IDOK errors are received while editing labels in a TreeView control.

CAUSE

While editing labels in a TreeView control, the edit control created by the TreeView control can, and usually does, have an identifier of 1. This identifier is the same as IDOK. This can cause the parent window or dialog box to receive WM_COMMAND messages with an identifier of 1. Then the TreeView control passes on the EN_UPDATE and EN_CHANGE notifications from the edit control to the TreeView's parent.

This was a design decision made to meet system requirements and cannot be changed. If the parent window is going to perform some action in response to a command with an identifier of 1, this problem can occur. This problem is especially significant in dialog boxes that use the standard IDOK for a command button control.

RESOLUTION

Avoid using command and control identifiers with an identifier of 1 (IDOK). To be safe, the application should not use any identifiers less than 100 when used in conjunction with a TreeView control.

Another way to avoid this problem is to check the notification codes in the WM_COMMAND messages. Then respond only to the proper notification codes such as BN_CLICKED.

STATUS

This behavior is by design.

Additional reference words: 1.30 4.00 95
KBCategory: kbprg kbui kbprb
KBSubcategory: UsrCtl W32s

PRB: EndPage() Returns -1 When Banding

PSS ID Number: Q118873
Authored 01-Aug-1994 Last modified 22-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), version 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SYMPTOMS

When an application that implements banding calls EndPage(), EndPage() returns SP_ERROR (-1).

CAUSE

EndPage() returns -1 if there has been no corresponding call to StartPage(). Windows keeps track of whether StartPage() has been called by maintaining an internal flag that is set when StartPage() is called and then is cleared when EndPage() is called.

This flag is also cleared when the NEXTBAND escape is called and there are no more bands on the page to be printed. At this point, Windows clears the internal flag and tells the device that a page has been finished. Because the internal flag has been cleared, a subsequent call to EndPage() returns -1.

RESOLUTION

Though EndPage() returns -1 when it is called from printing code that implements banding, it does no harm. An application can safely call StartPage() and EndPage() when banding and ignore the -1 error returned from EndPage().

NOTE: It is not recommended that a Win32-based application use banding. Windows NT, spools in a journal file and Windows 95 spools in an enhanced metafile, so all GDI calls are supported without banding.

Additional reference words: 3.10 3.50 4.00 NEWFRAME
KBCategory: kbprg kbprb
KBSubcategory: GdiPrn

PRB: Error 1 (NRC_BUFLEN) During NetBIOS Send Call

PSS ID Number: Q124879
Authored 15-Jan-1995 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SYMPTOMS

When making a NetBIOS Send call, the call with fail with error code 1 (NRC_BUFLEN). This error code indicates that the buffer length is invalid.

CAUSE

NRC_BUFLEN will be returned if the buffer length specified in the NetBIOS Control Block (NCB) is incorrect. Less obvious is the fact that this error will also be returned if the buffer pointed to by the NCB is protected from write operations.

RESOLUTION

Although the NetBIOS code does not write to the buffer supplied in the NCB, write access is required. You can solve the problem by changing the protection on your buffer to include write access.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbnetwork kbprg kberrmsg
KBSubcategory: NtwkNetbios

PRB: Error Message Box Returned When DLL Load Fails

PSS ID Number: Q117330
Authored 26-Jun-1994 Last modified 22-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.5, 3.51, and 4.0

SYMPTOMS

Under Windows NT, when you load a DLL, a message box titled "Invalid DLL Entrypoint" is displayed and has the following text:

> The dynamic link library <name> is not written correctly.
> The stack pointer has been left in an inconsistent state.
> The entry point should be declared as WINAPI or STDCALL.
> Select YES to fail the DLL load. Select NO to continue
> execution. Selecting NO may cause the application to operate
> incorrectly.

Under Windows 95, the message box is titled "Error starting program" and the text is:

> The <dll file name> file cannot start. Check the file to
> determine the problem.

The user is not given a choice to continue, only an OK button. Pressing the OK button fails program load.

CAUSE

The system expects DLL entrypoints to use the _stdcall convention. If you use the _cdecl convention, the stack is not properly restored and subsequent calls into the DLL can cause a general protection fault (GPF).

This error message is new to Windows NT, version 3.5. Under Windows NT, version 3.1, the DLL is loaded without an error message, but the application usually causes a GPF when calling a DLL routine.

RESOLUTION

Correct the prototype of your entrypoint. For example, if your entrypoint is as follows:

```
BOOL DllMain( HANDLE hDLL, DWORD dwReason, LPVOID lpReserved)
```

change it to the following:

```
BOOL WINAPI DllMain( HANDLE hDLL, DWORD dwReason, LPVOID lpReserved)
```

Then, link with the following linker option to specify the entry point if you are using the C run-time:

```
-entry:_DllMainCRTStartup$(DLLENTRY)
```

MORE INFORMATION

If you are using the Microsoft C run-time, you need to use the entry point given in the RESOLUTION section in order to properly initialize the C run-time. For additional information, please see the following article in the Microsoft Knowledge Base:

```
ARTICLE-ID: Q94248
TITLE     : Using the C Run Time
```

REFERENCES

For more information on the DLL entrypoint, please search on the topic "DllEntryPoint" (without the quotes) in the Win32 API help file.

Additional reference words: 3.50 4.00 libmain
KBCategory: kbprg kbprb
KBSubcategory: BseDll

PRB: Error on Win32s: R6016 - not enough space for thread

PSS ID Number: Q126709
Authored 28-Feb-1995 Last modified 03-Mar-1995

The information in this article applies to:

- Microsoft Win32s, versions 1.2 and 1.25

SYMPTOMS

Spawning and closing an application repeatedly succeeds around 60 times, then the spawn fails with this error:

R6016 - not enough space for thread data

CAUSE

The thread local storage (TLS) is not freed by the system.

The failure occurs only if there is another Win32-based application active while you are doing the spawns. The message itself is not generated by Win32s. It is generated by the Microsoft C Run-time (CRT) libraries LIBC.LIB and LIBCMT.LIB.

RESOLUTION

In Win32s version 1.25, TLS indices are freed during module cleanup. The TLS index is owned by the application's main module, so that it is freed when the application terminates. This solves the problem for LIBC and LIBCMT.

There is a similar problem with MSVCRT20.DLL. This DLL version of the CRT allocates a new TLS index each time a process attaches to it. MSVCRT20 doesn't free the TLS indices when unloading. The system should free them. If only one application uses MSVCRT20 at a time, then the application can be spawned successfully up to about 60 times on Win32s version 1.20. On Win32s version 1.25, there is no limitation.

If there is already an active application that uses MSVCRT20, it is not possible to spawn and close a second application that uses MSVCRT20 more than about 60 times under Win32s version 1.25. This is because MSVCRT20 allocates a TLS index each time a process attaches to it. Win32s will free all of the TLS indices only when MSVCRT20 is unloaded.

MORE INFORMATION

On Win32s, TLS allocation should be done once and not per process. Each process can use the index to store per-process data, just as a thread uses a TLS index on Windows NT. This is easy to do, because DLL data is

shared between all processes under Win32s.

The following example demonstrates how to do the TLS allocation once on Win32s:

```
   BOOL APIENTRY DllMain(HINSTANCE hinstDll, DWORD fdwReason,
           LPVOID lpvReserved)
   {
       static BOOL fFirstProcess = TRUE;
       BOOL fWin32s = FALSE;
       DWORD dwVersion = GetVersion();
       static DWORD dwIndex;

       if ( !(dwVersion & 0x80000000) && LOBYTE(LOWORD(dwVersion))<4 )
           fWin32s = TRUE;

       if (dwReason == DLL_PROCESS_ATTACH) {
           if (fFirstProcess || !fWin32s) {
               dwIndex = TlsAlloc();
            }
            fFirstProcess = FALSE;
       }
       .
       .
       .
   }
```

Additional reference words: 1.20
KBCategory: kbprg kbcode kbprb
KBSubcategory: W32s

PRB: Error with GetOpenFileName() and OFN_ALLOWMULTISELECT

PSS ID Number: Q99338
Authored 26-May-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, version 3.5

SYMPTOMS

Assume that the GetOpenFileName() function is called with the Flags parameter set to OFN_ALLOWMULTISELECT (allow for multiple selections in the list box) and OFN_NOVALIDATE (allow for invalid characters in the filename), the lpstrFile parameter points to a 2K buffer, and the corresponding nMaxFile is set appropriately to 2048.

When the GetOpenFileName() function call returns, the complete selection in the list box is not copied to the lpstrFile buffer, but the string in the buffer is truncated.

CAUSE

This is a problem with the GetOpenFileName() function in the current version of COMMDLG.DLL in that the OFN_NOVALIDATE flag cannot be used when multiple selections are allowed.

RESOLUTION

The following are two suggested solutions to this problem:

One solution for this problem is to not use the OFN_NOVALIDATE flag with the OFN_ALLOWMULTISELECT flag. That is, if only the OFN_ALLOWMULTISELECT flag is used, then multiple selections will be copied properly to the text buffer. Note that there is a buffer size limit of 2K bytes for the lpstrFile buffer, and characters are truncated after 2K bytes when the OFN_ALLOWMULTISELECT flag is used.

There is another solution, if both the OFN_ALLOWMULTISELECT and OFN_NOVALIDATE flags must be used simultaneously with GetOpenFileName(). Note that the entire string is always copied into the edit control even though the text gets truncated during the process of copying the text to the lpstrFile buffer. Therefore, one could write a hook procedure and read the entire string from the edit control and use it appropriately instead of using the lpstrFile buffer.

Additional reference words: 3.10 3.50
KBCategory: kbprg kbprb
KBSubcategory: UsrCmnDlg

PRB: ERROR_INVALID_PARAMETER from WriteFile() or ReadFile()

PSS ID Number: Q110148
Authored 13-Jan-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SYMPTOMS

The WriteFile() or ReadFile() function call may fail with the error ERROR_INVALID_PARAMETER if you are operating on a named pipe and using overlapped I/O.

CAUSE

A possible cause for the failure is that the Offset and OffsetHigh members of the OVERLAPPED structure are not set to zero.

RESOLUTION

Set the Offset and OffsetHigh members of your OVERLAPPED structure to zero.

STATUS

This behavior is by design. The online help for both WriteFile() and ReadFile() state that the Offset and OffsetHigh members of the OVERLAPPED structure must be set to zero or the functions will fail.

MORE INFORMATION

In many cases the function calls may succeed if you do not explicitly set OVERLAPPED.Offset and OVERLAPPED.OffsetHigh to zero. However, this is usually either because the OVERLAPPED structure is static or global and therefore is initialized to zero, or the OVERLAPPED structure is automatic (local) and the contents of that location on the stack are already zero. You should explicitly set the OVERLAPPED.Offset and OVERLAPPED.OffsetHigh structure members to zero.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: BseFileio

PRB: Errors When Windbg Switches Not Set for Visual C++ App

PSS ID Number: Q131111
Authored 04-Jun-1995 Last modified 22-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5

SYMPTOMS

If two switches are not set correctly, Windbg gives the following error message when the module is loaded:

```
(symbol format not supported)
```

In addition, Windbg gives the following error message for any breakpoints set in the module:

```
Unresolved Breakpoint
```

CAUSE

For Windbg to understand the symbolic format generated from Microsoft Visual C++ version 2.0, two linker switches have to be set correctly:

- /DEBUG is set in the Project Settings dialog box, under the Link tab. Choose the Debug category. Then select the Generate Debug Info check box, and choose Microsoft Format.

- /PDB:none is set in the Project Settings dialog box, under the Link tab. Choose the Customize category. Then clear the Use Program Database check box.

STATUS

This behavior is by design.

Additional reference words: 3.50
KBCategory: kbtool kbprb
KBSubcategory: TlsWindbg

PRB: Excel's =REQUEST() from DDEML Application Returns #N/A

PSS ID Number: Q107980
Authored 01-Dec-1993 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SYMPTOMS

When executing the =REQUEST() macro to request data from a DDEML server application, Excel returns a value of "#N/A", although the server application returned a valid data handle from the request.

CAUSE

When Excel executes a =REQUEST, it requests data in the most efficient format available. Verifying through DDESPY when Excel executes the =REQUEST macro, one can see that Excel sends out a request for data for each format, in this order:

```
XLTable  (Excel fast table format)
BIFF4    (Excel 4.0 file format)
BIFF3    (Excel 3.0 file format)
SYLK     (Symbolic Link)
WK1      (Lotus 1-2-3 release 2 format)
CSV      (comma-delimited text)
TEXT     (CF_TEXT)
RTF      (rich text format)
DIF      (data interchange format)
```

Knowing what formats it can handle best, Excel requests data in the most efficient format first, and so on, until it finds one that the server application supports. At this point, Excel stops sending further requests.

In response to a request, a DDEML server application that supports only one format (for example, the CF_TEXT format) may return a data handle in CF_TEXT format, regardless of the format being requested. When Excel sends its first request for data in XLTable format, this server application returns a data handle in CF_TEXT format, as demonstrated in the code below:

```
case XTYP_REQUEST:
   if ((ghConv == hConv) &&
        (!DdeCmpStringHandles (hsz1, hszTopicName)) &&
        (!DdeCmpStringHandles (hsz2, hszItemName)))  {

          lstrcpy (szBuffer, "The Simpsons");
          return (DdeCreateDataHandle (idInst,
                              szBuffer,
                              lstrlen (szBuffer)+1,
```

```
                                    0L,
                                    hszItemName,
                                    CF_TEXT,
                                    0);
      }
      return (HDDEDATA)NULL;
```

Because Excel expected to receive data in the format it had requested (that is, XLTable format), and instead received date in CF_TEXT format, Excel returns #N/A, not knowing how to handle the data it received.

RESOLUTION

In response to a request, a DDEML server application should return a valid data handle only for the format it supports. When processing an XTYP_REQUEST transaction, the server application should first check whether the data being requested is in its supported format; if so, the server application should return an appropriate data handle. Otherwise, the server application should return NULL.

The code above can be modified as follows to check for this condition:

```
   case XTYP_REQUEST:
      if ((ghConv == hConv) &&
           (!DdeCmpStringHandles (hsz1, hszTopicName)) &&
           (!DdeCmpStringHandles (hsz2, hszItemName)) &&
           (wFmt == CF_TEXT))  {     // Add this to the if clause
                                     // to check if data is being requested
                                       // in one of its supported formats.

                lstrcpy (szBuffer, "Fred Flintstone");
                return (DdeCreateDataHandle (idInst,
                                    szBuffer,
                                    lstrlen (szBuffer)+1,
                                    0L,
                                    hszItemName,
                                    CF_TEXT,
                                    0);
        }
        return (HDDEDATA)NULL;
```

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbprb kbcode
KBSubcategory: UsrDde

PRB: ExitProgman DDE Service Does Not Work If PROGMAN Is

PSS ID Number: Q69899
Authored 06-Mar-1991 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SYMPTOMS

Calling the ExitProgman() function documented in the Microsoft Windows SDK version 3.0 "Guide to Programming," section 22.4.4 (pages 22-19 through 22-22) fails under certain circumstances.

CAUSE

Calling this function will fail if the Program Manager is the Windows shell.

RESOLUTION

This behavior is by design. The Windows 3.1 documentation states:

> If Program Manager was started from another application, the ExitProgman command instructs Program Manager to exit and, optionally, save its groups information.

For another application to start Program Manager, the Program Manager cannot be the shell.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: UsrDde

PRB: FindText, ReplaceText Hook Function

PSS ID Number: Q96135
Authored 09-Mar-1993 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SYMPTOMS

When adding a hook function to the FindText and/or ReplaceText common dialog box, the dialog box is not shown. However, the hook function is receiving messages and the dialog box window does exist.

CAUSE

The hook function is returning FALSE after processing the WM_INITDIALOG message.

RESOLUTION

After processing the WM_INITDIALOG message in the hook function, return TRUE. If your hook function is setting the focus to a specific control, and therefore should return FALSE, add the following code:

```
   case WM_INITDIALOG:

      ...WM_INITDIALOG code...

      SetFocus(hCtrl);
      ShowWindow(hDlg, SW_NORMAL);
      UpdateWindow(hDlg);
      return(FALSE);
```

This code ensures that the dialog box is visible.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbprb kbcode
KBSubcategory: UsrCmnDlg

PRB: GetExitCodeProcess() Always Returns 0 for 16-Bit

PSS ID Number: Q111559
Authored 14-Feb-1994 Last modified 27-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5
- Microsoft Win32s versions 1.1, 1.15, and 1.2

SYMPTOMS

GetExitCodeProcess() always returns a status of 0 (zero) when the handle for a 16-bit process is passed. This applies to both Windows NT and Win32s.

STATUS

This behavior is by design in the Microsoft products listed at the beginning of this article. Microsoft may add functionality in future versions that support exit codes from 16-bit processes.

Additional reference words: 3.10 3.50 1.10 1.20
KBCategory: kbprg kbprb
KBSubcategory: BseMisc W32s

PRB: GetLogicalDrives() Indicates that Drive B: Is Present

PSS ID Number: Q126626
Authored 27-Feb-1995 Last modified 28-Feb-1995

The information in this article applies to:

- Microsoft Win32s, versions 1.2 and 1.25

SYMPTOMS

GetLogicalDrives() returns a bitmask that indicates that drive B is an available drive present on the system, even though there is no physical drive B.

CAUSE

Drive B is a ghosted drive, so you can use it even if it does not exist. This is useful for performing a diskcopy.

RESOLUTION

Use GetDriveType() to determine whether drive B: is present as a physical device.

STATUS

This behavior is by design.

Additional reference words: 1.20 1.25
KBCategory: kbprg kbprb
KBSubcategory: W32s

PRB: GetOpenFileName() and Spaces in Long Filenames

PSS ID Number: Q108233
Authored 07-Dec-1993 Last modified 29-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SYMPTOMS

GetOpenFileName() is the application programming interface (API) for the open file common dialog box. This API displays the long filenames (LFNs) on NTFS and HPFS.

When using the OFN_ALLOWMULTISELECT flag with the GetOpenFileName() API, the dialog box automatically presents the 8.3 names for all LFNs that contain embedded spaces.

CAUSE

The original design of GetOpenFileName() uses a filename list that is space-delimited when the OFN_ALLOWMULTISELECT flag is specified. Thus, there is no programmatic way to determine which string tokens are complete filenames or fragments of a complete name with spaces.

STATUS

This behavior is by design. Microsoft is considering changing this behavior in a future release of Windows NT.

MORE INFORMATION

Historically, FAT filenames that contained embedded spaces were branded as "illegal," even though the specifications of the FAT file system do not impose such a restriction. For example, many of the MS-DOS command-line utilities do no allow the user to specify filenames with embedded spaces, because of difficulties that would be introduced in parsing the command line. Under Windows NT, the command utilities have been enhanced to support such names if they are in quotation marks.

Additional reference words: 3.10 3.50
KBCategory: kbprg kbprb
KBSubcategory: UsrCmnDlg

PRB: Getsockopt() Returns IP Address 0.0.0.0 for UDP

PSS ID Number: Q129065
Authored 18-Apr-1995 Last modified 19-Apr-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.11 and 4.0
- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SYMPTOMS

By following the steps listed below, you might think you should get back the interface address over which the connection was made. However, it actually returns the address 0.0.0.0.

1. Open a UDP socket.

2. Bind it to INADDR_ANY.

3. Call connect() to make a UDP connection.

4. Call getsockname() on your socket.

However, if it was a TCP socket, you would get back the IP address of the interface.

CAUSE

UDP

This is the behaviour expected from some flavors of UNIX, notably those derived from BSD. When an application calls connect() on a UDP socket that is bound to INADDR_ANY, the operating system associates the remote address with the local socket. This saves the programmer from having to specify the remote IP address in each sendto() or recvfrom(). Instead they may use send() and recv(). Note that this is just a convenience provided by the operating system; there is no network traffic associated with this call. At this point, the underlying IP software determines the interface over which packets will be sent. As described earlier, under BSD UNIX, calling getsockname() will return the IP address of the interface to the application.

This however, is not expected behaviour under Windows NT, Windows 95, or Microsoft TCP IP/32 for Windows for Workgroups version 3.11. Calling getsockname() will return the IP address 0.0.0.0 (INADDR_ANY). Applications should not assume that they can get the IP address of the interface.

TCP

The behaviour is different if it was a TCP socket. In this case, calling getsockname() on a connected socket that was bound to INADDR_ANY will return the IP address of the interface over which the connection was made. The state of the connection can also be observed by typing 'netstat' at a command prompt.

NOTE: To enumerate all the IP addresses on an IP host, the application should call gethostname(), call gethostbyname(), and then iterate through the h_addr_list[] member of the hostent struct returned by gethostbyname() as in this example:

```
char     Hostname[100];
HOSTENT *pHostEnt;
int      nAdapter = 0;

gethostname( Hostname, sizeof( Hostname ));
pHostEnt = gethostbyname( Hostname );

while ( pHostEnt->h_addr_list[nAdapter] )
{
   // pHostEnt->h_addr_list[nAdapter] -the current address in host order
   nAdapter++;
}
```

STATUS

This behavior is by design.

Additional reference words: 3.11 4.00 3.10 3.50 3.51
KBCategory: kbnetwork kbprb
KBSubcategory: NtwkWinsock

PRB: GetVolumeInformation() Fails with UNC Name

PSS ID Number: Q119219
Authored 10-Aug-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SYMPTOMS

GetVolumeInformation() fails and GetLastError() returns 123 (ERROR_INVALID_NAME) if a UNC name is used. The UNC name has the form \\<SERVER>\<SHARE>.

RESOLUTION

GetVolumeInformation() requires an extra backslash with UNC names, so that the name has the form \\<SERVER>\<SHARE>\.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: BseMisc

PRB: GlobalAlloc() Pagelocks Blocks on Win32s

PSS ID Number: Q114611
Authored 08-May-1994 Last modified 21-Dec-1994

The information in this article applies to:

- Microsoft Win32s, versions 1.1, 1.15, and 1.2

SYMPTOMS

If a Win32-based application running in Win32s uses GlobalAlloc() to allocate memory from the global heap with GMEM_FIXED, with GPTR, with GMEM_ZEROINIT, or without specifying GMEM_MOVEABLE the memory allocated will be fixed and page-locked.

CAUSE

When a Win32 application running under Win32s on Windows 3.1 calls GlobalAlloc() the call is translated via a thunk supplied by Win32s in a 16-bit DLL. The 16-bit DLL then calls the Windows 3.1 function GlobalAlloc(). When GlobalAlloc() is called from a DLL in Windows 3.1 the allocated memory will be fixed and page-locked unless GMEM_MOVEABLE is specified.

RESOLUTION

The GlobalAlloc() flags should always include GMEM_MOVEABLE if memory does not need to be fixed and page-locked. This is expected behavior for Windows 3.1.

MORE INFORMATION

A Windows-based application will not fix or page-lock memory even when specifically using the GMEM_FIXED flag. This behavior is unique to Windows version 3.1; using GlobalAlloc() with GMEM_FIXED to allocate fixed and page-locked memory must be done in a DLL.

In Windows 3.1, the GMEM_FIXED flag is defined as 0x0000. Using GMEM_ZEROINIT without GMEM_MOVEABLE will command GlobalAlloc() to allocate using GMEM_FIXED by default. Since Win32s passes all GlobalAlloc() calls to the Windows 3.1 GlobalAlloc() by a DLL, GlobalAlloc() called from either a Win32 application or a Win32 DLL will allocate the block fixed and page-locked unless the GMEM_MOVEABLE flag is specified.

The following code illustrates this case:

```
{
HGLOBAL hMem;

// allocate a block from the global heap
```

```
   hMem = GlobalAlloc(GMEM_ZEROINIT, 512);

      .
      .
      .

   }
```

Although this source code is compatible between applications for Windows 3.1 and applications for Windows NT running on Win32s, the result is different. A 16-bit application running on Windows 3.1 will allocate the memory as moveable and zero the contents. A Win32 application running on Win32s will allocate the memory as fixed and page-locked and zero the contents.

REFERENCES

Appendix B, titled "System Limits", of the "Win32s Programmer's Reference Manual" briefly mentions on page 56 not to use GMEM_FIXED in GlobalAlloc() called by 32-bit applications.

Additional reference words: 1.10 1.20 3.10
KBCategory: kbprg kbprb
KBSubcategory: W32s

PRB: GP Fault Caused by GROWSTUB in POINTER.DLL

PSS ID Number: Q117864
Authored 11-Jul-1994 Last modified 16-Dec-1994

The information in this article applies to:

- Microsoft Win32s versions 1.1, 1.15, and 1.2

SYMPTOMS

As soon as your Win32-based application starts, there is a GP fault caused by GROWSTUB in POINTER.DLL.

CAUSE

This problem is caused by a bug in GROWSTUB, which is part of the Microsoft Mouse driver version 9.01.

RESOLUTION

Microsoft Mouse driver version 9.01b corrects this problem. Driver 9.01b is available via part number: 135-099-351 "Intellepoint Mouse Driver 2.0(dual)". This driver does not introduce new functionality, therefore, you need only upgrade if you have run into this problem.

For more information, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID: Q119775
TITLE : HD1061: POINTER.DLL Corrects GP Fault with Win32 Apps

You can also avoid the problem by removing POINTER.EXE from the load= line in your WIN.INI file.

Additional reference words: 1.10
KBCategory: kbprg kbprb
KBSubcategory: W32s

PRB: GP Fault in DDEML from XTYP_EXECUTE Timeout Value

PSS ID Number: Q83999
Authored 27-Apr-1992 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SYMPTOMS

A general protection (GP) fault occurs in DDEML When the following occurs:

1. A DDEML (Dynamic Data Exchange Server Library) server application requires more time to process a XTYP_EXECUTE transaction than the timeout value specified by a DDEML client application

2. The server application creates windows as part of its processing

3. The client application abandons the transaction because the transaction timed out

CAUSE

The server application receives a window handle with the same value as the hidden window created to control the transaction.

RESOLUTION

Specify a timeout value in the client application longer than the time required by the server application to complete the task.

MORE INFORMATION

To use DDEML, an application (either a client or a server) registers a callback function with the library. The DDEML calls the callback function for any DDE activity. A DDE transaction is similar to a message; it contains a named constant, accompanied by other parameters.

A client application issues a XTYP_EXECUTE transaction to instruct the server application to execute a command. When a client calls the DdeClientTransaction function to issue a transaction, it can specify a timeout value, which is the amount of time (in seconds) the client is willing to wait while the server processes the transaction. If the server fails to execute the command within the specified timeout value, the DDEML sends a message to the client that the transaction timed out. Upon receipt of this message, the client can inform the

user, reissue the command, abandon the transaction, or take other appropriate actions.

If a client application specifies a short timeout period (one second, for example) and the server requires fifteen seconds to execute a command, the client will receive notification that the transaction timed out. If the client terminates the transaction, which is an appropriate action, the DDEML will GP fault.

When the client sends an XTYP_EXECUTE transaction, the DDEML creates a hidden window for the conversation. If the client calls the DdeAbandonTransaction function to terminate the transaction, the DDEML destroys the associated hidden window.

At the same time, the server application processes the execute transaction, which might involve creating one or more windows. If the server creates a window immediately after the DDEML destroys a window, the server receives a window handle with the same value as that of the destroyed window. After the server completes processing the execute transaction, it returns control to the DDEML.

Normally, the DDEML determines that the callback function is returning to a conversation that has been terminated. It calls the IsWindow function with the window handle for the transaction's hidden window to ensure that the handle remains valid.

Because the window handle has been allocated to the server application, the IsWindow test succeeds. However, this handle no longer corresponds to the transaction's hidden window. Therefore, when the DDEML attempts to retrieve the pointer kept in the hidden window's window extra bytes, the pointer is not available. When the DDEML uses the contents of this memory, a GP fault is likely to result.

The current way to work around this problem is to specify a timeout value in the client application that is longer than the time required by the server to complete its processing.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: UsrDde

PRB: GPF When Spawn Windows-Based App w/ WinExec() in Win32s

PSS ID Number: Q121095
Authored 26-Sep-1994 Last modified 28-Nov-1994

The information in this article applies to:

- Microsoft Win32s, versions 1.1, 1.15, and 1.2

SYMPTOMS

Win32-based applications running under Win32s can spawn both Windows-based and Win32-based applications by using either WinExec() or CreateProcess(). However, there is a case where spawning a Windows-based application with WinExec() does not work as expected and may cause a general protection (GP) fault.

CAUSE

There is a bug in the C start-up code that comes with Microsoft C version 6.0. If you spawn an application built with Microsoft C version 6.0 by calling LoadModule() with an explicit environment, the application does not run correctly. This is true whether the application was spawned from a Win32-based application or a Windows-based application. Win32s calls LoadModule() with an explicit environment when you spawn a Windows-based application with WinExec(). As a result, under Win32s version 1.1 and 1.15, WinExec() will report success, but the Windows-based application built with Microsoft C version 6.0 may cause a GP fault.

RESOLUTION

The best solution is to rebuild the application with another compiler package. However, because a number of Windows accessories (such as Notepad and Write) were built with Microsoft C version 6.0 and you cannot modify these applications, changes were introduced into Win32s version 1.2 to help you work around this problem. These changes are detailed in the More Information section below.

NOTE: Win32s uses a different mechanism to spawn Win32-based applications, so the problems discussed in this article do not occur when spawning Win32-based applications with WinExec().

MORE INFORMATION

In Win32s version 1.2, WinExec() does not pass the environment to the spawned application (child). The child receives the standard global environment strings. This allows the application to run, but the child does not receive the modified environment from the parent. This seemed to be a reasonable compromise, because most applications do not change the environment for the child. If an application must modify the child's

environment, it can spawn the application using CreateProcess() and specify an explicit environment. However, if the child was built using Microsoft C version 6.0, it may cause a GP fault. In addition, if the parent exits, the child's environment becomes invalid. These three problems are not specific to Win32s and will happen with Windows-based applications as well.

Additional reference words: 1.00 1.15 1.20 GPF
KBCategory: kbprg kbprb
KBSubCategory: W32s

PRB: Inadequate Buffer Length Causes Strange Problems in

PSS ID Number: Q107387
Authored 18-Nov-1993 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.1, 3.5, 3.51, and 4.0

SYMPTOMS

Specifying an inadequate buffer length for an XTYP_POKE or an XTYP_EXECUTE command causes strange problems in DDEML.

Problems can range from a general protection (GP) fault or Exception 13, to DDEML timeout errors (such as DMLERR_EXECACKTIMEOUT or DMLERR_POKEACKTIMEOUT) or a DDEML transaction failure (or DMLERR_NOTPROCESSED). Sometimes, the application may seem to work for the most part, and then occasionally crash.

Data can be passed to the server application via XTYP_POKE or XTYP_EXECUTE in two ways:

- Directly, as a pointer to the data or command string, as in the sample code below:

```
char lpszString [80];

lstrcpy (lpszString, "[FileOpen(""C:\README.DOC"")]");
DdeClientTransaction (lpszString,             // string buffer
                      lstrlen (lpszString)+1, // string buffer length
                      hConv,
                      hszItem,
                      CF_TEXT,
                      XTYP_POKE,
                      1000,
                      NULL);
```

-or-

- By creating a data handle, and passing that on to the DdeClientTransaction() call:

```
char lpszString [80];
HDDEDATA hData;

lstrcpy (lpszString, "[FileOpen(""C:\README.DOC"")]");
hData = DdeCreateDataHandle (idInst,
                             lpszString,
                             lstrlen (lpszString)+1,
                             0,
                             NULL,
```

```
                                CF_TEXT,
                                0);
   if (!hData)
      DdeClientTransaction (hData,     // string buffer
                            -1,        // indicates hData is a data handle
                            hConv,
                            hszItem,
                            CF_TEXT,
                            XTYP_POKE,
                            1000,
                            NULL);
```

CAUSE

Because data is most commonly passed between applications in CF_TEXT format, a common problem with the string buffer length is setting it to lstrlen (lpszString), where lpszString is the buffer containing the string the client needs to pass to the server. Because the lstrlen() function does not include the terminating null character, this can cause the system to append garbage characters to the end of the string, thus sending an invalid string to the server application.

RESOLUTION

When passing strings between two applications, the string buffer length should be set to lstrlen (lpszString) +1, to include the terminating null character ('\0').

Using DDESPY, it is easy to track down this problem, because one can follow the string being passed from the client to the server application. Garbage characters incorrectly being appended to the string usually indicate a problem with specifying an inadequate string buffer length.

Additional reference words: 3.10 3.50 3.51 4.00 95 gpf gp-fault
KBCategory: kbprg kbprb kbcode
KBSubcategory: UsrDde

PRB: Inconsistencies in GDI APIs Between Win32s and Windows

PSS ID Number: Q123421
Authored 01-Dec-1994 Last modified 01-Dec-1994

The information in this article applies to:

- Microsoft Win32s version 1.2

SYMPTOMS

The StretchBlt() and StretchDIBits()/SetDIBits() GDI APIs do not behave consistently under Win32s and Windows NT.

StretchBlt()

If the source width and height specified in the call to StretchBlt() are greater than the actual bitmap width and height, StretchBlt() fails. The same call to StretchBlt() succeeds under Windows NT.

StretchDIBits()/SetDIBits()

If the memory pointed to by the lpBits parameter is read-only, the call to StretchDIBits()/SetDIBits() fails.

NOTE: When a Win32-based application uses the memory returned from LockResource() as a parameter to SetDIBits(), by default, it's using read-only memory, because the resource section is defined by default as read-only.

CAUSE

These problems are due to bugs in Windows. In the case of StretchDIBits() and SetDIBits(), Windows mistakenly verifies that the buffer is writable. This problem does not show up in a 16-bit Windows-based application running under Windows because resources are loaded into read/write (global) memory.

RESOLUTION

In Win32s version 1.25, Win32s will always make the resource section read/write, regardless of what is specified in the section attributes. This will work around the problem. In the meantime, use the following resolutions:

StretchBlt()

To work around the problem, specify the proper width and height for the source bitmap.

StretchDIBits()/SetDIBits()

To work around the problem, do one of the following:

- Copy the memory to a temporary read/write buffer.

  -or-

- Use the linker switch /SECTION:.rsrc,rw to make the resource section read/write. Windows NT will allocate separate resource sections for each copy of the application.

Additional reference words: 1.20
KBCategory: kbprg kbprb
KBSubcategory: W32s

PRB: Inter-thread SetWindowText() Fails to Update Window Text

PSS ID Number: Q125687
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SYMPTOMS

Calling SetWindowText() to set a static control text from a thread other than the one that created the control fails to display the new text in Windows 95.

CAUSE

When SetWindowText() is called from another thread, instead of sending a WM_SETTEXT message to the appropriate window procedure, only DefWindowProc() is called, so the edit and static controls do not paint the control appropriately because the appropriate code is never executed, so the text on the screen is never updated. In other words, calling SetWindowText() updates the buffer internally, but the change is not reflected on the screen.

RESOLUTION

One obvious workaround is to refrain from calling SetWindowText() from another thread, if possible.

If design considerations don't allow doing this, use one of these workarounds:

- Send a WM_SETTEXT message directly to the window or control.

  -or-

- Call InvalidateRect() immediately after the SetWindowText(). This works because DefWindowProc() updates the buffer where the text is stored.

STATUS

This inter-thread SetWindowText() behavior is by design in Windows version 3.x. It was maintained in Windows 95 for backward compatibility purposes. Applications written for Windows version 3.x can expect their inter-thread SetWindowText() calls to behave as they did in Windows version 3.x.

MORE INFORMATION

Calling SetWindowText() from another thread in Windows NT displays the window text correctly, so it works differently from Windows 95.

Additional reference words: 4.00
KBCategory: kbprg kbprb
KBSubcategory: UsrWndw

PRB: IsCharAlpha Return Value Different Between Versions

PSS ID Number: Q84843
Authored 21-May-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SYMPTOMS

Under Windows version 3.1, the IsCharAlpha function returns TRUE for the character values 8Ah, 8Ch, 9Ah, 9Ch, 9Fh, and DFh. Under Windows version 3.0, the function returns FALSE for these character values.

CAUSE

These characters represent alphabetic characters that were added to the Windows character set in Windows 3.1.

RESOLUTION

Applications that use the IsCharAlpha function should behave properly with the newly-defined characters. No changes should be required.

MORE INFORMATION

Appendix C.1, page 596, of the "Microsoft Windows Software Development Kit: Programmer's Reference, Volume 3: Messages, Structures, and Macros" lists the Windows character set.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: UsrLoc

PRB: IsGdiObject() Is Not a Part of the Win32 API

PSS ID Number: Q91072
Authored 27-Oct-1992 Last modified 05-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SYMPTOMS

There is no IsGdiObject() function in the Win32 API.

CAUSE

The function was added to the Windows 3.1 API because passing a handle to a non-GDI object to a GDI function causes a GP fault under Windows 3.0. Windows NT and Windows 95 detect whether the APIs are passed an inappropriate handle, so the function can return an error.

RESOLUTION

IsGdiObject() is not needed on Windows NT or Windows 95.

Additional reference words: 3.10 3.50
KBCategory: kbprg kbprb
KBSubcategory: GdiMisc

PRB: JournalPlayback Hook Can Cause Windows NT to Hang

PSS ID Number: Q124835
Authored 12-Jan-1995 Last modified 13-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SYMPTOMS

Incorrect use of the delay return value from a journal playback hook can cause Windows NT to hang temporarily.

CAUSE

The menu loop in Windows NT calls PeekMessage() with the PM_NOREMOVE flag, does some processing, and then removes the message from the queue. This sequence is repeated until the menu goes away. When JournalPlayback is occurring, the PeekMessage(PM_NOREMOVE) results in a callback to the application's JournalPlaybackProc with an HC_ACTION code. The subsequent PeekMessage(PM_REMOVE) also calls the JournalPlaybackProc with an HC_ACTION code. If the peek is successful, it is followed by an HC_SKIP callback.

In order to have playback from a journal playback hook occur at a certain rate, Microsoft designed it so that the value returned by the JournalPlaybackProc can be non-zero. This value represents the number of clock-ticks the system should wait before processing the event. What the documentation doesn't make clear is that when the delay has expired, another callback to the JournalPlaybackProc is made to obtain the same event again; the event provided with the previous non-zero delay is not used at all. All subsequent HC_ACTION calls that request the same event should be returned with a zero delay value. Only after an HC_SKIP callback has been made, may an HC_ACTION callback return a non-zero delay value again. Some applications do not do this correctly, and simply alternate between returning a delay and returning a non-delay.

This alternating delay/no delay method made the Windows NT menu loop hang because the PeekMessage(PM_NOREMOVE) would get an input event (with no delay), then the PeekMessage(PM_REMOVE) would get a non-zero return value from the JournalPlaybackProc. This represents no message -- so instead of issuing an HC_SKIP callback to the JournalPlaybackProc to advance to the next event, the Windows NT menu loop code simply looped back to the PeekMessage(PM_NOREMOVE) getting stuck in an infinite loop.

RESOLUTION

To work around this problem, make sure the JournalPlaybackProc correctly returns the delay only for the first request for an event.

Neither Windows version 3.1 nor Windows 95 have this problem.

STATUS

This behavior is by design.

MORE INFORMATION

The following sample code demonstrates correct and incorrect methods of handling delays in a journal playback hook.

Sample Code

```
LRESULT CALLBACK JournalPlaybackProc(
    int nCode,
    WPARAM wParam,
    LPARAM lParam)
{
    static BOOL     fDelay;
    static EVENTMSG event;
    static LRESULT  ticks_delay;
    BOOL            fCallNextHook = FALSE;
    LRESULT         lResult = 0;

    switch( nCode )
    {
        case HC_SKIP:
            fDelay = TRUE;      // <<<< CORRECT PLACE TO RESET fDelay

            // Get the next event from the list.  If the routine returns
            // FALSE, then we are done - release the hook.
            if( !GetNextEvent( &event, &ticks_delay ))
                SetJournalHook( FALSE, NULL );
            break;

        case HC_GETNEXT:
        {
            // Structure information returned from previous GetNextEvent
            // call
            LPEVENTMSG lpEvent = (LPEVENTMSG) lParam;

            // Set the event
            *lpEvent = event;

            if( fDelay )
            {
                // Toggle pause variable so that the next call won't
                // pause.  Return the pause length specified by ticks_delay
                // since this is the first time the event has been
                // requested.
```

```
                fDelay = FALSE; // <<<< CORRECT PLACE TO CLEAR fDelay
                return( ticks_delay );
            }
            break;
        }

        case HC_SYSMODALOFF:
            // System modal dialog is going away - something really got
            // hosed.  Windows took care of removing our JournalPlayback
            // hook, so no need to call SetJournalHook( FALSE ).

            fCallNextHook = TRUE;
            break;

        case HC_SYSMODALON:
        default:
            // Something is is not right here, let the next hook handle
            // it.

            fCallNextHook = TRUE;
            break;
    }

    // If the event wasn't processed by our code, call next hook
    if( fCallNextHook )
        lResult = CallNextHookEx( s_journalHook, nCode, wParam, lParam );

    // fDelay = TRUE;           // <<<< WRONG PLACE TO RESET bDelay !!!
    return lResult;
}
```

Additional reference words: 3.10 3.50
KBCategory: kbprg kbprb
KBSubcategory: UsrHks

PRB: Large DIBs May Not Display Under Win32s

PSS ID Number: Q126575
Authored 26-Feb-1995 Last modified 27-Feb-1995

The information in this article applies to:

- Microsoft Win32s versions 1.10, 1.15, or 1.20

SYMPTOMS

DIB functions fail when using large DIBs under Win32s.

CAUSE

There is a two-megabyte limit on the size of the area of a DIB that can be blitted using blting functions under Win32s. In versions of Win32s up to 1.2, Microsoft set this size to accommodate DIB blts of 1024*768*24 bits-per-pixel. In version 1.25, the maximum size of the blitted area will be enlarged to accommodate 1280*1024*24 bits-per-pixel.

The following functions are affected:

  SetDIBits
  SetDIBitsToDevice
  CreateDIBitmap
  StretchDIBits

WORKAROUND

To work around the problem, break down large blts into bands that are smaller than two megabytes. Please keep in mind that the biSizeImage field of the BITMAPINFOHEADER used with the blting functions will need to be set to a value smaller than the DIB size limit.

The following code demonstrates a simple implementation of StretchDIBits() that can be used with large DIBs under Win32s.

```
/* Macro to determine the bytes in a DWORD aligned DIB scanline */
#define BYTESPERLINE(Width, BPP) ((WORD)((((DWORD)(Width) *
(DWORD)(BPP) + 31) >> 5)) << 2)

int NewStretchDIBits(
    HDC  hdc,     // handle of device context
    int  XDest,     // x-coordinate of upper-left corner of dest. rect.
    int  YDest,     // y-coordinate of upper-left corner of dest. rect.
    int  nDestWidth,  // width of destination rectangle
    int  nDestHeight, // height of destination rectangle
    int  XSrc,     // x-coordinate of upper-left corner of source rect.
    int  YSrc,     // y-coordinate of upper-left corner of source rect.
    int  nSrcWidth,   // width of source rectangle
```

```
    int  nSrcHeight,  // height of source rectangle
    VOID  *lpBits,    // address of bitmap bits
    BITMAPINFO *lpBitsInfo,   // address of bitmap data
    UINT  iUsage,  // usage
    DWORD  dwRop   // raster operation code
   )
{
    BITMAPINFOHEADER  bmiTemp;
    float fDestYDelta;
    LPBYTE lpNewBits;
    int i;

    // Check for NULL pointers and return error
    if (lpBits == NULL) return 0;
    if (lpBitsInfo == NULL) return 0;

    // Get increment value for Y axis of destination
    fDestYDelta = (float)nDestHeight / (float)nSrcHeight;

    // Make backup copy of BITMAPINFOHEADER
    bmiTemp = lpBitsInfo->bmiHeader;

   // Adjust image sizes for one scan line
    lpBitsInfo->bmiHeader.biSizeImage =
                      BYTESPERLINE(lpBitsInfo->bmiHeader.biWidth,
                                   lpBitsInfo->bmiHeader.biBitCount);
    lpBitsInfo->bmiHeader.biHeight = 1;

    // Initialize pointer to the image data
    lpNewBits = (LPBYTE)lpBits;

    // Do the stretching
    for (i = 0; i < nSrcHeight; i++)
   if (!StretchDIBits(hdc,
          XDest, YDest + (int)floor(fDestYDelta * (nSrcHeight - (i+1))),
          nDestWidth, (int)ceil(fDestYDelta),
          XSrc, 0,
          nSrcWidth, 1,
          lpNewBits, lpBitsInfo,
          iUsage, SRCCOPY))
        break; // Error!
   else
     // Increment image pointer by one scan line
     lpNewBits += lpBitsInfo->bmiHeader.biSizeImage;

    // Restore BITMAPINFOHEADER
    lpBitsInfo->bmiHeader = bmiTemp;

    return(i);
}
```

STATUS

This behavior is by design.

Additional reference words: 1.15 1.20 1.10
KBCategory: kbprg kbprb kbcode
KBSubcategory: W32s

PRB: LB_DIR with Long Filenames Returns LB_ERR in Windows 95

PSS ID Number: Q131286
Authored 07-Jun-1995 Last modified 10-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 3.51, 4.0
- Microsoft Win32s version 1.3

SYMPTOMS

Sending an LB_DIR message to a list box that specifies a long filename in the lParam returns LB_ERR in Windows 95 but works fine in Windows NT version 3.51.

CAUSE

The implementation of list boxes in Windows 95 thunks down to 16-bit USER.EXE, and the LB_DIR command has not been enhanced to support long filenames.

RESOLUTION

Convert the long filename to its short form before passing it as the lParam to LB_DIR by using GetShortPathName(). Similarly, when calling DlgDirList() to fill a list box with filenames, make sure the lpPathSpec parameter refers to the short name of the file.

Sample Code

```
char  szLong [256], szShort [256];
DWORD dwResult;
LONG  lResult;

lstrcpy (szLong, "C:\\This Is A Test Subdirectory");
dwResult = GetShortPathName (szLong, szShort, 256);
if (!dwResult)
   dwResult = GetLastError ();

lstrcat (szShort, "\lResult = SendDlgItemMessage (hdlg,
                              IDC_LIST1,
                              LB_DIR,
                              (WPARAM)(DDL_READWRITE),
          (LPARAM)(LPSTR)szShort);
if (LB_ERR == lResult)
   // an error occurred
```

NOTE: If a file with a long filename exists under the subdirectory specified, Windows 95 displays the short name in the list box, whereas

Windows NT displays the long name.

STATUS

This behavior is by design.

MORE INFORMATION

This is not a problem under Windows NT because it always supported long filenames.

You can have an application check the system version and decide at run time if it should call GetShortPathName before passing the filename as lParam to the LB_DIR message. Windows NT will, however, take a short name and fill the list box with the filenames.

Additional reference words: 4.00 1.30 LongFileName LFN DlgDirList CB_DIR DlgDirListComboBox
KBCategory: kbprg kbcode kbpb
KBSubcategory: UsrCtl

PRB: Listview Comes Up with No Images

PSS ID Number: Q125628
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)
- The Microsoft Foundation Classes (MFC), included with:
  Microsoft Visual C++, 32-bit Edition, version 2.1

SYMPTOMS

A list view is displayed with text and column headings, but the icons are not displayed.

CAUSE

The CImageList used to store the images for the list view is no longer in scope.

RESOLUTION

This can occur, for example, if you create a CImageList on the stack and create your listview, but at the point the listview is displayed, the image list has been destroyed. The ImageList functions will still return success, but no images will be displayed.

To avoid the problem, make sure your image list stays in scope.

STATUS

This behavior is by design.

Additional reference words: 4.00 Windows 95
KBCategory: kbprg
KBSubcategory: UsrCtl

PRB: LoadCursor() Fails on IDC_SIZE/IDC_ICON

PSS ID Number: Q131280
Authored 07-Jun-1995 Last modified 10-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SYMPTOMS

LoadCursor() returns NULL when passed IDC_SIZE or IDC_ICON for a second parameter.

CAUSE

IDC_SIZE and IDC_ICON are obsolete. They are available only for backward compatibility. Applications marked as a version 4.0 application are not able to load these cursors under Windows 95.

STATUS

This behavior is by design.

Additional reference words: 4.00
KBCategory: kbprg kbprb
KBSubcategory: UsrRsc

PRB: LoadLibrary() Fails with _declspec(thread)

PSS ID Number: Q118816
Authored 31-Jul-1994 Last modified 22-Jun-1995

The information in this article applies to:

- Microsoft Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0
- Microsoft Win32s versions 1.1, 1.15, 1.2, and 1.25a

SYMPTOMS

Your dynamic-link library (DLL) uses __declspec(thread) to allocate thread local storage (TLS). There are no problems running an application that is statically linked with the DLL's import library. However, when an application uses LoadLibrary() to load the DLL instead of using the import library, LoadLibrary() fails on Win32s with "error 87: invalid parameter". LoadLibrary() succeeds under Windows NT in this situation; however, the application cannot successfully call functions in the DLL.

CAUSE

This is a limitation of LoadLibrary() and __declspec(). The global variable space for a thread is allocated at run time. The size is based on a calculation of the requirements of the application plus the requirements of all of the DLLs that are statically linked. When you use LoadLibrary(), there is no way to extend this space to allow for the thread local variables declared with __declspec(thread). This can cause a protection fault either when the DLL is dynamically loaded or code references the data.

RESOLUTION

DLLs that use __declspec(thread) should not be loaded with LoadLibrary().

Use the TLS APIs, such as TlsAlloc(), in your DLL to allocate TLS if the DLL might be loaded with LoadLibrary(). If you continue to use _declspec(), warn users of the DLL that they should not load the DLL with LoadLibrary().

Additional reference words: 1.10 1.20 3.10 3.50 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: BseDll

PRB: Local Reboot (CTRL+ALT+DEL) Doesn't Work Under Win32s

PSS ID Number: Q121092
Authored 26-Sep-1994 Last modified 28-Nov-1994

The information in this article applies to:

- Microsoft Win32s, versions 1.1, 1.15, and 1.2

SYMPTOMS

The Windows local reboot feature (CTRL+ALT+DEL) should terminate a hung application. However, if the application is a Win32-based application, using CTRL+ALT+DEL exits Windows.

CAUSE

This is a limitation of Win32s. When the VxD that handles hung applications tries to access the application stack, it uses ss:sp, instead of ss:esp, as if the application were a 16-bit application. This causes the VxD to crash, and when the VxD crashes, the whole system terminates.

STATUS

This behavior is by design.

Additional reference words: 1.10 1.15 1.20
KBCategory: kbenv kbprb
KBSubCategory: W32s

PRB: MDI Program Menu Items Changed Unexpectedly

PSS ID Number: Q74789
Authored 30-Jul-1991 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SYMPTOMS

In an application for the Microsoft Windows graphical environment developed using the Windows multiple document interface (MDI), when the Window menu changes to indicate the addition of MDI child windows, another menu of the application is also changed.

For example, if the menu resembles the following before any windows are opened

```
   FILE            WINDOW
      Load            Cascade
      Save            Tile
      Save As...      Arrange Icons
      Exit
```

the menu might resemble the following after the first window is opened:

```
   FILE            WINDOW
      Load            Cascade
      1: MENU.TXT     Tile
      Save As...      Arrange Icons
      Exit            -------------
                      1: MENU.TXT
```

CAUSE

One or more menu items have menu-item identifiers that are greater than or equal to the value of the idFirstChild member of the CLIENTCREATESTRUCT data structure used to create the MDI client window.

RESOLUTION

Change the value of the idFirstChild member to be larger than any menu item identifiers.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: UsrMdi

PRB: Messages Sent to Mailslot Are Duplicated

PSS ID Number: Q127905
Authored 21-Mar-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.5, 3.51, and 4.0

SYMPTOMS

One application creates a mailslot using CreateMailSlot() and reads from it using ReadFile(). A second application opens the mailslot using CreateFile() and writes to it using WriteFile(). The second application writes one message to the mailslot, but the first application receives three duplicates of the message.

CAUSE

This is expected behavior if you have three network transports loaded. There is no way to know which transport should be used to deliver to a given mailslot on a remote machine, so all transports are used.

RESOLUTION

Send a unique ID at the beginning of each message. The listening end can detect duplicates and delete them. If you have multiple clients sending messages, their messages may be interleaved in the mailslot. You may need to track which client sent which message last, in order to successfully detect duplicates.

STATUS

This behavior is by design.

Additional reference words: 3.50 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: BseIpc

PRB: Most Common Cause of SetPixelFormat() Failure

PSS ID Number: Q126019
Authored 12-Feb-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.5, 3.51, and 4.0

SYMPTOMS

SetPixelFormat() will fail with incorrect class or window styles.

CAUSE

Win32-based applications that use Microsoft's implementation of OpenGL to render onto a window must include WS_CLIPCHILDREN and WS_CLIPSIBLINGS window styles for that window.

RESOLUTION

Include WS_CLIPCHILDREN and WS_CLIPSIBLINGS window styles when in a Win32-based application, you use Microsoft's implementation of OpenGL to render onto a window.

Additionally, the window class attribute should not include the CS_PARENTDC style. The two window styles can be added to the dwStyles parameter of CreateWindow() or CreateWindowEX() call. If MFC is used, override PreCreateWindow() to add the flags. For example:

```
BOOL CMyView::PreCreateWindow(CREATESTRUCT& cs)
{
    cs.style |= (WS_CLIPCHILDREN | WS_CLIPSIBLINGS);

    return CView::PreCreateWindow(cs);
}
```

For more information, please refer to "comments" section of the online documentation on SetPixelFormat.

STATUS

This behavior is by design.

Additional reference words: 3.50 4.00 95
KBCategory: kbgraphic kbprb
KBSubcategory: GdiOpenGL

PRB: Moving or Resizing the Parent of an Open Combo Box

PSS ID Number: Q76365
Authored 23-Sep-1991 Last modified 16-May-1995

-
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

-

SYMPTOMS

When the user resizes or moves the parent window of an open drop-down combo box, the list box portion of the combo box does not move.

CAUSE

The list box portion of the combo box does not receive a move message. Therefore, it remains on the screen at its original position.

RESOLUTION

Close the drop down list before the combo box is moved. To perform this task, during the processing of the WM_PAINT message, send the combo box a CB_SHOWDROPDOWN message with the wParam set to FALSE.

Additional reference words: 3.00 3.10 3.50 4.00 95 combobox
KBCategory: kbprg kbprb
KBSubcategory: UsrCtl

PRB: MS-SETUP Uses \SYSTEM Rather Than \SYSTEM32

PSS ID Number: Q98888
Authored 18-May-1993 Last modified 29-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SYMPTOMS

Call GetWindowsSysDir() in the SETUP.MST file of a 16-bit setup application. The return value is C:\WINNT\SYSTEM\ instead of C:\WINNT\SYSTEM32\. Note that this doesn't happen with the 32-bit Setup Toolkit.

CAUSE

Windows on Win32 (WOW) returns the SYSTEM directory, not the SYSTEM32 directory, to 16-bit applications such as MS-SETUP. This is done for compatibility reasons.

RESOLUTION

Determine whether the setup code is being run under WOW or Windows version 3.1 by checking the WF_WINNT bit (0x4000) in the return from GetWinFlags(). Choose either the return from GetWindowsSysDir() or <winows dir>\system32 as appropriate.

MORE INFORMATION

Note that there are additional considerations for network installs for Win32s, because the SYSTEM directory may not be a branch off of the Windows directory.

Additional reference words: 3.10 3.50
KBCategory: kbtool kbprb
KBSubcategory: TlsMss

PRB: Named Pipe Write() Limited to 64K

PSS ID Number: Q119218
Authored 10-Aug-1994 Last modified 28-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SYMPTOMS

WriteFile() returns FALSE and GetLastError() returns ERROR_MORE_DATA when WriteFile() writes to a message-mode named pipe using a buffer greater than 64K.

CAUSE

There is a 64K limit on named pipe writes.

RESOLUTION

The error is different from ERROR_MORE_DATA on the reader side, where bytes have already been read and the operation should be retried for the remaining message. The real error is STATUS_BUFFER_OVERFLOW. No data is transmitted; therefore, the write operation must be retried using a smaller buffer.

Additional reference words: 3.10 3.50
KBCategory: kbprg kbprb
KBSubcategory: BseIpc

PRB: NetBIOS Command NCBSEND Gets Return Code Error 0x3C

PSS ID Number: Q123457
Authored 01-Dec-1994 Last modified 03-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SYMPTOMS

While opening sessions, you start to get error 0x3C (lock of user area failed) in the NcbSend command call.

CAUSE

An application in Windows NT uses the NcbListen command to accept NetBIOS calls. After a call is received, the application uses the NcbSend command to send data back.

If the computer running Windows NT has 32 megabytes of main memory, an application can request a large number (for example, 128) of sessions, by using the NcbReset command, without difficulty.

However, with only 16 megabytes of main memory, an application can request only a moderate number (for example, 80) of sessions. If more sessions are opened they start to get error 0x3C (lock of user area failed) in the NcbSend command call. The error persists until some of the sessions that are currently open are closed, at which time NcbSend will get a good return status.

STATUS

This behavior is by design. The system stops the process from using so many resources that it jeopardizes the performance of other applications and/or the system itself.

Additional reference words: 3.10 3.50
KBCategory: kbprg kbenv kbprb
KBSubcategory: NtwkNetbios

PRB: Netbios RESET Cannot Be Called with Pending Commands

PSS ID Number: Q125659
Authored 01-Feb-1995 Last modified 07-Mar-1995

The information in this article applies to:

- Microsoft Win32s, versions 1.2 and 1.25

SYMPTOMS

A Win32-based application cannot call the Netbios RESET command as long as there are asynchronous commands still pending.

CAUSE

This behavior is by design in Win32s.

RESOLUTION

The application should cancel all pending commands, and wait for the post routines of all the pending commands to be called. After that, the application can issue a Netbios RESET command.

STATUS

This behavior is by design but may be designed differently in a future version of Win32s.

Additional reference words: 1.20 1.25
KBCategory: kbprg
KBSubcategory: W32s

PRB: NetDDE Fails to Connect Under Windows 95

PSS ID Number: Q131025
Authored 02-Jun-1995 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SYMPTOMS

An application using Network Dynamic Data Exchange (NetDDE) fails to connect to another DDE application on another computer.

CAUSE

One of the reasons could be that NETDDE.EXE (a system component) is not running.

Network Dynamic Data Exchange (NetDDE) allows two DDE applications to communicate with each other over the network. In Windows for Workgroups, the NETDDE.EXE (a system component) was loaded by default. However under Window 95, NETDDE.EXE is not loaded by default.

RESOLUTION

An application using the netDDE services should check if the netDDE system component is loade. if NETDDE.EXE isn't running, the application should run it.

Sample Code

The following sample code checks to see if NETDDE.EXE is loaded and tries to load it if necessary. The sample code works for both 32-bit and 16-bit applications.

```
BOOL IsNetDdeActive()
{
HWND  hwndNetDDE;

// find a netDDE window
hwndNetDDE = FindWindow("NetDDEMainWdw", NULL);
// if exists then NETDDE.EXE is running
if(NULL == hwndNetDDE)
   {
   UINT  uReturn;
   // otherwise launch the NETDDE.EXE with show no active
   uReturn = WinExec("NETDDE.EXE", SW_SHOWNA);
   // if unsucessful return FALSE.
   if(uReturn <= 31)
```

```
      return FALSE;
   }
// NetDDE is running
return TRUE;
}
```

STATUS

This behavior is by design.

Additional reference words: 4.00
KBCategory: kbprg kbnetwork kbcode kbinterop
KBSubcategory: Usrdde

PRB: Number Causes Help Compiler Invalid Context ID Error

PSS ID Number: Q85490
Authored 10-Jun-1992 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SYMPTOMS

When the Windows Help Compiler compiles a help project file, the following error message appears:

```
Error P1083: Invalid context identification number
```

CAUSE

The representation of the context identification number begins with a zero and contains the digit 8 or 9.

RESOLUTION

Remove the leading zeros from the number.

MORE INFORMATION

The Windows Help Compiler parses a number that has a leading zero as an octal number. C compilers also interpret numbers in this manner. Only the digits 0 through 7 are legal in an octal number.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool kbprg kbprb
KBSubcategory: TlsHlp

PRB: Oracle7 for Win32s Hangs When Initialize Database

PSS ID Number: Q127760
Authored 15-Mar-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32s versions 1.00, 1.10, 1.15, 1.20, and 1.25a

SYMPTOMS

There is a known problem with using Oracle7 for Win32s with the Windows Sound System version 2.0. The problem occurs when starting up a database in the Database Manager. Oracle7 may hang during this process. The machine will need to be rebooted.

CAUSE

The SNDEVNTS.DRV file is causing the problem by performing stack checking on the application stacks of Oracle7. The stack checking is corrupting the application stack of Oracle7 causing the application to hang.

RESOLUTION

Download SEVNT022.EXE, a self extracting file from the Microsoft Software Library (MSL) available on the following services:

- CompuServe
  GO MSL
  Search for SEVNT022.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download SEVNT022.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the SOFTLIB\MSLFILES directory
  Get SEVNT022.EXE

Follow the installation directions contained in the README.TXT file.

Additional reference words: 1.00 1.10 1.20 SNDEVNTS.DRV
KBCategory: kb3rdparty kbfile kbprb
KBSubcategory: W32s

PRB: Page Fault in WIN32S16.DLL Under Win32s

PSS ID Number: Q115082
Authored 18-May-1994 Last modified 21-Feb-1995

The information in this article applies to:

- Microsoft Win32s, versions 1.1 and 1.15

SYMPTOMS

Start two instances of the Win32 application under Win32s. Close one instance, then close the other instance. A page fault is generated in WIN32S16.DLL. Under Win32s, version 1.1, a dialog box appears. Under Win32s, version 1.15, the error only appears in the output from the debug version of Win32s.

CAUSE

This problem can be caused by using the static C run-time (CRT) libraries or using MSVCRT10.DLL. Note that even if the application does not make any CRT calls, one of its DLLs may call the CRT initialization and cleanup code. The CRT code makes the following sequence of API calls:

```
DeleteCriticalSection
DeleteCriticalSection
DeleteCriticalSection
TlsFree
VirtualFree
VirtualFree
VirtualFree
```

When the second application instance terminates, it faults before it makes the call to TlsFree(). The CRT has two blocks, one that contains strings from the environment and one that contains pointers to the first blocks. These are allocated by the first process that attach to the DLL. Other processes that attach do not allocate these blocks. When processes are terminated, these two blocks are freed. However, this succeeds only when the process that owns the memory frees them. Any other process that tries to access these blocks will fail.

RESOLUTION

Because there is no instance data by default under Win32s, DLLs should use the CRT in a DLL instead of linking to the CRT statically. MSVCRT10.DLL (which comes with Visual C++) is not compatible with Win32s because MSVCRT10.DLL falsely assumes that Win32s implements instance data, which is only available on Windows NT. Therefore, until an updated MSVCRT.LIB file is released, use CRTDLL.LIB (which comes with the Win32 SDK) because Win32s has its own CRTDLL.DLL file that was specifically designed for this use.

Microsoft Visual C++ 2.0 contains two versions of MSVCRT20.DLL: one version is intended for use on Windows NT, the other is intended for use on Win32s. To avoid this problem, use and ship the Win32s version of MSVCRT20.DLL in your application.

MORE INFORMATION

The real problem is that memory is allocated for several applications. The allocation is done by the first application. When this application terminates, it takes with it all of its memory. This means that each time the remaining applications try to access this memory, an error occurs. Symptoms include data corruption, hanging, or the WIN32S16.DLL page fault mentioned above.

Additional reference words: 1.10 1.15
KBCategory: kbprg kbprb
KBSubcategory: W32s

PRB: PaintRgn() Fills Incorrectly with Hatched Brushes

PSS ID Number: Q82169
Authored 29-Mar-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, version 4.0

SYMPTOMS

When the TRANSPARENT background mode and a mapping mode other than MM_TEXT are selected and an application calls the PaintRgn() API to fill a complex region with a hatched brush, a disconnected pattern results.

CAUSE

The Windows Graphics Device Interface (GDI) draws a complex region by filling the individual rectangles that make up the region. The code to compute the position of each rectangle on the screen fails when the screen coordinates are not in units of pixels. The error is visible when a hatched brush style is used in TRANSPARENT mode.

RESOLUTION

When a hatched brush and TRANSPARENT background mode are required, use the MM_TEXT mapping mode.

Additional reference words: 3.10 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: GdiDrw

PRB: Poor TCP/IP Performance When Doing Small Sends

PSS ID Number: Q126716
Authored 28-Feb-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SYMPTOMS

When doing multiple sends of less than the Maximum Transmission Unit (MTU), you may see poor performance. On an Ethernet network, the MTU for TCP/IP is 1460 bytes.

CAUSE

When an application does two sends of less than a transport MTU, the second send is delayed until an ACK is received from the remote host. The delay occurs in case the application does another small send. TCP can then coalesce the two small sends into one larger packet. This concept of collecting small sends into larger packets is called Nagling.

RESOLUTION

There are a number of ways to avoid Nagling in an application. Here are two. The second is more complex but gives a better performance benefit:

- Set the TCP_NODELAY socket option for the socket. This tells TCP/IP to send always, regardless of packet size. This will result in sub-optimal use of the physical network, but it will avoid the delay of waiting for an ACK.

- Send larger blocks of data. The send() API call, when you include the overhead of the other network components involved, costs a couple of thousand instructions. One large send() call will be more efficient than two smaller send() calls, even if you need to do some buffer copies.

  Sending larger data blocks will also result in more efficient use of the physical network because packets will typically be larger and less numerous. This option is much better than the first (enabling TCP_NODELAY) and should be used if at all possible.

On Windows NT 3.51, if you are sending files, you should use the new TransmitFile() API. This call reads the file data directly from the file system cache and sends it out over the wire. The TransmitFile() call can also take a data block that will be sent ahead of the file, if desired.

REFERENCES

More information about Nagling and the Nagle algorithm can be found in RFC 1122.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg kbnetwork kbprb
KBSubcategory: NtwkWinsock

PRB: Pressing the ENTER Key in an MDI Application

PSS ID Number: Q99799
Authored 08-Jun-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SYMPTOMS

In a standard Microsoft Windows version 3.1 multiple document interface (MDI) application, when a minimized MDI child is active and the user presses the ENTER key, the child is not restored.

This is inconsistent with other MDI applications, such as File Manager and Program Manager. An MDI child in one of these applications is restored when it is the active MDI child and the ENTER key is pressed. When a normal Windows-based application is minimized and the user presses ENTER, that application is restored to a normal state.

RESOLUTION

One quick workaround to this problem is to create an accelerator for the ENTER key and restore the minimized MDI child when the key is pressed.

MORE INFORMATION

It may be desirable to implement the same restore feature that File Manager and Program Manager have implemented in order to enable the user to restore an MDI child by pressing the ENTER key. If this feature is implemented, then the MDI application can be consistent with other popular applications such as File Manager, Microsoft Excel, and Microsoft Word.

To achieve this effect in an MDI application, create an accelerator in the accelerator table of the resource file for the application. This can be done as follows:

```
MdiAccelTable ACCELERATORS
     {
     .  .  .
     .  .  .
     VK_RETURN, IDM_RESTORE, VIRTKEY
     }
```

After this accelerator has been installed in the MDI application, each time the ENTER key is pressed by the user, an IDM_RESTORE command will be sent to the MDI frame window's window procedure through a

WM_COMMAND message. When the MDI frame receieves this message, its window procedure should retrieve a handle to the active MDI child and determine if it is minimized. If it is minimized, then it can restore the MDI child by sending the MDI client a WM_MDIRESTORE message. This can all be done with the following code:

```
        case IDM_RESTORE:
           {
           HWND hwndActive;

           hwndActive = SendMessage(hwndClient,WM_MDIGETACTIVE,0,0L);
           if (IsIconic(hwndActive))
              SendMessage(hwndClient,WM_MDIRESTORE,hwndActive,0L);
           break;
           }
```

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbprb kbcode
KBSubcategory: UsrMdi

PRB: Printer Font too Small with ChooseFont() Common Dialog

PSS ID Number: Q89544
Authored 24-Sep-1992 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SYMPTOMS

In an application for the Microsoft Windows graphical environment, the user selects a printer font through the ChooseFont() common dialog box function. When the application calls the CreateFontIndirect() to create the selected font, even though the style and face name are correct, the point size is much too small.

CAUSE

The LOGFONT structure returned to the application is based on screen metrics--even when the user selects a printer font. Because the resolution (dots per inch) on a printer is generally much greater than that of the screen, the resulting printer font is smaller than desired.

RESOLUTION

Modify the lfHeight member of the LOGFONT data structure according to the printer metrics.

MORE INFORMATION

The following code demonstrates how to modify the lfHeight member. The lpcf variable contains a pointer to the CHOOSEFONT data structure. The hDC member of the CHOOSEFONT data structure is a handle to the printer device context.

```
   if (ChooseFont(lpcf))
   {
      if (lpcf->nFontType & PRINTER_FONT)
      {
         iLogPixelsy = GetDeviceCaps(lpcf->hDC, LOGPIXELSY);
         lpcf->lpLogFont->lfHeight =
             MulDiv(-iLogPixelsy, (lpcf->iPointSize / 10), 72);
         hPrinterFont =
             CreateFontIndirect((LPLOGFONT)(lpcf->lpLogFont));
      }
      else
      {
         // Create screen font
```

```
      }
   }
   else
   {
      // Process common dialog box error
   }
```

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbprb kbcode
KBSubcategory: UsrCmnDlg

PRB: Private Button Class Can't Get BM_SETSTYLE in Windows 95

PSS ID Number: Q130951
Authored 31-May-1995 Last modified 02-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SYMPTOMS

When an application creates a new button class, the new buttons do not receive BM_SETSTYLE messages under Windows 95.

CAUSE

In previous versions of Windows, the new button had only to return DLGC_BUTTON in response to the WM_GETDLGCODE message. This was all that was required to identify the window as a "button."

However, in Windows 95, returning DLGC_BUTTON to WM_GETDLGCODE is no longer sufficient to identify the window as a "button." The dialog manager code in Windows 95 is implemented in 16 bits. When a message is dispatched to a 32-bit window, the system automatically generates a thunk. Because the system does not know that the new class is actually a "button," it does not automatically perform the thunk - so the BM_SETSTYLE messages are not sent.

RESOLUTION

To tell the system to treat the window as a "button," the window must call one of the following APIs at least once:

IsDlgButtonChecked
CheckRadioButton
CheckDlgButton

The preferable method for doing this is to call IsDlgButtonChecked during the WM_CREATE message. Once this is done, the window will receive all standard button messages.

STATUS

This behavior is by design.

Additional reference words: 4.00
KBCategory: kbprg kbui kbprb
KBSubcategory: UsrCtl

PRB: Problems with the Microsoft Setup Toolkit

PSS ID Number: Q106382
Authored 07-Nov-1993 Last modified 29-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SYMPTOMS

1. When the /zi option is used with the Win32 DSKLAYT2.EXE to provide compression, it causes an access violation.

2. The Win32 Setup Toolkit does not contain a setup bootstrapper to copy the needed setup files to a temp drive and run the Setup program. Setup runs from floppy disks.

3. Install programs for applications that may run on Win32s must be created with the 16-bit version of the Setup Toolkit if the installation program will install Win32s. However, the 16-bit DSKLAY2.EXE cannot read the version information in a Win32 binary.

4. The Win32 DSKLAYT.EXE only shows 8.3 names in the list box.

5. The Setup program reports "out of memory" during installation, but there seems to be plenty of memory.

6. Installation fails from a CD-ROM. If the same files are copied from the CD to the hard disk, installation succeeds.

7. Setup programs created with the 32-bit setup toolkit will not run under Win32s.

RESOLUTIONS

1. The fix for this problem is available in the Alpha SDK Update and later.

   Note that COMPRESS.EXE has been updated to use a better compression algorithm, and therefore /zi is no longer recommended for best compression. The option has been kept for compatibility reasons.

2. The bootstrapper is not necessary in a 32-bit environment. It is required for Windows because it is not possible to remove the floppy disk of a currently running Win16 application (the resources could not all be preloaded and locked). If you want to use a bootstrapper for compatibility, a 32-bit version is available on CompuServe.

3. If a Win32s installation is provided on a separate disk, the install program can be developed with the Win32 Setup Toolkit.

4. The program is actually a 16-bit program, and therefore it can display only the 8.3 name. Use 8.3 names for the source names and specify that the files be renamed (using the long names) when they are installed.

5. This error can be caused when a DLL on disk 1 is needed when when a different disk is currently inserted. To work around this problem, use LoadLibrary() to load the DLL.

6. This problem was corrected in the Win32 3.5 SDK.

7. This problem was corrected in the Win32 3.5 SDK.

Additional reference words: 3.10 3.50
KBCategory: kbtool kbprb
KBSubcategory: TlsMss

PRB: Processing the WM_QUERYOPEN Message in an MDI

PSS ID Number: Q99411
Authored 27-May-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SYMPTOMS

When a multiple document interface (MDI) child window processes the WM_QUERYOPEN message to prevent the child from being restored out of a minimized state, the system menu and restore button are not removed from the menu bar when a maximized MDI child loses the focus or is closed.

This problem occurs only when the maximized MDI child loses the focus or is closed and the focus is given to an MDI child that returns 0 (zero) on the WM_QUERYOPEN message.

CAUSE

When an MDI child is maximized, the system menu and restore button are added to the frame menu as bitmap menu items. When a maximized MDI child is destroyed or another MDI child is given the focus, the MDI child given the focus afterwards is maximized to replace the old MDI child. Windows cannot maximize an MDI child when it is processing the WM_QUERYOPEN message, and therefore the child is not maximized. Unfortunately, the system menu and restore button bitmaps are not removed from the menu bar.

RESOLUTION

To prevent this problem, restore the maximized MDI child before giving the focus to another child.

MORE INFORMATION

It may sometimes be desirable to prevent an MDI child from being restored during part or all of its life. This can be done by trapping the WM_QUERYOPEN message by placing the following code in the window procedure of the MDI child:

```
   case WM_QUERYOPEN:
      return 0;
```

Unfortunately, this causes the added restore and system menu bitmaps to remain on the menu bar when a maximized MDI child loses the focus

or is closed and the focus is given to a child processing this message. The following code can be used to restore a maximized MDI child when it loses the focus:

```
     case WM_MDIACTIVATE:
        if ((wParam == FALSE) && (IsZoomed(hwnd)))
           SendMessage(hwndMDIClient, WM_MDIRESTORE, hwnd, 0L);

        return DefMDIChildProc (hwnd, msg, wParam, lParam);
```

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbprb kbcode
KBSubcategory: UsrMdi

PRB: Property Sheet w/ Multiline Edit Control Ignores ESC Key

PSS ID Number: Q130765
Authored 25-May-1995 Last modified 22-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SYMPTOMS

Pressing the ESC key when the focus is on a Multiline Edit control that is a child of a property sheet page, does not dismiss the property sheet control as expected.

CAUSE

When the ESC key is pressed while focus is on a Mutiline Edit control, the IDCANCEL notification (sent with a WM_COMMAND message) is sent to the property sheet dialog proc whose template contains the Multiline Edit control. Property sheet dialog procs do not process this message, so it is not forwarded to the property sheet control.

RESOLUTION

Trap the IDCANCEL notification that is sent along with the WM_COMMAND message in the property sheet dialog proc that contains the multiline edit control. Then forward the message to the property sheet control. (The property sheet control is the parent of all the property sheet page dialogs.) The following code shows how to do this:

Code Sample

```
//
//  FUNCTION: SheetDialogProc(HWND, UINT, WPARAM, LPARAM)
//
//  PURPOSE:  Processes messages for a page in the PPT sheet control.
//
//  PARAMETERS:
//    hdlg - window handle of the property sheet
//    wMessage - type of message
//    wparam - message-specific information
//    lparam - message-specific information
//
//  RETURN VALUE:
//    TRUE -  message handled
//    FALSE - message not handled
//

LRESULT CALLBACK SheetDialogProc(HWND hdlg,
```

```
                      UINT uMessage,
                     WPARAM wparam,
                     LPARAM lparam)
{
    LPNMHDR lpnmhdr;
    HWND    hwndPropSheet;
    switch (uMessage)
    {
   case WM_INITDIALOG:

      // Do whatever initializations you have here.
      return TRUE;


   case WM_NOTIFY:

      // more code here ...

      break;


    case WM_COMMAND:

       switch(LOWORD(wparam))
      {

         case IDCANCEL:

         // Forward this message to the parent window
         // so that the PPT sheet is dismissed
            SendMessage(GetParent(hdlg),
                        uMessage,
                        wparam,
                        lparam);
         beak;

         default:
         break;
      }
      break;

   }
```

NOTE: This solution also works with wizard controls. This behavior is seen under both Windows NT and Windows 95; the solution in this article works for both platforms.

Additional reference words: 4.00 user styles
KBCategory: kbprg kbprb kbcode
KBSubcategory: UsrCtl

PRB: Quotation Marks Missing from Compiled Help File

PSS ID Number: Q110540
Authored 24-Jan-1994 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SYMPTOMS

After upgrading from Word for Windows 2.x to Word for Windows 6.x and rebuilding a Windows Help file with HCP.EXE or HC31.EXE, all of the quotation marks are missing from the Help file.

CAUSE

Word 6.x uses a special .RTF keyword to represent the quotation mark character, and the Help compiler does not understand the new .RTF keyword so it drops the character. The same problem occurs for the curly double quotation mark, single quotation mark, en dash (char 150), em dash (char 151), and the bullet character (char 149).

RESOLUTION

You can prevent this problem by turning off the "Smart Quotes" option in Word for Windows. The following three steps accomplish this:

1. Choose Options from Tools menu.
2. Select the AutoFormat tab.
3. In the Replace group box, clear the "Straight Quotes with Smart Quotes" check box, and choose OK.

If you wish to include smark quotes, bullets, em-dashes, and en-dashes in a Help file, you can open the file as text only and replace the RTF keywords with their ANSI hexidecimal equivalents, which are recognized by the help compiler.

Find String	Replace String
\emdash	\'97"
\endash	\'96"
\bullet	\'95"
\rdblquote	\'94"
\ldblquote	\'93"
\rquote	\'92"
\lquote	\'91"

Make sure that there is a blank character included at the end of the Find

String, but not in the Replace String.

This replacement must be made every time you edit and convert the text from document to RTF format.

Additional reference words: 3.10 3.50 4.00
KBCategory: kbtool kbprg kbprb
KBSubcategory: TlsHlp

PRB: RegCreateKeyEx() Gives Error 161 Under Windows NT 3.5

PSS ID Number: Q117261
Authored 23-Jun-1994 Last modified 28-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5

SYMPTOMS

A call to RegCreateKeyEx() is successful under Windows NT version 3.1, but the call fails with error 161 (ERROR_BAD_PATHNAME) under Windows NT version 3.5.

The sample code below demonstrates this problem.

CAUSE

This is by design. Windows NT version 3.1 allows the subkey to begin with a backslash ("\"), however Windows NT version 3.5 does not. The subkey is given as the second parameter to RegCreateKeyEx().

RESOLUTION

Remove the backslash from the beginning of the subkey name.

MORE INFORMATION

In the sample code below, RegCreateKeyEx() fails with error 161 while the string defined by SUBKEY_FORMAT_STRING begins with a backslash, but succeeds if the initial backslash is removed.

Sample Code

```
#include <windows.h>
#include <stdio.h>

#define SUBKEY_FORMAT_STRING \
"\\SYSTEM\\CurrentControlSet\\Services\\EventLog\\Application\\%s"

void main(int argc, char *argv[])
{
   DWORD dwErrorCode;
   char lpszSubKey[MAX_PATH];
   HKEY hKey;
   DWORD dwDisposition;
```

```
      sprintf( lpszSubKey, SUBKEY_FORMAT_STRING, argv[1] );

      printf( "Trying to open: %s\n", lpszSubKey );

      dwErrorCode = RegCreateKeyEx(HKEY_LOCAL_MACHINE,
                                   lpszSubKey,
                                   0,
                                   "",
                                   REG_OPTION_NON_VOLATILE,
                                   KEY_ALL_ACCESS,
                                   NULL, //Security
                                   &hKey,
                                   &dwDisposition );

      if (dwErrorCode != ERROR_SUCCESS)
         printf( "Code = %d.\n", dwErrorCode );

      RegCloseKey(hKey);
   }
```

NOTE: Double backslashes ("\\") are required in strings in C code to represent a single backslash, since a backslash ordinarily indicates the beginning of an escape sequence.

Additional reference words: 3.50
KBCategory: kbprg kbprb
KBSubcategory: BseMisc

PRB: RegisterClass()/ClassEx() Fails If cbWndExtra > 40 Bytes

PSS ID Number: Q131288
Authored 07-Jun-1995 Last modified 10-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SYMPTOMS

Under Windows 95, a call to RegisterClass() or RegisterClassEx() returns NULL if a value greater than 40 is specified for the cbWndExtra or cbClsExtra members of the WNDCLASS or WNDCLASSEX structure.

CAUSE

Windows 95 checks to see if cbWndExtra or cbClsExtra is greater than 40. If so, it outputs these debug messages and returns NULL to indicate failure:

```
RegisterClassEx: Unusually large cbClsExtra (>40)
RegisterClassEx: Unusually large cbWndExtra (>40)
```

RESOLUTION

If more than 40 bytes are needed to store window-specific or class-specific information, an application should allocate memory. Then set the cbWndExtra or cbClsExtra to 4 bytes, and pass the pointer to the allocated memory by using SetClassLong() as follows:

```
SetClassLong (hWnd, GCL_CBCLSEXTRA, lpMemoryAllocated);
SetClassLong (hWnd, GCL_CBWNDEXTRA, lpMemoryAllocated);
```

The pointer can then be retrieved when needed by using GetClassLong() as follows:

```
lpMemoryAllocated = GetClassLong (hWnd, GCL_CBCLSEXTRA);
lpMemoryAllocated = GetClassLong (hWnd, GCL_CBWNDEXTRA);
```

STATUS

This behavior is by design. However, as of version 3.51, Windows NT does not have this 40-byte limitation.

Additional reference words: 4.00
KBCategory: kbprg kbprb
KBSubcategory: UsrCls

PRB: Result of localtime() Differs on Win32s and Windows NT

PSS ID Number: Q117893
Authored 13-Jul-1994 Last modified 12-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.1, 1.15, and 1.2

SYMPTOMS

Under Windows NT, localtime(time()) returns the correct local time. However, under Win32s, the local time returned is not correct if the TZ environment variable is set. For example, suppose that you are in the Pacific time zone (GMT-08:00) and have set tz=pst8pdt. The time returned under Win32s is off by 8 hours.

CAUSE

This is by design.

The localtime() function depends on time zone information, which is not available in Win32s. This is the reason that the Win32 API GetLocalTime() is not supported under Win32s. The C Run-time functions, like localtime(), use the tz environment variable for time zone information.

The time() function returns the current local time under Win32s, then the call to localtime() adjusts the time by the offset of your time zone from GMT, which it finds by reading the tz environment variable.

Under Windows NT, time() and GetSystemTime() return GMT, therefore localtime(time()) is the current local time.

RESOLUTION

To get the current local time under both Win32s and Windows NT, use the following code to clear the tz environment variable and get the time:

```
_putenv( "TZ=" );
_tzset();

localtime( time() );
```

Note that _putenv() affects only the tz environment variable for the application. All other applications use the global environment settings and make their own modifications.

Additional reference words: 1.10 1.20
KBCategory: kbprg kbprb
KBSubcategory: W32s

PRB: RichEdit Control Hides Mouse Pointer (Cursor)

PSS ID Number: Q131381
Authored 11-Jun-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.51, 4.0
- Microsoft Win32s version 1.3

SYMPTOMS

When the mouse pointer (cursor) is over a RichEdit control and the user starts typing in the control, the RichEdit control hides the pointer.

CAUSE

When the user is typing in a RichEdit control, the control automatically hides the mouse pointer if it is is over the control. The control does this intentionally so the cursor does not obscure the text in the control. When the mouse is moved again, the pointer re-appears.

STATUS

This behavior is by design.

Additional reference words: 4.00 95 1.30
KBCategory: kbprg kbui kbprb
KBSubcategory: UsrCtl

PRB: RoundRect() and Ellipse() Don't Match Same Shaped

PSS ID Number: Q119455
Authored 16-Aug-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows, version 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SYMPTOMS

When CreateRoundRectrRgn() is used to create a region with the shape of a rectangle that has rounded corners and RoundRect() is called with the same parameters to draw the same rectangle that has rounded corners, the calculated region does not match the drawn rectangle. The same can be said of the ellipses created by CreateEllipticRgn() and Ellipse().

CAUSE

This behavior is because of the design on Windows. The mismatch between fills and frames is because of the way that the boundaries and fills must be specified in order to get polygons to fit together properly. Windows NT duplicates this behavior for compatibility.

RESOLUTION

Perform the fill first, then draw the frame. Some of the frame pixels will overwrite fill pixels and some will not; however, there will be no gap between the frame and the fill, and the fill will not extend past the frame. Use CreateRoundRectRgn() or CreateEllipticRgn() for the fill and RoundRect() or Ellipse, respectively, for the frame. Use the same parameters for both the region API and the filled-shape API.

NOTE: If you use a NULL pen when drawing the filled shape, the pixels will match those drawn by creating a region through the corresponding region API and then calling FillRgn() with the same parameters. It draws the frame with the pen from the filled-shape API that causes the discrepancy.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: GdiDrw

PRB: RPC Installation Problem

PSS ID Number: Q104315
Authored 13-Sep-1993 Last modified 28-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SYMPTOMS

There is a problem with the Remote Procedure Call (RPC) service installation. Setup searches the Windows directory for WINSOCK.DLL. If the file is found, Setup installs RPC16C3.DLL.

This is a problem if you are dual booting a machine between Windows 3.1 and Windows NT because Windows 3.1 uses an older TCP/IP interface, which is called RPC16C3X.DLL on the distribution disks. When you run Windows NT, you will want to use the newer TCP/IP interface, called RPC16C3.DLL.

WORKAROUND

If you are dual booting and using RPC, a workaround to this problem is to rename your WINSOCK.DLL file to something else, such as WINSOCK.XXX. This will cause Setup to copy the correct version of the TCP/IP dynamic-link library (DLL).

Additional reference words: 3.10
KBCategory: kbprg kbprb
KBSubcategory: NtwkRpc

PRB: RW1004 Error Due to Unexpected End of File (EOF)

PSS ID Number: Q106064
Authored 31-Oct-1993 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SYMPTOMS

The resource compiler generates the following errors when the .RC file includes a .H file whose last line is a define (that is, there was no final carriage return at the end of the #define statement):

```
   fatal error RC1004: unexpected EOF
```

CAUSE

The resource compiler preprocessor follows C syntax. A newline character is required on a #define statement.

RESOLUTION

Add a carriage return following the #define.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool kbprb
KBSubcategory: TlsRc

PRB: Saving/Loading Bitmaps in .DIB Format on MIPS

PSS ID Number: Q85844
Authored 22-Jun-1992 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, and 3.51

SYMPTOMS

In Win32, saving or loading a bitmap in .DIB file format is basically the same as in Win16. However, care must be taken in DWORD alignment, especially on the MIPS platform.

An exception occurs when loading or saving a bitmap on the MIPS platform. In NTSD, the following error message is received:

```
data mis-alignment
```

CAUSE

A non-DWORD aligned actual parameter was passed to a function such as GetDIBits().

The .DIB file format contains the BITMAPFILEHEADER followed immediately by the BITMAPINFOHEADER. Notice that the BITMAPFILEHEADER is not DWORD aligned. Thus, the structure that follows it, the BITMAPINFOHEADER, is not on a DWORD boundary. If a pointer to this DWORD misaligned structure is passed to the sixth argument of GetDIBits(), an exception will occur.

RESOLUTION

To resolve this problem, copy the data in the structure over to a DWORD-aligned memory and pass the pointer to the latter structure to the function instead. See the sample code LOADBMP.C for detail.

MORE INFORMATION

The is a sample to illustrates this process. Refer to the LOADBMP.C file in the MANDEL sample that comes with the Win32 SDK.

Additional reference words: 3.10 3.50
KBCategory: kbprg kbprb
KBSubcategory: GdiBmp

PRB: Search Button Disabled in Windows Help

PSS ID Number: Q71761
Authored 01-May-1991 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SYMPTOMS

When a help file is loaded into the Windows Help program, the Search button is disabled.

CAUSE

There are two likely causes:

1. The help file defines no keywords for searching.

2. The keywords are defined using a lowercase "k" footnote.

RESOLUTION

For cause 1, if searching is desired, define some keywords in the file. For cause 2, modify the RTF text to use an uppercase "K" for keyword footnotes.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbtool kbprg kbprb
KBSubcategory: TlsHlp

PRB: SEH with Abort() in the try Body

PSS ID Number: Q91146
Authored 29-Oct-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SYMPTOMS

When using Structured Exception Handling, if the try body calls abort(), the finally body is not executed.

CAUSE

The finally body is not executed because the abort() never returns. It calls ExitProcess(), which terminates the process.

RESOLUTION

This behavior is by design.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: BseExcept

PRB: SEH with return in the finally Body Preempts Unwind

PSS ID Number: Q91147
Authored 29-Oct-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51 and 4.0

SYMPTOMS

When using Structured Exception Handling (SEH), returning out of a finally body results in a return from the containing procedure scope. For example, in the following code fragment, the return in the finally block results in a return from func():

```
int func()
{
    int status = 0;
    __try {
       ...
       status = test();
       ...
    }
    __finally {
       if (status != 0) {
          status = FAILURE;
          return status;
       }
    }
    return status;
}
```

CAUSE

A return from within a __finally is equivalent to a goto to the closing brace in the enclosing function [for example, func()]. This is allowed, but has consequences that should normally be avoided.

Exception handling has two stages. First, the exception stack is walked, looking for an accepting __except. When an accepting handler has been found, all __finallys between the top-of-exception-stack and the target __except will be called. During this "unwind", the __finallys are assumed to each execute and then return to their caller (the system unwind code).

A return in a finally abnormally aborts this unwinding. Instead of returning to the system unwinder, the __finally returns to the enclosing function's caller [for example, func()'s parent]. The accepting __except filter may set some status or perform an allocation in anticipation of the __except handler being entered. In this case, the intervening

__finally with the return will stop the unwind, and the __except handler is never entered.

RESOLUTION

This is by design. It makes it possible for a finally handler to stop an unwind and return a status. This is what is referred to as a collided unwind.

Abnormal termination from try/except or try/finally blocks is not generally recommended because it is a performance hit.

The example can be rewritten so that the unwind chain is not aborted:

```
int func()
{
   int status = 0;
   __try {
      ...
      status = test();
      ...
   }
   __except(status != 0) {

       /* null */
   }
   if (status != 0)
      status = FAILURE;
   return status;
}
```

This does not have identical semantics because the exception filters higher up the exception stack will not be executed. However, ensuring that both phases of exception handling progress to the same depth is a more robust solution.

MORE INFORMATION

Normally this behavior is transparent to any higher-level exception handling code. If, however, a filter function, as a side effect, stores information that it expects to process in an exception handler, then it may or may not be transparent. Storing such information in a filter function should be avoided because it is always possible that the exception handler will not be executed because the unwind is preempted. In the absence of storing such side effects, it will be transparent that an exception occurred and an attempted unwind occurred if one of the descendent functions has a try/finally block with an finally clause that preempts the unwind.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: BseExcept

PRB: SelectClipRgn() Cannot Grow Clip Region in WM_PAINT

PSS ID Number: Q118472
Authored 19-Jul-1994 Last modified 23-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SYMPTOMS

Setting a smaller clipping region in WM_PAINT by using SelectClipRgn() works fine; however, setting a larger clipping region seems to have no effect. GetClipBox() can be used to verify this after calling SelectClipRgn().

CAUSE

When you call SelectClipRgn() within a BeginPaint()/EndPaint() block in an application's WM_PAINT case, the maximum size to which you can set your clipping region is the size of the update region of your paint structure. This is because the resulting clip region is the intersection of the update region and the region specified in the call to SelectClipRgn(). In other words, you can use SelectClipRgn() to shrink your update region, but not to grow it. This behavior is by design.

RESOLUTION

Invalidate the clipping region area you want before calling BeginPaint(). For example:

```
case WM_PAINT:
   InvalidateRect(hWnd, ....); // Invalidate the size you'll want
                               // for the clip region.
   BeginPaint()
   SelectClipRgn();
   ... paint away ...
   EndPaint();
   break;
```

Something similar could be done in the Microsoft Foundation Classes (MFC), such as:

```
void CMyView::OnPaint()
{
   InvalidateRect(...); // Invalidate the size you'll want.
   CPaintDC dc(this);   // CPaintDC wraps BeginPaint()/EndPaint().
   // Do drawing here...
}
```

MORE INFORMATION

This is addressed in the documentation for the Windows NT SDK version 3.1 [Section 20.1.5, "Window Regions" in Chapter 20, "Painting and Drawing" in the "Microsoft Win32 Programmer's Reference, Volume 1" or in the Win32 API Reference online help (search on "Window Regions")] which states:

> In addition to the update region, every window has a visible region that defines the window portion visible to the user. The system changes the visible region for the window whenever the window changes size or whenever another window is moved such that it obscures or exposes a portion of the window. Applications cannot change the visible region directly, but Windows automatically uses the visible region to create the clipping region for any display DC retrieved for the window.
>
> The clipping region determines where the system permits drawing. When the application retrieves a display DC using the BeginPaint, GetDC, or GetDCEx function, the system sets the clipping region for the DC to the intersection of the visible region and the update region. Applications can change the clipping region by using functions such as SelectClipPath and SelectClipRgn, to further limit drawing to a particular portion of the update area.

Additional reference words: 3.10 3.50 4.00 SelectClipRegion big small large
KBCategory: kbprg kbprb
KBSubcategory: GdiMisc

PRB: Selecting Overlapping Controls in Dialog Editor

PSS ID Number: Q90384
Authored 13-Oct-1992 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

SYMPTOMS

Create a dialog box using the Dialog Editor. Place a button onto the dialog box. Create a frame and place it so that it encompasses the button. It is not possible to select that button with the mouse. However, if the frame is created before the button and then moved or placed over the button, then it is possible to select either the frame or the button.

CAUSE

This behavior is by design. When controls are overlapped, the control that is selected when the mouse is clicked is the one that comes last in Z-order.

As a special case, it is possible to select a control placed "underneath" a group box.

RESOLUTION

From the Arrange menu, choose Order/Group. This will bring up a dialog box. Change the Z-order of the button to be after that of the frame. The Z-order may also be changed by manually editing the resource file. The controls that are further down in the file will be "on top."

NOTE: If the frame is selected and is on top of the button, pressing SHIFT+TAB selects the previous control, which will be the button. This does not allow the position of the control to be changed with the mouse; however, it does allow the text and size to be changed.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool kbprb
KBSubcategory: TlsDlg

PRB: SelectObject() Fails After ImageList_GetImageInfo()

PSS ID Number: Q131279
Authored 07-Jun-1995 Last modified 10-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SYMPTOMS

When you try to select the hBitmap returned by ImageList_GetImageInfo() into a device context, the call to SelectObject() fails and returns NULL.

CAUSE

Under the debug version of Windows 95, attempting to select the hBitmap returned by ImageList_GetImageInfo() into a DC causes the GDI to output the message "Bitmap already selected."

Image lists maintain memory DCs with the image and mask bitmaps already selected. This is done to prevent applications from modifying the images contained in the image list that are currently being used by the system. Because a bitmap cannot be selected into more than one DC at a time, applications that call SelectObject() on the same bitmap fail.

RESOLUTION

An application can work around this in Windows 95 by calling CopyImage() on the hBitmap, as demonstrated in the following sample code. This API is new for Windows 95. Remember to delete the hBitmap copy when using this function.

Sample Code

```
HDC        hDC;
HBITMAP    hBitmap, hOldBitmap;
IMAGEINFO  imageInfo;

ImageList_GetImageInfo (hImgList, iWhichImage, &imageInfo);
hBitmap = CopyImage (imageInfo.hbmImage,
                     IMAGE_BITMAP, 0, 0, LR_COPYRETURNORG);

hOldBitmap = SelectObject(hDC, hBitmap);
  :
  :

// Delete hBitmap when you finish using it.
DeleteObject (SelectObject (hDC, hOldBitmap));
```

STATUS

This behavior is by design.

Additional reference words: 4.00 unusable hDC Image Lists beta
KBCategory: kbprg kbgraphic kbprb kbcode
KBSubcategory: UsrCtl

PRB: SetConsoleOutputCP() Not Functional

PSS ID Number: Q99795
Authored 08-Jun-1993 Last modified 03-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SYMPTOMS

SetConsoleOutputCP() apparently has no effect. The correct codepage is returned from a call to GetConsoleOutputCP, but the displaying of the text remains unchanged.

CAUSE

SetConsoleOutputCP() was designed to change the mapping of the 256 8-bit character values into the glyph set of a fixed-pitch Unicode font, rather than loading a separate, non-Unicode font for each call to SetConsoleOutputCP(). Unfortunately, a fixed-pitch Unicode font was not available by release time, so you can't view the effects of the SetConsoleOutputCP() application programming interface (API) because the currently available console fonts are not Unicode fonts.

STATUS

This behavior is by design in Windows NT versions 3.1 and 3.5.

Additional reference words: 3.10 3.50
KBCategory: kbprg kbprb
KBSubcategory: BseCon

PRB: SetCurrentDirectory Fails on a CD-ROM Drive on Win32s

PSS ID Number: Q125013
Authored 19-Jan-1995 Last modified 20-Jan-1995

--
The information in this articles applies to:

- Microsoft Win32s, versions 1.0, 1.1, and 1.2
--

SYMPTOMS

In the following code, assuming that drive E corresponds to a CD-ROM drive, SetCurrentDirectory() always fails to set the current directory to the root directory on the CD-ROM drive. Instead the current directory remains unchanged:

```
char szCurDir[256];

SetCurrentDirectory("E:\\");
GetCurrentDirectory(sizeof(szCurDir),szCurDir);
MessageBox(NULL, szCurDir, "SCD", MB_OK);
```

CAUSE

SetCurrentDirectory() calls the MS-DOS Interrupt 21h, function 0x4300 to get the file attributes of the specified drive to check whether the specified parameter is a directory. This MS-DOS call always fails if you try to get the attributes of the root directory on a CD-ROM drive, and therefore SetCurrentDirectory() also fails on the root directory of a CD-ROM drive.

STATUS

Microsoft is aware of this problem with SetCurrentDirectory() in Win32s. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

RESOLUTION

As a workaround for the problem with SetCurrentDirectory(), thunk to the 16-bit environment and utilize MS-DOS functions from a 16-bit DLL. For example, you can use Interrupt 21h, function 0x0E (Set Default Drive) followed by Interrupt 21h, function 0x3Bh (Change Current Directory).

MORE INFORMATION

Note that SetCurrentDirectory() fails only on the root directory of a CD-ROM drive. If you pass any directory path other than the root directory to SetCurrentDirectory(), it will work properly.

This is a problem with MS-DOS and can be reproduced from an MS-DOS application in Windows version 3.1.

Additional reference words: 1.00 1.10 1.20 win32s
KBCategory: kbprg kbprb
KBSubcategory: W32s

PRB: Setup Toolkit File Copy Progress Gauge not Updated

PSS ID Number: Q114609
Authored 08-May-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SYMPTOMS

The following may be noticed during the execution of a setup program created with the Setup Toolkit:

1. The copy progress bar appears frozen even though the user may be prompted to change disks.

2. The filename shown at the top of the copy progress dialog box does not change for long periods of time.

CAUSE

The "gas gauge" copy progress dialog box (or Copy Gauge dialog box described on pages 106-107 of the "Setup Toolkit for Windows" manual) is updated only when files are actually being copied to the hard disk. The Setup Toolkit does not update the copy progress dialog box when it checks the version of an existing file. This version check can take a significant amount of time under certain circumstances. A version check is only performed if "Check For Version" is marked in the DSKLAYT program AND the file has a version information resource.

RESOLUTION

There is no way to change this behavior. The dialog box is managed by CopyFilesInCopyList(). The only way to avoid this behavior is to avoid marking files with "Check For Version" in DSKLAYT.

MORE INFORMATION

Under certain circumstances, a file may need to be copied to the temporary directory before its version can be checked. This occurs when the version information in the .INF file matches the version information (exactly) in the file already residing on the hard drive. In this case, the file will be copied from the Setup disks to a temporary location (decompressed if necessary), and other version information will be verified. This can be a time-consuming process and the copy progress dialog box will not be updated while this is occurring.

Additional reference words: 3.10 3.50 4.00 95 MSSETUP CopyFilesInCopyList
KBCategory: kbtool kbprg kbprb
KBSubcategory: TlsMss

PRB: SetWindowsHookEx() Fails to Install Task-Specific Filter

PSS ID Number: Q92659
Authored 12-Nov-1992 Last modified 13-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SYMPTOMS

In Windows version 3.1, the SetWindowsHookEx() function fails when it is called to install a task-specific filter (hook) that resides in a DLL.

CAUSE

According to the documentation, the third parameter to the SetWindowsHookEx() function must be the instance handle of the application or the DLL that contains the filter function. However, because of a problem in Windows 3.1, the SetWindowsHookEx() function fails when it is called to install a task-specific filter using the DLL's instance handle.

Note that such a problem does not exist when the SetWindowsHookEx() function is called to install a system-wide filter in a DLL. The DLL's instance handle is accepted as a valid parameter. The first argument passed to the LibMain function of a DLL contains its instance handle.

RESOLUTION

To install a task-specific filter that resides in a DLL, pass the module handle of the DLL as the third parameter to the SetWindowsHookEx() function. The module handle can be retrieved using the GetModuleHandle() function. For example, to install a task-specific keyboard filter, the code might resemble the following:

```
g_hHook = SetWindowsHookEx( WH_KEYBOARD,
                  HookCallbackProc,
                  GetModuleHandle( "HOOK.DLL" ),
                  hTargetTask );
```

This resolution is compatible with future versions of Windows.

Additional reference words: 3.10 3.50 3.51 4.00 95 hook not allowed
KBCategory: kbprg kbprb kbcode
KBSubcategory: UsrHks

PRB: ShellExecute() Succeeds But App Window Doesn't Appear

PSS ID Number: Q124133
Authored 19-Dec-1994 Last modified 21-Dec-1994

The information in this article applies to:

- Microsoft Win32s, versions 1.15, 1.15a, and 1.2

SYMPTOMS

The following call to ShellExecute() succeeds and the file association is set in File Manager, but the application does not appear to execute (the window is not shown):

```
   hShell = ShellExecute( hWnd,
                          NULL,
                          lpszFile,
                          NULL,
                          lpszDir,
                          SW_SHOWDEFAULT );
```

CAUSE

ShellExecute() is directly thunked to 16-bit Windows. Windows-based applications do not support the SW_SHOWDEFAULT flag.

RESOLUTION

Under Win32s, use SW_NORMAL instead of SW_SHOWDEFAULT when using ShellExecute() with a 16-bit Windows-based application. You can use SW_SHOWDEFAULT if the application specified is a Win32-based application.

STATUS

This behavior is by design.

Additional reference words: 1.20
KBCategory: kbprg kbprb
KBSubcategory: W32s

PRB: SNMP Extension Agent Gives Exception on Windows NT 3.51

PSS ID Number: Q130562
Authored 23-May-1995 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.5 and 3.51

SYMPTOMS

An SNMP extension agent built using Windows NT version 3.5 SDK headers and libraries generates an exception when run under Windows NT version 3.51.

For example, the SDK toaster sample (\MSTOOLS\SAMPLES\WIN32\SNMP\TESTDLL) works under Windows NT version 3.5 but generates an exception under Windows NT version 3.51.

CAUSE

The SNMP.LIB SDK library has changed between the release of Windows NT version 3.5 and the release of Windows NT version 3.51. Memory is now allocated dynamically with the Win32 API GlobalAlloc() rather than the c-runtime malloc(). See the SNMP.H SDK header file for details.

An SNMP application that is allocating (or freeing) memory that is passed to a function in SNMP.LIB should use SNMP_malloc() (or SNMP_free()). The sample code for the extension DLL provided with the Windows NT version 3.51 beta SDK incorrectly uses malloc().

RESOLUTION

Rebuild the extension agent with the Win32 SDK headers and libraries for Windows NT version 3.51. Please make sure that the Win32 SDK headers and libraries are used before Visual C++ headers and libraries.

Also, to allocate and free any memory, use the SNMP_malloc() and SNMP_free() macros. Both are defined in SNMP.H.

NOTE: If you are using a beta version of Windows NT version 3.51, please change all references to malloc() and free() in the samples to SNMP_malloc() and SNMP_free(). This is a known problem with the testdll sample (MSTOOLS\SAMPLES\WIN32\WINNT\SNMP\TESTDLL).

STATUS

This behavior is by design.

REFERENCES

For more information, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID: Q124961
TITLE : BUG: SNMP Sample Generates an Application Error

Additional reference words: 3.50
KBCategory: kbnetwork kbprb
KBSubcategory: NtwkSnmp

PRB: SnmpMgrGetTrap() Fails

PSS ID Number: Q130564
Authored 23-May-1995 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5

SYMPTOMS

If an SNMP manager application calls SnmpMgrGetTrap() to recieve traps and there are traps available, it returns FALSE. If the application calls GetLastError(), the error code returned is 42 (SNMP_MGMTAPI_TRAP_ERRORS).

RESOLUTION

The SNMP Manager API SnmpMgrGetTrapListen() must be called before calling SnmpMgrGetTrap(). The event handle returned by SnmpMgrGetTrapListen() can then be ignored to poll for traps.

STATUS

This behavior is by design.

REFERENCES

For more information, please see the Microsoft Windows NT SNMP Programmer's Reference (PROGREF.RTF).

Additional reference words: 3.50
KBCategory: kbnetwork kbprb
KBSubcategory: NtwkSnmp

PRB: SnmpMgrStrToOid Assumes Oid Is in Mgmt Subtree

PSS ID Number: Q129063
Authored 18-Apr-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SYMPTOMS

When using the SNMP Manager API SnmpMgrStrToOid() and passing it a valid Oid, the application is unable to get the requested variables.

CAUSE

The SNMP Manager API SnmpMgrStrToOid assumes that the Oid that is supplied to it is under the internet MIB of the mgmt subtree (1.3.6.1.2.1.x).

RESOLUTION

To get variables that are not under the mgmt subtree, the Oid must be preceeded by a period (.). For example, say an application is trying to get the system group and the Oid passed to SnmpMgrStrToOid is this:

```
1.3.6.1.2.1.1
```

Then the application will try to get the following, which does not exist:

```
iso.org.dod.internet.mgmt.1.1.3.6.1.2.1.1
```

The correct way to get the system group is to pass this:

```
.1.3.6.1.2.1.1
```

STATUS

This behavior is by design.

REFERENCES

Microsoft Windows/NT SNMP Programmer's Reference (PROGREF.RTF).

Additional reference words: 3.10 3.50
KBCategory: kbnetwork kbprb kbdocerr
KBSubcategory: NtwkSnmp

PRB: Special Characters Missing from Compiled Help File

PSS ID Number: Q86719
Authored 15-Jul-1992 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SYMPTOMS

In version 2.0 of Microsoft Word for Windows, the RTF (rich-text format) source file for a Microsoft Windows Help file contains curly (smart) single or double quotation marks, bullets, or em or en dash characters. However, even though the specified font contains these special characters, they are missing when the compiled file is viewed in Help.

CAUSE

Word for Windows stores special RTF tokens for these characters, which the Microsoft Windows Help Compiler cannot process.

RESOLUTION

In order to avoid these problems do one of the following:

Format the character to use the Symbol font (without using Insert.Symbol).

-or-

Select the desired font (not Symbol) and choose the Symbol option from the Insert menu. Then choose Normal Text in the Symbols From list box. This will format the special character in the desired font.

-or-

Use a bitmap to represent the character or symbol. Use the "bmc" statement to include the bitmap into the text as a character. For more information on the bmc statement, see page 29 of the "Programming Tools" manual provided with version 3.1 of the Microsoft Windows SDK.

Additional reference words: 3.10 3.50 4.00 95 HC HC30.EXE HC31.EXE HCP.EXE
KBCategory: kbtool kbprg kbprb
KBSubcategory: TlsHlp

PRB: Spy Repeatedly Lists a Single Message

PSS ID Number: Q74278
Authored 15-Jul-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SYMPTOMS

SPY sometimes reports messages as having been sent twice. Messages from DDE conversations, especially the WM_DDE_DATA, WM_DDE_ACK, and WM_DDE_POKE messages, are the most often duplicated. This behavior can be seen by choosing All Windows from Spy's Windows menu, selecting DDE messages in Spy's Options dialog, and then running two applications that communicate using DDE.

CAUSE

Spy displays a message each time it is retrieved. If an application retrieves a message once by calling PeekMessage() with PM_NOREMOVE, and then again with GetMessage(), Spy will report it twice. Spy cannot determine that the message was already retrieved by the application. Since the application is using the message twice, the message should indeed be shown twice. This is a useful feature for determining how an application is handling the DDE messages sent to it.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbtool kbprg kbprb
KBSubcategory: TlsSpy

PRB: Starting a Service Returns "Logon Failure" Error

PSS ID Number: Q120556
Authored 14-Sep-1994 Last modified 28-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SYMPTOMS

Starting a service from either the service control manager or from the StartService API may return error 1069, "ERROR_SERVICE_LOGON_FAILED."

CAUSE

This error occurs if the service was started from an account that does not have the "Log on a service" privilege.

RESOLUTION

An account can be granted the "Log on a service right" through the User Manager Application. From the Policies menu, choose User Rights. In the User Rights Dialog Box, select the "Show Advanced User Rights" check box. Choose "Log on a service," in the "Right" scroll box. Then choose the Add button to grant your account the "Log on a service" privilege.

Additional reference words: 3.10 3.50
KBCategory: kbprg kbprb
KBSubcategory: BseService

PRB: Successful LoadResource of Metafile Yields Random Data

PSS ID Number: Q86429
Authored 06-Jul-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SYMPTOMS

When an application for the Microsoft Windows graphical environment calls the LoadResource() function to load a metafile from the application's resources, locks the metafile with the LockResource() function, and uses the metafile, the application receives random data even though the LoadResource() and LockResource() functions indicate successful completion.

CAUSE

The application loaded the metafile previously and when the application freed the metafile, it used the DeleteMetaFile() function to invalidate the metafile handle.

RESOLUTION

Modify the code that unloads the metafile from memory to call the FreeResource() function.

MORE INFORMATION

The LoadResource() and FreeResource() functions change the lock count for a memory block that contains the resource. If the application calls DeleteMetaFile(), Windows does not change the lock count. When the application subsequently calls LoadResource() for the metafile, Windows does not load the metafile because the lock count indicates that it remains in memory. However, the returned memory handle points to the random contents of that memory block.

For more information on the resource lock count, query in the Microsoft Knowledge Base on the following words:

   multiple and references and LoadResource

Most of the time, an application uses the DeleteMetaFile() function to remove a metafile from memory. This function is appropriate for metafiles created with the CopyMetaFile() or CreateMetaFile() functions, or metafiles loaded from disk with the GetMetaFile() function. However, DeleteMetaFile() does not decrement the lock count

of a metafile loaded as a resource.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95 EnumMetaFile GetMetaFile GetMetaFileBits PlayMetaFile PlayMetaFileRecord SetMetaFileBits SetMetaFileBitsBetter
KBCategory: kbprg kbprb
KBSubcategory: UsrRsc

PRB: TAB Key, Mnemonics with FindText and ReplaceText Dialogs

PSS ID Number: Q96134
Authored 09-Mar-1993 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SYMPTOMS

When implementing the FindText and/or ReplaceText common dialog box, the dialog box is displayed but the TAB key and mnemonics do not work properly.

CAUSE

The FindText and ReplaceText common dialog boxes are modeless dialog boxes. Therefore, a call to IsDialogMessage must be made in the application's main message loop in order for the TAB key and mnemonics to work properly.

RESOLUTION

This problem can be corrected by adding a call to IsDialogMessage() in the application's main message loop.

MORE INFORMATION

A typical message loop might resemble the following

```
   while (GetMessage(&msg,NULL,NULL,NULL))
     {
        if ( ghFFRDlg==NULL || !IsDialogMessage(ghFFRDlg, &msg) )
           {
              TranslateMessage(&msg);
              DispatchMessage(&msg);
           }
     }
```

where ghFFRDlg is a global window handle for the currently active modeless common dialog box.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: UsrCmnDlg

PRB: TrackPopupMenu() on LoadMenuIndirect() Menu Causes UAE

PSS ID Number: Q75254
Authored 15-Aug-1991 Last modified 16-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SYMPTOMS

When LoadMenuIndirect() is used to create a menu from a menu template and the menu handle is passed to TrackPopupMenu(), Windows reports an unrecoverable application error (UAE). Windows NT and Windows 95 display the floating pop-up menu as a vertical bar.

CAUSE

The menu handle returned from LoadMenuIndirect() does not point to a menu with the MF_POPUP bit set.

RESOLUTION

The following code fragment demonstrates the correct procedure to "wrap" the menu created by LoadMenuIndirect() inside another menu. This procedure sets the MF_POPUP bit properly.

```
   hMenu1 = LoadMenuIndirect(lpMenuTemplate);

   hMenuDummy = CreateMenu();
   InsertMenu(hMenuDummy, 0, MF_POPUP, hMenu1, NULL);

   hMenuToUse = GetSubMenu(hMenuDummy, 0);
```

Use hMenuToUse when TrackPopupMenu() is called. The values of hMenu1 and hMenuToUse should be the same.

When the menu is no longer required, call DestroyMenu() to remove hMenuDummy. This call will also destroy hMenu1 and free the resources it used.

Additional reference words: 3.00 3.10 3.50 4.00 gpf
KBCategory: kbprg kbprb kbcode
KBSubcategory: UsrMen

PRB: Unable to Choose Kanji Font Using CreateFontIndirect

PSS ID Number: Q119914
Authored 29-Aug-1994 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Japanese Windows 95

SYMPTOMS

You have difficulty getting the Japanese font handle in Microsoft Win32 Software Development Kit (SDK) for Japanese Windows 95. If you use CreateFontIndirect() to create the font handle, only the English font is selected. You are unable to select Kanji (the main system of writing in Japan) fonts such as MS Mincho and MS Gothic.

CAUSE

The lfCharSet field in the LOGFONT structure is not set to "SHIFTJIS_CHARSET".

RESOLUTION

Specify SHIFTJIS_CHARSET as the value for lfCharSet field.

MORE INFORMATION

In Windows 95, Japanese fonts cannot be selected without the lfCharSet field being set to SHIFTJIS_CHARSET. This standard was not enforced in Japanese Windows NT and Japanese Windows 3.1, So an application with lfCharSet set to a value other than SHIFTJIS_CHARSET might be able to select Japanese fonts under Japanese Windows NT and Japanese Windows version 3.1 using CreateFontIndirect(), but not under Japanese Windows 95.

ShiftJIS is a double-byte character set (DBCS) unique to the Japanese version of Windows NT, Windows 95, and Windows version 3.1. It requires a specialized font, keyboard input, and DBCS string-handling support.

Additional reference words: 4.00 kbprb
KBCategory: kbprg kbprb
KBSubcategory: GdiFnt WIntlDev

PRB: Unexpected Result of SetFilePointer() with Devices

PSS ID Number: Q98892
Authored 18-May-1993 Last modified 28-Nov-1994

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SYMPTOMS

Open a floppy drive with CreateFile():

```
h = CreateFile( "\\\\.\\a:", ... );
```

Use SetFilePointer() to advance the file pointer associated with the file handle returned from CreateFile():

```
SetFilePointer( h,    // file handle
                5,    // distance (in bytes) to move file pointer
             NULL,    // optional high 32-bits of distance
       FILE_BEGIN     // specifies the starting point
);
```

If the offset is not a multiple of the sector size of the floppy drive, the function will return success; however, the pointer will not be exactly where requested. The pointer value is rounded down to the beginning of the sector that the pointer value is in.

CAUSE

The behavior of this application programming interface (API) is by design for the following reasons:

- The I/O system is unaware of device particulars such as sector size; any offset is valid.

- SetFilePointer() is very frequently used. Because speed is an important goal for Windows NT, time is not spent on querying device particulars and detecting such errors.

- The logic to handle this situation is built into the file system, which actually performs the rounding, and therefore there was no need to put this into the code for SetFilePointer().

RESOLUTION

When using SetFilePointer() with a handle that represents a floppy drive, the offset must be a multiple of the sector size for the floppy drive in order for the function to perform as expected.

MORE INFORMATION

Think of a file pointer as merely a stored value, which is where the next read or write will take place. In fact, it is possible to override this value on either the read or write itself, using certain APIs, by supplying a different location. The new pointer location is remembered after the operation. Therefore, the operation of "setting a file pointer" merely means to go store a large integer in a cell in the system's data structures, for possible use in the next file operation. In the case of a handle to a device, the file pointer must be on a sector boundary.

In a similar way, ReadFile() only reads amounts that are multiples of the sector size if it is passed a handle that represents a floppy drive.

Additional reference words: 3.10
KBCategory: kbprg kbprb
KBSubcategory: BseFileio

PRB: Unicode ChooseColor() Help Button May Crash Common

PSS ID Number: Q102026
Authored 27-Jul-1993 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1, 3.5, and 3.51

SYMPTOMS

A Unicode compiled program, or a program that explicitly calls the ChooseColorW() function, may suffer an access violation in the common dialog boxes if the Help button in the ChooseColor() dialog box is chosen.

CAUSE

This problem may not be 100-percent reproducible. Because of the way that the program loader loads code into memory, a certain address referenced by the common dialog box Help button logic may or may not be valid depending on what other applications have been run, what application is calling ChooseColorW(), and the order in which these programs are run.

WORKAROUND

The following are suggested workarounds:

- Try running an ANSI-compiled application that uses the common dialog boxes, and open one of the common dialog boxes before you run the application having this problem.

- A workaround for new applications developers is to call ChooseColorW() with a hook function (see the Windows Software Development Kit documentation for more information on common dialog box hook functions), handle the Help button logic, and return TRUE from the hook function to bypass the standard common dialog box push-button code.

Additional reference words: 3.10 3.50
KBCategory: kbprg kbprb
KBSubCategory: UsrCmnDlg

PRB: UnrealizeObject() Causes Unexpected Palette Behavior

PSS ID Number: Q86800
Authored 16-Jul-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SYMPTOMS

In an application for the Microsoft Windows graphical environment, when a logical palette (HPALETTE) is used with a device-dependent bitmap (DDB) and the application realizes the palette, the DDB is painted in incorrect colors.

CAUSE

The UnrealizeObject function was previously called to unrealize the palette.

RESOLUTION

Modify the code to remove the call to UnrealizeObject()/

MORE INFORMATION

Because the colors of a DDB are stored using indices into the system palette rather than explicit RGB colors, proper DDB rendering depends on the colors of the system palette being set properly. An application sets up the system palette to display a DDB by realizing a logical palette. The realization process changes the colors in the system palette and creates a mapping between entries in the logical palette and entries in the system palette.

When an application first renders a logical palette with RealizePalette(), Windows sets an internal flag to indicate that the palette has been realized and stores the current mapping from logical palette entries to physical palette entries. When an application realizes the palette again (for example, after another application modifies the palette), Windows restores the effected entries of the system palette to the state they had when the logical palette was realized for the first time.

This mechanism allows a bitmap first realized with a specific palette to display correctly when the same palette is realized subsequently.

If the application calls UnrealizePalette() on a logical palette, Windows discards the stored state information for the palette. If the

application realizes the palette subsequently, its colors may map into new locations in the system palette. Because the bitmap contains indices into the old system palette, it may display incorrectly.

To address this situation, do not call UnrealizeObject() on a palette if the application has a DDB that depends on that palette.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: GdiPal

PRB: Vertical Scroll Bars Missing from Windows Help

PSS ID Number: Q77841
Authored 28-Oct-1991 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SYMPTOMS

The vertical scroll bars do not appear in Windows Help when the window is sized smaller than the amount of text being displayed.

CAUSE

The displayed topic is formatted as a nonscrolling region.

RESOLUTION

Select the topic in the RTF editor and turn off the "Keep Paragraph with Next" formatting. This format is used by the Help Compiler to delimit the nonscrolling regions.

MORE INFORMATION

To remedy the situation in Microsoft Word for Windows, perform the following three steps:

1. Highlight the entire topic.

2. From the Format menu, choose Paragraph.

3. In the Format Paragraph dialog box, cancel the "With Next" check box.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool kbprg kbprb
KBSubcategory: TlsHlp

PRB: Video for Windows Skips AVI Frames

PSS ID Number: Q122775
Authored 13-Nov-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SYMPTOMS

When Video For Windows is playing an AVI file, the processing power of the computer system and the data rate of the AVI stream determine whether frames will be skipped during playback. If the data rate exceeds the ability of Video For Windows to render all the AVI frames in time, Video For Windows will first try to catch up by decompressing delta frames but not drawing them. If Video For Windows still can't catch up, it will eventually skip all the way to the next key frame before it renders a new frame.

STATUS

This behavior of Video for Windows is by design and cannot be controlled by the application being affected.

Additional reference words: 3.10 3.50 4.00 95 VfW AVI
KBCategory: kbmm kbprb
KBSubcategory: MMVideo

PRB: Win32-Based Screen Saver Shows File Name in Control

PSS ID Number: Q126239
Authored 16-Feb-1995 Last modified 22-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1 and 3.5

SYMPTOMS

After writing a Win32-based screen saver, if the IDS_DESCRIPTION string is missing from the string table, the file name of the screen saver module is displayed in the desktop control panel applet. This happens even if the DESCRIPTION entry is specified in the .DEF file of the module.

CAUSE

Each Windows screen saver has a name, which is a string packed into the screen saver module by the linker. This name is displayed in the screen saver name drop down list box under the desktop applet in the control panel. The user can select different screen savers for the desktop through this list box.

Under 16-bit Windows, the name of a screen saver is specified by the DESCRIPTION entry in the .DEF file. Under Windows NT, this is no longer true. Instead, a special entry in the string table is used to specify the name of a screen saver. This string must have IDS_DESCRIPTION as its string ID. IDS_DESCRIPTION is defined in SCRNSAVE.H.

If the DESCRIPTION entry is missing from the .DEF file for a 16-bit screen saver, or the IDS_DESCRIPTION is missing from the string table for a 32-bit screen saver, Windows NT displays the file name of the screen saver module in the drop down list box in the Desktop control panel applet.

RESOLUTION

Place the DESCRIPTION entry in .DEF file for a 16-bit screen saver, or place the IDS_DESCRIPTION in the string table for a 32-bit screen saver.

STATUS

This behavior is by design.

REFERENCES

Chapter 14, "Screen Saver Library," Microsoft Windows Software Development Kit, version 3.1, Programmer's Reference, Volume 1: Overview.

Chapter 79, "Screen Saver Library," Microsoft Win32 Programmer's Reference, Volume 2: System Services, Multimedia, Extensions, and Application Notes.

Additional reference words: 3.10 3.50
KBCategory: kbprg kbprb
KBSubcategory: GdiScrsav

PRB: Win32s GetVolumeInformation() Returns 0x12345678 or 0

PSS ID Number: Q93639
Authored 17-Dec-1992 Last modified 07-Dec-1994

The information in this article applies to:

- Microsoft Win32s version 1.0, 1.1, 1.15, and 1.2

SYMPTOMS

In Win32s version 1.0 and 1.1, GetVolumeInformation() always returns a volume ID of 0x12345678. In Win32s version 1.15 and later, the return is 0.

STATUS

This is a known limitation of Win32s and is by design.

Additional reference words: 1.00 1.10 1.20
KBCategory: kbprg kbprb
KBSubcategory: W32s

PRB: Windows REQUEST Function Not Working With Excel

PSS ID Number: Q26234
Authored 18-Dec-1987 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SYMPTOMS

The REQUEST function does not work correctly with Excel. The request message is received, however, Excel does not process the WM_DDE_DATA message that is sent back.

RESOLUTION

The fResponse bit must also be set in the WM_DDE_DATA message (bit 12). This bit tells Excel that the data message is in reply to a REQUEST function and not an ADVISE function. If "lpddeup->fResponse=1" is added, the REQUEST function should work correctly.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: UsrDde

PRB: WinExec() Fails Due to Memory Not Deallocated

PSS ID Number: Q126710
Authored 28-Feb-1995 Last modified 03-Mar-1995

The information in this article applies to:

- Microsoft Win32s, versions 1.15, 1.2, and 1.25

SYMPTOMS

Under Win32s version 1.15, when a Win32-based application spawns a 16-bit application several times using WinExec(), after a few successful spawns, WinExec() fails.

Each time WinExec() is called to start a 16-bit application, Win32s allocates a fixed and pagelocked block. The owner of this block is the Win32-based application. The memory is not deallocated when the 16-bit application is terminated, only when the 32-bit application is terminated.

CAUSE

This is actually a bug in Windows version 3.1. The 32-bit WinExec() calls the 16-bit LoadModule(). Win32s passes the environment of the calling process to LoadModule(). Then the Windows 3.1 LoadModule() allocates a buffer for the environment, copies this environment to the buffer, and passes this buffer to the child process. The problem is that the owner of the new allocated buffer is the parent, so the memory is freed when the parent exits. There is no code for otherwise freeing the memory. This bug also affects 16-bit Windows-based applications if LoadModule() is called with an environment selector that is not NULL.

In a related problem, when the parent terminates, the child's environment becomes invalid. This may cause a general protection (GP) fault.

RESOLUTION

To work around the problem, you can call the Windows version 3.1 WinExec() through the Universal Thunk. However, the parent will not be able to modify the child's environment.

In Win32s version 1.2x, this problem exists only if you start 16-bit application using CreateProcess() or LoadModule() and pass it explicit environment strings. In this case, you will encounter the Windows version 3.1 bug. If you do not pass an explicit environment, the environment passed to the 16-bit application is NULL. This resolves the problems mentioned in the Symtpoms and Cause sections of this article.

MORE INFORMATION

With the changes in Win32s version 1.2x, if the calling application modifies the environment, the child process will not get the modified environment of the parent. It will get the global MS-DOS environment. This is also true for WinExec().

If you need to pass a modified environment, call LoadModule() or CreateProcess() with the environment set to what GetEnvironmentStrings() returns. Be aware that this will cause a memory leak. In addition, if the parent terminates before the child, the child's environment will become invalid.

Additional reference words: 1.20 GPF
KBCategory: kbprg kbprb
KBSubcategory: W32s

PRB: WINS.MIB & DHCP.MIB Files Missing from Win32 SDK 3.5

PSS ID Number: Q121625
Authored 12-Oct-1994 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5

SYMPTOMS

Two Simple Network Management Protocol (SNMP) files (WINS.MIB and DHCP.MIB) are missing from the currently released version of the Win32 SDK for Windows NT version 3.5.

The files are two Management Information Base (MIB) files for WINS and DHCP SNMP usage. The two files (WINS.MIB and DHCP.MIB) are used in the generation of a MIB.BIN file for use with the NT SNMP Extendible Agent Management API.

RESOLUTION

Only developers working on SNMP programming using the SNMP Extendible Agent Management API will need these two .MIB files. To get the two .MIB files:

Download NEWMIB.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for NEWMIB.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download NEWMIB.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the SOFTLIB\MSLFILES directory
  Get NEWMIB.EXE

The two missing .MIB files will be included with the next release of the Win32 SDK for Windows NT version 3.5.

All licensing and redistribution agreements from the Win32 SDK for Windows NT version 3.5 also apply to these .MIB files. Please consider the two .MIB files part of the Win32 SDK. Please see your licensing agreement for the Win32 SDK for more details.

Additional reference words: 3.50 softlib

KBCategory: kbprg kbfile kbprb
KBSubcategory: NtwkSnmp

PRB: WSAAsyncSelect() Notifications Stop Coming

PSS ID Number: Q94088
Authored 23-Dec-1992 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SYMPTOMS

I have set a WSAAsyncSelect() call to notify me of read (FD_READ) and disconnection (FD_CLOSE). When a read call is posted on my message queue, I continually read from the socket until there are no more characters waiting. After each read, I use a select() call to determine if more data needs to be read. However, after a while, the notifications stop coming. Why is this?

CAUSE

The message queue must be cleared of extraneous notification messages for each read notification message.

RESOLUTION

Call WSAAsyncSelect(sockt, hWnd, 0, 0) to clear the message queue for each read notification.

MORE INFORMATION

Sample Code

```
WSA_READCLOSE:
  if (WSAGETSELECTEVENT( lParam ) == FD_READ) {

    FD_ZERO( &readfds );
    FD_SET( sockt, &readfds);

    timeout.tv_sec = 0;
    timeout.tv_usec = 0;

    /* Clear the queue of any extraneous notification messages. */

    WSAAsyncSelect( sockt, hWnd, 0, 0);

    while (select(0, &readfds, NULL, NULL, &timeout) != 0) {
      recv(sockt, &ch, 1, 0);
      }
```

```
    /* Reset the message notification. */

    WSAAsyncSelect( sockt, hWnd, WSA_READCLOSE, FD_READ | FD_CLOSE);
    }
```

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: NtwkWinsock

PRB: WSAStartup() May Return WSAVERNOTSUPPORTED on Second

PSS ID Number: Q130942
Authored 30-May-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

SYMPTOMS

If two sections of code within the same process call WSAStartup(), the second call to WSAStartup() fails and returns error WSAVERNOTSUPPORTED unless the second call specifies the version negotiated in the first call.

This happens even if the requested version would normally be accepted. Often the extra calls to WSAStartup() come from one or more DLLs loaded by the process.

RESOLUTION

If multiple calls are made to WSAStartup(), the second call must request the same version negotiated in the first call.

MORE INFORMATION

Some specific examples may help. Currently, if the version of Winsock requested is 1.1 or greater, the negotiated version will be 1.1. If a version less than 1.1 is requested, the call fails and returns the WSAVERNOTSUPPORTED error.

Example One

First call : 1.1 requested
Second call: 1.1 requested
Result : Success

Example Two

First call : 2.0 requested
Second call: 1.1 requested
Result : Success

Example Three

```
First call : 2.0 requested
Second call: 2.0 requested
Result     : WSAVERNOTSUPPORTED

Example Four
------------

First call : 1.1 requested
Second call: 2.0 requested
Result     : WSAVERNOTSUPPORTED

Additional reference words: 3.50 4.00 95 3.10
KBCategory: kbprg kbprb
KBSubcategory: NtwkWinsock
```

PRB:Scroll Bar Continues Scrolling After Mouse Button

PSS ID Number: Q102552
Authored 04-Aug-1993 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0, 3.1, and 4.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SYMPTOMS

The scroll bar continuously scrolls even after the left mouse button is released. The type of scroll bar is irrelevant to this problem, that is, the same problem occurs regardless of whether the scroll bar is part of the window or is a scroll bar control.

CAUSE

This problem occurs usually when a message retrieval loop is executed as the result of actions taken for scrolling upon receiving one of the scroll bar notification messages.

When scrolling, an internal message retrieval loop is started in Windows. The task of this message loop is to keep track of scrolling and to send the appropriate scroll bar notification messages, WM_HSCROLL and WM_VSCROLL. Scrolling is terminated once WM_LBUTTONUP is received. If another message loop is started during scrolling, the WM_LBUTTONUP is retrieved by that message loop, and because an application does not have access to the scroll bar's internal message retrieval loop, WM_LBUTTONUP cannot be dispatched correctly. Therefore, the WM_LBUTTONUP is never received by the internal message retriever, and scrolling is never ended.

The application that is scrolling does not have to retrieve messages explicitly to cause this problem. Calling any of the following functions or processing any message that has a message retrieval loop, while scrolling, can cause the WM_LBUTTONUP to be lost. The functions listed below fall into this category:

```
DialogBox()
DialogBoxIndirect()
DialogBoxIndirectParam()
DialogBoxParam()
GetMessage()
MessageBox()
PeekMessage()
```

RESOLUTION

While Scrolling, the WM_LBUTTONUP message should not be retrieved from the queue by any message retrieval loop other than the scroll bar's internal one.

An application may come across this problem as follows:

- An application implements a message retrieval loop to implement background processing, for example, background processing while performing a time consuming paint.

- An application implements a message retrieval loop to implement communication with another application or DLL. For example, in order to scroll, the application needs to receive data from elsewhere.

Possible Workarounds

Two possible workarounds are listed below. The first workaround is used by many exiting applications and by Windows; however, under rare circumstances the first workaround may not be a feasible one. In this case, the second workaround may be used. However, if possible, please try to avoid implementing message retrieval completely while scrolling.

- Use timer-message-based processing. Break down the complicated processing into smaller tasks and keep track of where each task starts and ends, then perform each task based on a timer message. When all components of the processing are complete, kill the timer. See below for an example of this workaround.

- Implement a message retrieval loop, but make sure WM_LBUTTONUP is not retrieved by it. This can be accomplished by using filters. See below for some examples of this workaround.

Example Demonstrating Workaround 1

An application has a complex paint procedure. Calling ScrollWindow(), to scroll, generates paint messages. Background processing takes place while painting.

1. When receiving the WM_PAINT message do the following:

   a. Call BeginPaint().

   b. Copy the invalidated rect to a global rect variable (for example, grcPaint) to be used in step 2. The global rect grcPaint would be a union of the previously obtained rect (grcPaint) and the new invalidated rect (ps.rcPaint). The code for this will resemble the following:

```
RECT grcPaint;    // Should be initialized before getting the
                  // first paint message.
   :
```

```
          :
        UnionRect(&grcPaint, &ps.rcPaint,&grcPaint);
```

   c. Call ValidateRect() with ps.rcPaint.

   d. Call EndPaint().

   e. Set a Timer.

   This way, no more WM_PAINT messages are generated, because there are no invalid regions, and a timer is set up, which will generate WM_TIMER messages.

2. Upon receiving a WM_TIMER message, check the global rect variable; if it is not empty, take a section and paint it. Then adjust the global rect variable so it no longer includes the painted region.

3. Once the global rect variable is empty then kill the timer.

Example Demonstrating Workaround 2

An application needs to obtain some data through DDE or some other mechanism from another application, which is then displayed in the window. In order to scroll, the application needs to request and then obtain the data from a server application.

There are three different filters that can be used to set up a PeekMessage() and get the information. The filters can be set up by using the uFilterFirst and uFilterLast parameters of PeekMessage(). uFilterFirst specifies the fist message in the range to be checked and uFilterLast specifies the last message in the range to be checked. For more information on PeekMessage() and its parameters, see the Windows SDK "Programmer's Reference, Volume 2: Functions" for version 3.1 and "Reference, Volume 1" for version 3.0.

1. Check and retrieve only the related message(s) for obtaining the needed data.

2. Check for WM_LBUTTONUP without removing it form the queue; if it is in the queue, break. Otherwise, retrieve and dispatch all messages.

3. Retrieve all messages less than WM_LBUTTONUP and greater than WM_LBUTTONUP, but do not retrieve WM_LBUTTONUP.

MORE INFORMATION

Steps to Reproduce Behavior

The following is the sequence of events leading to the loss of the WM_LBUTTONUP message:

1. Click the scroll bar using the mouse.

2. Step 1 generates a WM_NCLBUTTONDOWN message.

3. Step 2 causes a Windows internal message loop to be started. This message loop looks for scroll-bar-related messages. The purpose of this message loop is to generate appropriate WM_HSCROLL or WM_VSCROLL messages. The message loop and scrolling terminates once WM_LBUTTONUP is received.

4. When receiving the WM_HSCROLL or WM_VSCROLL message, the application either gets into a message retrieval loop directly or calls functions which result in retrieval of messages.

5. WM_LBUTTONUP is removed from the queue by the message loop mentioned in step 4. WM_LBUTTONUP is then dispatched.

6. As result of step 5 WM_LBUTTONUP message is dispatched elsewhere and the internal message retrieval loop, mentioned in step 3 never receives it. The message loop in step 3 is looking for the WM_LBUTTONUP to stop scrolling. Because it is not received, the scroll bar continues scrolling.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95 scrollbar stuck
KBCategory: kbprg kbprb
KBSubcategory: UsrCtl

PRB:Unselecting Edit Control Text at Dialog Box

PSS ID Number: Q96674
Authored 24-Mar-1993 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, versions 3.50, 3.51, and 4.0

SYMPTOMS

To remove the highlight (selection) from an edit control text, an EM_SETSEL message must be sent to the control. However, while processing the WM_INITDIALOG message of a dialog box, sending an EM_SETSEL fails to remove the highlight from (unselect) the edit control text.

CAUSE

While processing the WM_INITDIALOG message, sending the EM_SETSEL message fails to remove the highlight from the edit control. This happens because the edit control has not yet been drawn. Because it's not drawn and there is no selection information available to the edit control's procedure, the EM_SETSEL message is ignored. In other words, the SendMessage() function passes the EM_SETSEL message too early to the edit control for it to become effective.

RESOLUTION

There are two solutions to the above problem.

Solution 1

Use SetFocus() to set the input focus on the edit control. Use PostMessage() to post the EM_SETSEL message to the edit control rather than using SendMessage() and return FALSE from the WM_INITDIALOG handler.

Solution 2

When a newly created dialog box is displayed with focus on an edit control, the default text of the edit control is shown highlighted. In some cases, the text highlighting is undesirable because accidentally pressing a character key removes the original text from the edit control. Therefore, the workaround is to unselect the text by sending an EM_SETSEL message to the edit control at the dialog box initialization.

Delay the EM_SETSEL message until the focus is set to the edit control. That is, while processing the first EN_SETFOCUS notification message, an EM_SETSEL message must be sent to the edit control to remove the highlight

from its text. For example:

```
   static  BOOL    bFirstTime;   // We want to unselect only once.

   switch ( message )
   {
       case WM_INITDIALOG:
           bFirstTime = TRUE;
           return TRUE;

       case WM_COMMAND:
           switch ( wParam )
           {
               case IDC_EDIT:
                   // If this is the first time, then unselect.
                   if ( HIWORD( lParam ) == EN_SETFOCUS &&
                        bFirstTime )
                   {
                      SendMessage(GetDlgItem( hwndDialog, IDC_EDIT ),
                                  EM_SETSEL, 0,
                      MAKELPARAM( -1, -1 ));
                      bFirstTime = FALSE;
                   }
                   break;
           .
           .
           .
           }  // switch ( wParam )
      .
      .
      .
   } // switch ( message )
```

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbprb
KBSubcategory: UsrCtl

Precautions When Passing Security Attributes

PSS ID Number: Q94839
Authored 19-Jan-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

All Win32 APIs that allow security to be specified take a parameter of type LPSECURITY_ATTRIBUTES as the means to attach the security descriptor. However, it is a common error to pass a PSECURITY_DESCRIPTOR type to such functions instead. Because PSECURITY_DESCRIPTOR is of type LPVOID (for opaque data-type reasons), by C Language definition, it is implicitly converted to the correct type. Therefore, the compiler does not generate any warnings; however, unexpected run-time errors will result.

MORE INFORMATION

Below is a correct example of creating a named pipe with a security descriptor attached.

Sample Code

```
   saSecurityAttributes.nLength = sizeof(SECURITY_ATTRIBUTES);
   saSecurityAttributes.lpSecurityDescriptor = psdAbsoluteSD;
   saSecurityAttributes.bInheritHandle = FALSE;

   hPipe = CreateNamedPipe(TEST_PIPE_NAME,
                           PIPE_ACCESS_DUPLEX,

   (PIPE_TYPE_BYTE|PIPE_READMODE_BYTE|PIPE_WAIT),
                           100,  // maximum instances
                           0,    // output buffer, sized as needed
                           0,    // input buffer, sized as needed
                           100,  // timeout in milliseconds

   (LPSECURITY_ATTRIBUTES)&saSecurityAttributes);
   if( INVALID_HANDLE_VALUE == hPipe )
   { // handle error
   }
```

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

Preventing Screen Flash During List Box Multiple Update

PSS ID Number: Q66479
Authored 26-Oct-1990 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.1 3.5, 3.51, and 4.0

SUMMARY

The WM_SETREDRAW message can be used to set and clear the redraw flag for a window. Before an application adds many items to a list box, this message can be used to turn the redraw flag off, which prevents the list box from being painted after each addition. Properly using the WM_SETREDRAW message keeps the list box from flashing after each addition.

MORE INFORMATION

The following four steps outline ways to use the WM_SETREDRAW message to facilitate making a number of changes to the contents of a list box in a visually pleasing manner:

1. Clear the redraw flag by sending the list box a WM_SETREDRAW message with wParam set to FALSE. This prevents the list box from being painted after each change.

2. Send appropriate messages to make any desired changes to the contents of the list box.

3. Set the redraw flag by sending the list box a WM_SETREDRAW message with wParam set to TRUE. The list box does not update its display in response to this message.

4. Call InvalidateRect(), which instructs the list box to update its display. Set the third parameter to TRUE to erase the background in the list box. If this is not done, if a short list box item is drawn over a long item, part of the long item will remain visible.

The following code fragment illustrates the process described above:

```
/* Step 1: Clear the redraw flag. */
SendMessage(hWndList, WM_SETREDRAW, FALSE, 0L);

/* Step 2: Add the strings. */
for (i = 0; i < n; i++)
   SendMessage(hWndList, LB_ADDSTRING, ...);

/* Step 3: Set the redraw flag. */
SendMessage(hWndList, WM_SETREDRAW, TRUE, 0L);
```

```
   /* Step 4: Invalidate the list box window to force repaint. */
   InvalidateRect(hWndList, NULL, TRUE);
```

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95 flash flicker
KBCategory: kbprg
KBSubcategory: UsrCtl

Preventing the Console from Disappearing

PSS ID Number: Q99115
Authored 23-May-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

When a console application is started from the File Manager, from the Program Manager, or by typing "start <progname>" from the command prompt, it executes in its own console. This console disappears as soon as the application terminates, and therefore the user can't read anything written to the screen between the last pause and program exit. To resolve this problem, the programmer should pause the application before termination to allow the user to read all of the information on the screen.

It is not likely that the programmer will want to introduce this pause if the application is started directly from the command prompt, because in this situation it won't make much sense to the user. However, there is no API (application programming interface) that directly determines whether or not the application shares a console with CMD.EXE. There is a method that can be used to determine this information in most cases. When the application first starts up, call GetConsoleScreenBufferInfo(). If the cursor position is (0, 0), then the application has its own console, which will disappear when the application terminates. Otherwise, the application is operating within a console belonging to another program, typically CMD.EXE.

NOTE: This method will not work if the user combines a clear screen (CLS) and execution of the application into one step ([C:\] CLS & <progname>), because the cursor position will be (0, 0), but the application is using the console, which belongs to CMD.EXE.

MORE INFORMATION

To start a console application with its own console that will not disappear when the application is terminated, use CMD /K. For example, use

```
start CMD /K <progname>
```

Note that it is possible to programmatically force an application to always have its own console by immediately doing a FreeConsole() and an AllocConsole(). The disadvantage is that the C run-time handles are no longer valid. Use CreateFile("CONIN$", ...) with lpsa->bInherit=TRUE, in combination with _open_osfhandle() and dup2() to close the current handles (stdin, stdout, stderr) and associate handles that will be inherited.

Additional reference words: 3.10 3.50

KBCategory: kbprg
KBSubcategory: BseCon

Preventing Word Wrap in Microsoft Windows Help Files

PSS ID Number: Q88142
Authored 18-Aug-1992 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

By default, the Microsoft Windows Help application wraps lines of text to reflect the size of its window. However, there are situations (such as a table of information) in which wrapping text is undesirable. The information below presents two methods of preventing a section of text from changing when the Help application window changes sizes. These techniques apply to version 2.0 of the Microsoft Word for Windows-based application.

MORE INFORMATION

Method 1

This method, which is compatible with versions 3.0 and 3.1 of the Microsoft Windows Help Compiler, involves two steps:

1. Place either a hard or a soft carriage return at the end of each line.

2. Format the section with the "keep lines together" paragraph attribute. From the Format menu, choose Paragraph, and select the Keep Lines Together check box in the Paragraph dialog box.

Method 2

This method, which is compatible only with version 3.1 of the Help Compiler, is to create a one row, one column table in Word for Windows. Set the width of the table as desired and allow the text to wrap within the table normally. Windows Help will duplicate the word breaks in the table provided that the font used to author the table is selected by the Help engine when displaying the table. If Help uses a different font, the text may wrap differently, even though the table keeps the specified width.

Note

When you use either of these methods, if the Windows Help window is

not large enough to display the entire width of a topic, Help displays a horizontal scroll bar rather than wrapping the text to make it visible.

Additional reference words: 3.00 3.10 3.50 4.00 95 HLP word wrap wordwrap engine HC31 HC31.EXE HCP HCP.EXE
KBCategory: kbtool
KBSubcategory: TlsHlp

Primitives Supported by Paths Under Windows 95

PSS ID Number: Q125697
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

Windows 95 supports the full set of Win32 path APIs. However, only a limited set of primitives can be used to build a path. Windows 95 supports the following primitives to build paths:

ExtTextOut
LineTo
MoveToEx
PolyBezier
PolyBezierTo
Polygon
Polyline
PolylineTo
PolyPolygon
PolyPolyline
TextOut

All other Win32 primitives will be ignored if used in a path. As with other Win32 primitives in Windows 95, paths are not mapped pixel-by-pixel to Windows NT; they support only 16-bit coordinates.

Additional reference words: 4.00 GDI
KBCategory: kbprg
KBSubcategory: GdiMisc

Printer Escapes Under Windows 95

PSS ID Number: Q125692
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

Printer escapes are used to access special printer device features and have been used widely in Windows version 3.x. With Windows 95, Microsoft is encouraging application developers to move away from these escapes by providing GDI functionality to replace them.

For example, a Win32-based application should not call the NEXTBAND and BANDINFO escapes. Banding is no longer needed in Windows 95. Most of these escapes, however, are still provided for 16-bit-based applications for backwards compatibility. The only recommended escapes for 32-bit, Windows 95-based applications are the QUERYESCSUPPORT and PASSTHROUGH escapes.

MORE INFORMATION

Applications written for Windows version 3.x can use the QUERYESCSUPPORT and the PASSTHROUGH escapes, as well as the following 10 escapes. It is important to note that these escapes are only supported for backwards compatibility. All new Windows 95-based applications should use Win32 API that replaces these escapes:

```
ABORTDOC
ENDDOC
GETPHYSPAGESIZE
GETPRINTINGOFFSET
GETSCALINGFACTOR
NEWFRAME
NEXTBAND
SETABORTPROC
SETCOPYCOUNT
STARTDOC
```

The following functions should always be used in place of a printer escape:

Function	Printer Escape Replaced
AbortDoc	ABORTDOC
EndDoc	ENDDOC
EndPage	NEWFRAME
SetAbortProc	SETABORTPROC
StartDoc	STARTDOC

Windows 95 provides six new indexes for the GetDeviceCaps function that replace some additional printer escapes:

Index for GetDeviceCaps	Printer Escape Replaced
PHYSICALWIDTH	GETPHYSPAGESIZE
PHYSICALHEIGHT	GETPHYSPAGESIZE
PHYSICALOFFSETX	GETPRINTINGOFFSET
PHYSICALOFFSETY	GETPRINTINGOFFSET
SCALINGFACTORX	GETSCALINGFACTOR
SCALINGFACTORY	GETSCALINGFACTOR

Although a lot of the escapes have been replaced with Win32 GDI equivalent APIs, not all device-dependent escapes have been replaced. It is up to the printer driver manufacturer to decide whether or not its Windows 95-based driver will contain device-specific escapes that were present in its Windows version 3.x driver. An example of a device-specific escape would be the Windows version 3.x PostScript driver's POSTSCRIPT_IGNORE escape. Before calling any of these escapes, an application must first call the QUERESCSUPPORT escape to find out if the escape is supported or not.

Additional reference words: 4.00
KBCategory: kbprint
KBSubcategory: GdiPrnMisc

Printing in Windows Without Form Feeds

PSS ID Number: Q11915
Authored 26-Oct-1987 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

A Windows-based application cannot print directly to a printer without issuing a form feed. In 16-bit Windows, the NEWFRAME escape is required to start the print spooler. This escape, in turn, sends a form feed to the printer. Without the NEWFRAME escape, the spooler never runs, and nothing is output to the printer. Win32-based applications should use EndPage(), which replaces the NEWFRAME escape.

MORE INFORMATION

It is possible to drive the spooler directly by using the spooler functions documented in the Windows Device Development Kit (DDK). This allows the printer driver to be bypassed. However, by doing this, the ability to use GDI output functions and Windows's device-independent capabilities will be lost.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiPrn

Printing Monochrome and Color Bitmaps from Windows

PSS ID Number: Q64520
Authored 06-Aug-1990 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The format of a display bitmap determines the procedure that an application uses to print it. The two display bitmap formats available under Windows are device-dependent bitmaps (DDBs) and device-independent bitmaps (DIBs). DIBs and DIB functions should be used for printing color bitmaps.

MORE INFORMATION

An application can use the BitBlt() or StretchBlt() function to print or display a monochrome bitmap. Both printer drivers and display drivers can process monochrome DDBs. However, an application must account for the difference in resolution between a typical display and a typical laser printer. The StretchBlt() function enables an application to appropriately change the size of a monochrome bitmap.

When the display bitmap is a color DDB, printing is more difficult because the display DDB format may not match the printer DDB format. Because Windows supports a wide variety of devices, this situation is quite common. When the formats DDB differ, the application must convert the display DDB into a print DDB or a DIB.

DIBs are designed to ease the process of transferring images between devices. When an application uses a DIB, the GDI or the output driver performs any conversions required for the device. The ShowDIB sample application, provided in the Windows SDK and the Win32 SDK, demonstrates converting a DDB to a DIB and other common manipulations. The file DIB.C is of particular interest. It contains the functions that perform the manipulations. This code can be incorporated into other applications.

For more information, please see the Windows SDK 3.1 DIBView sample or the Win32 SDK WinCap32 sample.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiPrn

Printing Offset, Page Size, and Scaling with Win32

PSS ID Number: Q115762
Authored 02-Jun-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

The Win32 documentation for the Escape() function says that the GETPHYSPAGESIZE, GETPRINTINGOFFSET, and GETSCALINGFACTOR escapes are obsolete, but it fails to mention the recommended way to get this information.

The information retrieved by all three escapes can now be retrieved by calling GetDeviceCaps() with the appropriate index:

- For the GETPHYSPAGESIZE escape, the indexes to be used with GetDeviceCaps() are PHYSICALWIDTH and PHYSICALHEIGHT.

- For GETPRINTINGOFFSET, the indexes are PHYSICALOFFSETX and PHYSICALOFFSETY.

- For GETSCALINGFACTOR, the indexes are SCALINGFACTORX and SCALINGFACTORY.

All six new indexes are defined in the file WINGDI.H, though they are missing from the GetDeviceCaps() documentation.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiPrn

Priority Inversion and Windows NT Scheduler

PSS ID Number: Q96418
Authored 16-Mar-1993 Last modified 13-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1 and 3.5

SUMMARY

The kernel schedules a thread with the real-time process priority class ahead of every thread with another priority class (nearly all user-mode threads). Windows NT does not alter the priority of real-time threads. The system trusts that the programmer will avoid priority inversion. The remainder of this article talks about the scheduling of threads that are not real-time priority class and how the system solves the problem of priority inversion.

Threads are scheduled according to their priority. When the kernel is choosing which thread will execute on a processor, the highest dynamic (variable) priority thread is picked. Priority inversion occurs when two (or more) threads with different priorities are in contention to be scheduled. Consider a simple case with three threads: Thread 1 is high priority and becomes ready to be scheduled, while thread 2, a low-priority thread, is executing in a critical section. Thread 1, the high-priority thread, begins waiting for a shared resource from thread 2. A third thread has medium priority. The third thread receives all the processor time, because the high-priority thread (thread 1) is busy waiting for shared resources from the low-priority thread (thread 2). Thread 2 won't leave the critical section, because it isn't the highest priority thread and won't be scheduled.

The Windows NT scheduler solves this problem by randomly boosting the priority of threads that are ready to run (in this case the low priority lock-holders). The low priority threads run long enough to let go of their lock (exit the critical section), and the high- priority thread gets the lock back. If the low-priority thread doesn't get enough CPU time to free its lock the first time, it will get another chance on the next scheduling round.

Priority inversion is handled differently in Windows 95. If a high priority thread is dependent on a low priority thread which will not be allowed to run because a medium priority thread is getting all of the CPU time, the system recognizes that the high priority thread is dependent on the low priority thread and will boost the low priority thread's priority up to the priority of the high priority thread. This will allow the formerly low priority thread to run and unblock the high priority thread that was waiting on it.

MORE INFORMATION

Each Process has a base priority. Each thread has a base priority that is a function of its process base priority. A thread's base priority is settable to:

- 1 or 2 points above the process base
- equal to the process base
- 1 or 2 points below the process base

Priority setting is exposed through the Win32 API. In addition to a base priority, all threads have a dynamic priority. The dynamic priority is never less than the base priority. The system raises and lowers the dynamic priority of a thread as needed.

All scheduling is done strictly by priority. The scheduler chooses the highest priority thread which is ready to run. On a multi-processor (MP) system, the highest N runnable threads run (where N is the number of processors). The thread priority used to make these decisions is the dynamic priority of the thread.

When a thread is scheduled, it is given a quantum of time in which to run. The quantum is in units of clock ticks. The system currently uses 2 units of quantum (10ms on r4000 and 15ms on x86).

When a thread is caught running during the clock interrupt, its quantum is decremented by one. If the quantum goes to zero and the thread's dynamic priority is not at the base priority, the thread's dynamic priority is decremented by one and the thread's quantum is replenished. If a priority change occurs, then the scheduler locates the highest priority thread which is ready to run. Otherwise, the thread is placed at the end of the run queue for it's priority allowing threads of equal priority to be "round robin" scheduled. The above is a description of what is usually called priority decay, or quantum and priority decay.

When a thread voluntarily waits (an an event, for I/O, etc), the system will usually raise the thread's dynamic priority when it resumes. Internally, each wait type has an associated priority boost. For example, a wait associated with disk I/O has a one point dynamic boost. A wait associated with a keyboard I/O has a 5 point dynamic boost. In most cases, this boost will raise the priority of the thread such that it can be scheduled very soon afterwards, if not immediately.

There are other circumstances under which priority will be raised. For example, whenever a window receives input (timer messages, mouse move messages, etc), an appropriate boost is given to all threads within the process that owns the window. This is the boost that allows a thread to reshape the mouse pointer when the mouse moves over a window.

By default, the foreground application has a base process priority that is one point higher than the background application. This allows the foreground process to be even more responsive. This can be changed by bringing up the System applet, selecting the Tasking button, and choosing a different option.

Additional reference words: 3.10 3.50
KBCategory: kbprg

KBSubcategory: BseProcThrd

Process Will Not Terminate Unless System Is In User-mode

PSS ID Number: Q92761
Authored 15-Nov-1992 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

Under Windows NT, a process will not be terminated unless the system is in user-mode. Suppose that TerminateProcess() is called while a device driver or filesystem code is being executed. The system will wait until the threads are running user code before marking the process for termination. On system exit, processes that were the target of a TerminateProcess() will be killed.

This may affect drivers. If a driver is waiting for an object or multiple objects in WaitMode or UserMode, its wait may complete unsuccessfully due to a termination request. Any code that does a UserMode wait or an Alertable wait must check the return status of the wait call. If the wait fails with STATUS_USER_APC or STATUS_ALERTED, this is not an error. The driver should cleanup and return to user-mode.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseProcThrd

Process WM_GETMINMAXINFO to Constrain Window Size

PSS ID Number: Q67166
Authored 22-Nov-1990 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

Microsoft Windows sends a WM_GETMINMAXINFO message to a window to determine the maximized size or position for the window, and the maximum or minimum tracking size for the window. An application can change these parameters by processing the WM_GETMINMAXINFO message.

Each window type has an absolute minimum size. If an application changes any of the values associated with WM_GETMINMAXINFO to a value smaller than the minimum, Windows will override the values specified by the application and use the minimum size. This minimum window size restriction has been removed from Windows version 3.1.

Note that Windows can send a WM_GETMINMAXINFO message to a window prior to sending a WM_CREATE message. Therefore, any processing for the WM_GETMINMAXINFO message must be independent of processing done for the WM_CREATE message.

MORE INFORMATION

An application can use the WM_GETMINMAXINFO message to constrain the size of a window. For example, the application can prevent the user from changing a window's width while allowing the user to affect its height, or vice versa. The following code demonstrates fixing the width:

```
  int width;
  LPPOINT lppt;
  RECT rect;

  case WM_GETMINMAXINFO:
      lppt = (LPPOINT)lParam;   // lParam points to array of POINTs

      GetWindowRect(hWnd, &rect);   // Get current window size
      width = rect.right - rect.left + 1;

      lppt[3].x = width   // Set minimum width to current width
      lppt[4].x = width   // Set maximum width to current width

      return DefWindowProc(hWnd, message, wParam, lParam);
```

The modifications required to fix the height are quite

straightforward.

For more information on the array of POINT structures that accompanies the WM_GETMINMAXINFO message, please refer to the "Microsoft Windows Software Development Kit Reference."

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrWndw

Processes Maintain Only One Current Directory

PSS ID Number: Q84244
Authored 05-May-1992 Last modified 21-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

Processes under Windows NT maintain only one current directory. Under MS-DOS or OS/2, a process will maintain a current directory for each drive.

MORE INFORMATION

For example, if you do the following

1. Set the current drive to be drive C and set the current directory to be \MAINC\MAINSUBC.

2. Change the current drive to be drive D and set the current directory to be \MAIND\MAINSUBD.

when you reset the current drive to drive C, the current directory will be the original directory: \MAINC\MAINSUBC.

MS-DOS and OS/2 use a current directory structure (CDS) to maintain this information. The memory for this structure is allocated at boot time, and is set by the LASTDRIVE= line in the CONFIG.SYS file. For example, if you set LASTDRIVE=Z, you will have 26 entries in the CDS and will be able to track 26 current directories.

Windows NT by default allows a process to track only one current directory--the one for the current drive--because the underlying operating system does not use drive letters; it always uses fully-qualified names such as:

```
\Device\HardDisk0\Partition1\autoexec.bat
```

The Win32 subsystem maintains drive letters by setting up symbolic links such as:

```
\DosDevices\C: == \Device\HardDisk0\Paritition1
\DosDevices\D: == \Device\HardDisk0\Paritition2
\DosDevices\E: == \Device\HardDisk1\Paritition1
```

(Partitions are 1-based while hard disks are 0-based because Partition0 refers to the entire physical device, which is the "file" that FDISK opens to do its work.) Therefore, when you do SetCurrentDirectory("c:\tmp\sub"), the Win32 subsystem translates that to "\DosDevices\c:\tmp\sub", "...".

As far as Windows NT is concerned, there are no "drives," there is one object namespace.

CMD.EXE maintains a private current directory for each drive it has touched and uses environment variables to associate a current directory with each drive. These environment variables have the form "=<drive>:".

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseFileio

Processing CBN_SELCHANGE Notification Message

PSS ID Number: Q66365
Authored 23-Oct-1990 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

When a combo box receives a CBN_SELCHANGE notification message, GetDlgItemText() will give the text of the previous selection and not the text of the new selection.

To get the text of the new selection, send the CB_GETCURSEL message to retrieve the index of the new selection and then send a CB_GETLBTEXT message to obtain the text of that item.

MORE INFORMATION

When an application receives the CBN_SELCHANGE notification message, the edit/static portion of the combo box has not been updated. To obtain the new selection, send a CB_GETLBTEXT message to the combo box control. This message places the text of the new selection in a specified buffer. The following is a brief code fragment:

```
   ...  /* other code */

   case CBN_SELCHANGE:
      hCombo = LOWORD(lParam);  /* Get combo box window handle */

    /* Get index of current selection and then the text of that selection
*/

      index = SendMessage(hCombo, CB_GETCURSEL, (WORD)0, 0L);
      SendMessage(hCombo, CB_GETLBTEXT, (WORD)index, (LONG)buffer);
      break;

   ...  /* other code */
```

NOTE: For Win32 applications, change the WORD and LONG casts to WPARAM and LPARAM, respectively.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95 combobox
KBCategory: kbprg
KBSubcategory: UsrCtl

Processing WM_PALETTECHANGED and WM_QUERYNEWPALETTE

PSS ID Number: Q77702
Authored 23-Oct-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows, versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

An application that manipulates the system palette should process the WM_PALETTECHANGED and WM_QUERYNEWPALETTE messages to maintain its appearance during system palette and input focus changes.

MORE INFORMATION

The WM_PALETTECHANGED message informs all windows that the window with input focus has realized its logical palette, thereby changing the system palette. This message allows a window without input focus that uses a color palette to realize its logical palettes and update its client area.

This message is sent to all windows, including the one that changed the system palette and caused this message to be sent. The wParam of this message contains the handle of the window that caused the system palette to change. To avoid an infinite loop, care must be taken to check that the wParam of this message does not match the window's handle. The following sample code demonstrates how to process WM_PALETTECHANGED:

```
    case WM_PALETTECHANGED:
        {
        HDC hDC;           // Handle to device context
        HPALETTE hOldPal;  // Handle to previous logical palette

        // If this application did not change the palette, select
        // and realize this application's palette
        if (wParam != hWnd)
           {
           // Need the window's DC for SelectPalette/RealizePalette
           hDC = GetDC(hWnd);

           // Select and realize hPalette
           hOldPal = SelectPalette(hDC, hPalette, FALSE);
           RealizePalette(hDC);

           // When updating the colors for an inactive window,
           // UpdateColors can be called because it is faster than
           // redrawing the client area (even though the results are
```

```
            // not as good)
            UpdateColors(hDC);

            // Clean up
            if (hOldPal)
               SelectPalette(hDC, hOldPal, FALSE);
            ReleaseDC(hWnd, hDC);
            }
         }
         break;
```

NOTE: The WM_PALETTECHANGED message is sent to all top-level and overlapped windows; therefore, if any child window uses a color palette, this message must be passed on to it.

The WM_QUERYNEWPALETTE message informs a window that it is about to receive input focus. In response, the window receiving focus should realize its palette as a foreground palette and update its client area. If the window realizes its palette, it should return TRUE; otherwise, it should return FALSE. The following sample code demonstrates processing WM_QUERYNEWPALETTE:

```
    case WM_QUERYNEWPALETTE:
        {
        HDC hDC;           // Handle to device context
        HPALETTE hOldPal;  // Handle to previous logical palette

        // Need the window's DC for SelectPalette/RealizePalette
        hDC = GetDC(hWnd);

        // Select and realize hPalette
        hOldPal = SelectPalette(hDC, hPalette, FALSE);
        RealizePalette(hDC);

        // Redraw the entire client area
        InvalidateRect(hWnd, NULL, TRUE);
        UpdateWindow(hWnd);

        // Clean up
        if (hOldPal)
           SelectPalette(hDC, hOldPal, FALSE);
        ReleaseDC(hWnd, hDC);

        // Message processed, return TRUE
        return TRUE;
        }
```

NOTE: The WM_QUERYNEWPALETTE message is sent to all top-level and overlapped windows; therefore, if any child window uses a color palette, this message must be passed on to it.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiPal

Programatically Appending Text to an Edit Control

PSS ID Number: Q109550
Authored 04-Jan-1994 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

Windows-based applications often use edit controls to display text. These applications sometimes need to append text to the end of an edit control instead of replacing the existing text. There are several ways to do this in Windows:

- Use the EM_SETEL and EM_REPLACESEL messages.

  -or-

- Use the EM_SETSEL message with the clipboard functions to append text to the edit control's buffer.

MORE INFORMATION

The EM_SETSEL message can be used to place a selected range of text in a Windows edit control. If the starting and ending positions of the range are set to the same position, no selection is made and a caret can be placed at that position. To place a caret at the end of the text in a Windows edit control and set the focus to the edit control, do the following:

```
HWND hEdit = GetDlgItem (hDlg, ID_EDIT);
int ndx = GetWindowTextLength (hEdit);
SetFocus (hEdit);
SendMessage (hEdit, EM_SETSEL, 0, MAKELONG (ndx, ndx));
```

Once the caret is placed at end in the edit control, the EM_REPLACESEL message can be use to append text to the edit control. An application sends an EM_REPLACESEL message to replace the current selection in an edit control with the text specified by the lpszReplace (lParam) parameter. Because there is no current selection, the replacement text is inserted at the current cursor location. This example sets the selection to the end of the edit control and inserts the text in the buffer:

```
SendMessage (hEdit, EM_SETSEL,     0, MAKELONG (ndx, ndx));
SendMessage (hEdit, EM_REPLACESEL, 0, (LPARAM) ((LPSTR) szBuffer));
```

One other way of inserting text into an edit control is to use the Windows clipboard. If the application has the clipboard open or finds it convenient to open the clipboard, and copies the text into the clipboard, then it can send the WM_PASTE message to the edit control to append text.

Before sending the WM_PASTE message, the caret must be placed at the end of the edit control text using the EM_SETSEL message. Below is pseudo code that shows how to implement this method:

```
   OpenClipBoard () ;
   EmptyClipBoard() ;
   SetClipBoardData() ;

   SendMessage (hEdit, EM_SETSEL,  0, MAKELONG (ndx, ndx));
   SendMessage (hEdit, WM_PASTE,   0, 0L);
```

This pseudo code appends text to the end of the edit control. Note that the data in the clipboard must be in CF_TEXT format.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Propagating Environment Variables to the System

PSS ID Number: Q104011
Authored 02-Sep-1993 Last modified 27-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

User environment variables can be modified using the System control panel application or by editing the following Registry key:

```
HKEY_CURRENT_USER \
      Environment
```

System environment variables can be modified using the System control panel application (in Windows NT 3.5 and later) or by editing the following Registry key:

```
HKEY_LOCAL_MACHINE \
            SYSTEM \
 CurrentControlSet \
          Control \
   Session Manager \
       Environment
```

Note, however, that modifications to the environment variables do not result in immediate change. For example, if you start another Command Prompt after making the changes, the environment variables will reflect the previous (not the current) values. The changes do not take effect until you log off and then log back on.

To effect these changes without having to log off, broadcast a WM_WININICHANGE message to all windows in the system, so that any interested applications (such as Program Manager, Task Manager, Control Panel, and so forth) can perform an update.

MORE INFORMATION

For example, on Windows NT, the following code fragment should propagate the changes to the environment variables used in the Command Prompt:

```
SendMessage( FindWindow( "Progman", NULL ), WM_WININICHANGE,
  0L, (LPARAM) "Environment" );
```

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMisc

Prototypes for SetSystemCursor() & LoadCursorFromFile()

PSS ID Number: Q122564
Authored 08-Nov-1994 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.5

SUMMARY

The function prototypes for SetSystemCursor() and LoadCursorFromFile() were inadvertently omitted from the Win32 SDK header files. These APIs are resolved by linking for USER32.LIB.

Additionally, the use of the OCR_* constants as described in the online help for LoadCursorFromFile() is not currently implemented. However, this functionality is available through LoadCursor().

MORE INFORMATION

The correct function prototypes are given below. NOTE: These prototypes were included correctly in the Win32 SDK 3.51/4.0 documentation.

To use these functions, add the prototypes to a file in your project after including WINDOWS.H.

```
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

/* SetSystemCursor prototype */
WINUSERAPI BOOL WINAPI SetSystemCursor (HCURSOR hcur, DWORD id);

/* LoadCursorFromFile prototypes - UNICODE aware */
WINUSERAPI HCURSOR WINAPI LoadCursorFromFileA (LPCSTR lpFileName);
WINUSERAPI HCURSOR WINAPI LoadCursorFromFileW (LPCWSTR lpFileName);

#ifdef UNICODE
#define LoadCursorFromFile  LoadCursorFromFileW
#else
#define LoadCursorFromFile  LoadCursorFromFileA
#endif // !UNICODE

#ifdef __cplusplus
}
#endif  /* __cplusplus */
```

Additional reference words: 3.50
KBCategory: kbprg kbdocerr
KBSubcategory: GdiCurico

Providing a Custom Wordbreak Function in Edit Controls

PSS ID Number: Q109551
Authored 04-Jan-1994 Last modified 13-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

An application sends the EM_SETWORDBREAKPROC message to an edit control to replace the default wordwrap function with an application-defined wordwrap function. The default wordwrap function breaks a line in a multiline edit control (MLE) at a space character. If an application needs to change this functionality (that is, to break at some character other than a space), then the application must provide its own wordwrap (wordbreak) function.

MORE INFORMATION

A wordwrap function scans a text buffer (which contains text to be sent to the display), looking for the first word that does not fit on the current display line. The wordwrap function places this word at the beginning of the next line on the display.

A wordwrap function defines the point at which Windows should break a line of text for multiline edit controls, usually at a space character that separates two words. This can be changed so that the line in an MLE can be broken at any character. For more information on the EM_SETWORDBREAKPROC message, please refer to the Windows 3.1 SDK "Programmer's Reference, Volume 3: Messages, Structures, and Macros" manual.

Below is sample code that demonstrates how to break a line in a multiline edit control at the "~" (tilde) character (for example) instead of the regular space (" ") character.

The sample code assumes that the edit control is a multiline edit control and that it is a child control in a dialog box.

Sample Code

```
//Prototype the application-defined wordbreakproc.
int CALLBACK WordBreakProc(LPSTR, int, int, int) ;

//Install wordbreakproc in the WM_INITDIALOG case.
  case WM_INITDIALOG:

    lpWrdBrkProc = MakeProcInstance(WordBreakProc, hInst);
```

```
    //Send the EM_SETWORDBREAKPROC message to the edit control
    //to install the new wordbreak procedure.
     SendDlgItemMessage(hDlg, ID_EDIT, EM_SETWORDBREAKPROC, 0,
                        (LPARAM)(EDITWORDBREAKPROC)lpWrdBrkProc) ;
     return (TRUE);

int FAR PASCAL WordBreakProc(LPSTR lpszEditText, int ichCurrent,
                             int cchEditText, int wActionCode)
{

   char FAR *lpCurrentChar;
   int  nIndex;
   int  nLastAction;

   switch (wActionCode) {

     case WB_ISDELIMITER:

     // Windows sends this code so that the wordbreak function can
     // check to see if the current character is the delimiter.
     // If so, return TRUE. This will cause a line break at the ~
     // character.

      if ( lpszEditText[ichCurrent] == '~' )
         return TRUE;
      else
         return FALSE;

      break;

     // Because we have replaced the default wordbreak procedure, our
     // wordbreak procedure must provide the other standard features in
     // edit controls.

     case WB_LEFT:

     // Windows sends this code when the user enters CTRL+LEFT ARROW.
     // The wordbreak function should scan the text buffer for the
     // beginning of the word from the current position and move the
     // caret to the beginning of the word.

         {
            BOOL bCharFound = FALSE;

            lpCurrentChar = lpszEditText + ichCurrent;
            nIndex = ichCurrent;

            while (nIndex > 0  &&
                  (*(lpCurrentChar-1) != '~' &&
                   *(lpCurrentChar-1) != 0x0A) ||
                  !bCharFound )
            {
               lpCurrentChar = AnsiPrev(lpszEditText ,lpCurrentChar);
               nIndex--;
```

```
                if (*(lpCurrentChar) != '~' &&  *(lpCurrentChar) != 0x0A )

                   // We have found the last char in the word. Continue
                   // looking backwards till we find the first char of
                   // the word.
                  {
                    bCharFound = TRUE;

                    // We will consider a CR the start of a word.
                    if (*(lpCurrentChar) == 0x0D)
                        break;
                  }

              }
              return nIndex;

          }
       break;

      case WB_RIGHT:

      //Windows sends this code when the user enters CTRL+RIGHT ARROW.
      //The wordbreak function should scan the text buffer for the
      //beginning of the word from the current position and move the
      //caret to the end of the word.

       for (lpCurrentChar = lpszEditText+ichCurrent, nIndex = ichCurrent;
            nIndex < cchEditText;
            nIndex++, lpCurrentChar=AnsiNext(lpCurrentChar))

        if ( *lpCurrentChar == '~' ) {
       lpCurrentChar=AnsiNext(lpCurrentChar);
          nIndex++;

          while ( *lpCurrentChar == '~' ) {
            lpCurrentChar=AnsiNext(lpCurrentChar);
            nIndex++;
           }

          return nIndex;
        }

        return cchEditText;
        break;

  }
}
```

The wordwrap (wordbreak) function above needs to be exported in the .DEF file of the application. The function can be modified and customized according to the application's needs.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95 multi-line
KBCategory: kbprg
KBSubcategory: UsrCtl

PSTR's in OUTLINETEXTMETRIC Structure

PSS ID Number: Q90085
Authored 08-Oct-1992 Last modified 05-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

The OUTLINETEXTMETRIC structure ends with four fields of type PSTR. The four fields in question are not actually absolute pointers. They are offsets from the beginning of the OUTLINETEXTMETRIC structure to the strings in question, as the documentation indicates:

otmpFamilyName
Specifies the offset from the beginning of the structure to a string specifying the family name for the font.

otmpFaceName
Specifies the offset from the beginning of the structure to a string specifying the face name for the font. (This face name corresponds to the name specified in the LOGFONT structure.)

otmpStyleName
Specifies the offset from the beginning of the structure to a string specifying the style name for the font.

otmpFullName
Specifies the offset from the beginning of the structure to a string specifying the full name for the font. This name is unique for the font and often contains a version number or other identifying information.

The only difference between this structure in Windows 3.1 and Windows NT is that the strings may be stored in either Unicode or ASCII under Windows NT.

Additional reference words: 3.10 3.50 4.00
KBCategory: kbprg
KBSubcategory: GdiMisc

Querying and Modifying the States of System Menu Items

PSS ID Number: Q83453
Authored 13-Apr-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

An application should query or set states of the Restore, Move, Size, Minimize, and Maximize items on the system menu during the processing of a WM_INITMENU or a WM_INITMENUPOPUP message.

MORE INFORMATION

Windows changes the state of the Restore, Move, Size, Minimize, and Maximize items on the system menu just before it draws the menu on the screen and sends the WM_INITMENU and WM_INITMENUPOPUP messages.

Windows sets the states of these menu items according to the state of the window just before the menu is displayed. For example, if the window is minimized when its system menu is pulled down, the Minimize menu item is unavailable (grayed). If an overlapped window is maximized when its system menu is pulled down, the Move, Size, and Maximize items are unavailable.

If an application queries or sets the state of any of these system menu items, the query or change should occur during the processing of the WM_INITMENU or WM_INITMENUPOPUP message. If any menu item state is queried before one of these messages is processed, it could reflect a previous state of the window. If any state is set before one of these messages is processed, Windows will reset the menu items to correspond to the state of the window just prior to sending these messages.

Windows does not change the state of the Close menu item. Its state can be changed or queried at any time.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMen

Querying Device Support for MaskBlt

PSS ID Number: Q108929
Authored 20-Dec-1993 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, and 3.51

The Win32 documentation for MaskBlt() states:

> Not all devices support the MaskBlt function. An application should call the GetDeviceCaps function to determine whether a device supports this function.

To query support for MaskBlt(), an application should query the device for BitBlt support by passing RC_BITBLT constant to GetDeviceCaps().

MaskBlt() implements transparent blts in Windows NT. For more information, please see the following article in the Microsoft Knowledge Base:

> ARTICLE-ID: Q89375
> TITLE : Transparent Blts in Windows NT

GDI implements this application programming interface (API) by calling BitBlt(). Because BitBlt() is implemented at the driver level, applications that calls MaskBlt() should check for BitBlt() support on the device.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: GdiBmp

Raster and Stroke Fonts; GDI and Device Fonts

PSS ID Number: Q77126
Authored 08-Oct-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

NOTE: The information contained in this article does not address TrueType fonts. For information on TrueType fonts, please see chapter 18 of "Guide to Programming" for the Windows SDK version 3.1.

In Windows version 3.0, there are two different ways that the graphical device interface (GDI) can generate characters for a font. For a raster font, GDI displays the font by copying bitmaps to the output device. For a stroke font, GDI displays the font by drawing lines between a series of points that describe each character. Each font is owned by either GDI or by a specific device. Type and ownership information can be determined by enumerating the fonts. This article discusses the two font types and two font ownership types.

MORE INFORMATION

A raster font stores its characters as a series of bitmaps; a stroke font stores its characters as a set of vector operations that describe the characters. When a character from a raster font is drawn, the bitmap is copied onto the device. When a character from a stroke font is displayed, the lines are drawn connecting the points that describe the character. Examples of raster fonts provided with Windows are Courier and Helv; examples of stroke fonts are Script and Roman.

Raster fonts are only available in specific sizes. Some devices can scale installed raster fonts to integer multiples of their size. Use the GetDeviceCaps() function to determine whether the device has this capability. The Windows GDI will scale its raster fonts as required regardless of the device capability. Stroke fonts can be scaled to any size and can also be rotated.

GDI fonts are owned by the GDI; they are available to all devices. Device fonts are fonts that are owned by a particular device; they are available only on that device.

By enumerating the fonts, an application can determine which ones are raster or stroke fonts, and which are GDI or device fonts. The callback function used with EnumFonts() has the parameter nFontType. As stated on page 4-118 of the "Microsoft Windows Software Development Kit Reference Volume 1," the bitwise AND (&) operator can be used with the constants

RASTER_FONTTYPE and DEVICE_FONTTYPE to determine the font type. If the RASTER_FONTTYPE bit is set, the font is a raster font; otherwise, it is a stroke font. If the DEVICE_FONTTYPE bit is set, the font is owned by the device that corresponds to the display context handle (HDC) used in the EnumFont() call; otherwise it is a GDI font.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiFnt

RCDATA Begins on 32-Bit Boundary in Win32

PSS ID Number: Q84081
Authored 29-Apr-1992 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

RCDATA is guaranteed to begin on a DWORD boundary. However, the strings and the integers specified in the statement are not aligned by the Resource Compiler (RC)

RCDATA statement:

```
resname RCDATA
BEGIN

    0,0,

END
```

The definition of RCDATA is not changed. The strings and integers specified in the statement, 0 in this case, are not aligned on the DWORD boundary. However, the beginning of the data is DWORD-aligned.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsRc

ReadFile() at EOF Changed in Windows NT 3.5

PSS ID Number: Q119220
Authored 10-Aug-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5

The documentation for ReadFile() states that:

> If the function succeeds, the return value is TRUE. If the return value is TRUE and the number of bytes read is zero, the file pointer was beyond the current end of file at the time of the read operation.
>
> If the function fails, the return value is FALSE. To get extended error information, call GetLastError().

Under Windows NT 3.1, when you read at the end of the file (EOF), ReadFile() returns TRUE and reports that 0 (zero) bytes were read. However, if you are using asynchronous I/O and EOF is reached, GetOverlappedResult() will hang because there is no outstanding I/O. ReadFileEx() properly handles EOF by failing and setting the error code to ERROR_HANDLE_EOF.

Under Windows NT 3.5, ReadFile() returns FALSE and GetLastError() returns error 38 (ERROR_HANDLE_EOF) for asynchronous I/O. This change in behavior was made to avoid the problem described above. The behavior for synchronous I/O has not changed under Windows NT 3.5.

Additional reference words: 3.50
KBCategory: kbprg
KBSubcategory: BseFileio

Reasons for Failure of Clipboard Functions

PSS ID Number: Q92530
Authored 09-Nov-1992 Last modified 13-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

The following clipboard functions:

```
OpenClipboard()
CloseClipboard()
EmptyClipboard()
GetClipboardData()
SetClipboardData()
EnumClipboardFormats()
SetClipboardViewer()
ChangeClipboardChain()
GetOpenClipboardWindow()
GetClipboardOwner()
```

can fail for several reasons. Different functions return different values to indicate failure. Read the documentation for information about each function. This article combines the causes of failure for all functions and provides a resolution or explanation. In the More Information section, a list of affected functions follows each cause. The causes are:

1. The clipboard is not opened by any application.

2. The current application does not have the clipboard open.

3. The current application does not own the clipboard.

4. User's data segment is full.

5. Insufficient global memory.

6. The specified clipboard format is not supported.

7. The application that set the clipboard data placed a corrupt or invalid metafile in the clipboard.

8. An application is attempting to open an already open clipboard. The debug mode of Windows 3.1 will send the "Clipboard already open" message.

9. The application that opened the clipboard used NULL as the window handle.

MORE INFORMATION

Cause 1: The clipboard is not opened by any application.

Resolution 1: Open the clipboard using OpenClipboard(). If a DLL needs to open the clipboard, it may pass hwnd = NULL to OpenClipboard().

Explanation 1: An application cannot copy data (using SetClipboardData()) when no application has the clipboard currently open.

Affected Functions: SetClipboardData().

Cause 2: The current application does not have the clipboard open.

Resolution 2: Open the clipboard using OpenClipboard(). If a DLL needs to open the clipboard, it may pass hwnd = NULL to OpenClipboard().

Explanation 2: An application cannot empty or close the clipboard without first opening it.

Affected Functions: EmptyClipboard(), CloseClipboard().

Cause 3: The current application does not own the clipboard.

Resolution 3: Open the clipboard and get ownership by emptying it.

Explanation 3: An application cannot enumerate the clipboard formats without owning it.

Affected Functions: EnumClipboardFormats().

Cause 4: User's data segment is full.

Explanation 4: There should be space available in User's data segment to store internal data structures when SetClipboardData() is called.

Affected Function: SetClipboardData().

Cause 5: Insufficient global memory.

Explanation 5: If the clipboard has data in either the CF_TEXT or CF_OEMTEXT format and if GetClipboardData() requests text in the unavailable format, then Windows will perform the conversion. The converted text must be stored in global memory.

Affected Function: GetClipboardData().

Cause 6: The specified clipboard format is not supported.

Resolution 6: Use IsClipboardFormatAvailable() to check whether the specified format is available on the clipboard.

Affected Function: GetClipboardData().

Cause 7: The application that set the clipboard data placed a corrupt or invalid metafile in the clipboard.

Resolution 7: There are no functions to tell whether a given metafile is corrupt or invalid. Try playing the metafile and see if the metafile plays as expected.

Affected Function: SetClipboardData().

Cause 8: Application is attempting to open an already open clipboard. The debug mode of Windows 3.1 will send the "Clipboard already open" message.

Explanation 8: The clipboard must be closed by the application that opened it, before other applications can open it.

Affected Functions: OpenClipboard().

Cause 9: The application that opened the clipboard used NULL as the window handle.

Explanation 9: An application can call OpenClipboard(NULL) to successfully open a clipboard. The side effects are that subsequent calls to GetClipboardOwner() and GetOpenClipboardWindow() return NULL. An application can also call SetClipboardViewer(NULL) successfully. However, there is no reason why this should be allowed, and it is currently reported as a bug. The side effects are that subsequent calls to GetClipboardViewer() and ChangeClipboardChain() return NULL. NULL from these functions does not necessarily imply that they failed.

Affected Functions: GetClipboardOwner(), GetOpenClipboardWindow(), GetClipboardViewer(), ChangeClipboardChain().

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
GetClipboardFormatName
RegisterClipboardFormat
KBCategory: kbprg
KBSubcategory: UsrClp

Reasons for Failure of Menu Functions

PSS ID Number: Q89739
Authored 29-Sep-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

The Menu functions (AppendMenu(), CheckMenuItem(), CreateMenu(), CreatePopupMenu(), DeleteMenu(), DestroyMenu(), GetMenu(), GetMenuItemID(), GetMenuString(), GetSubMenu(), GetSystemMenu(), HiliteMenuItem(), InsertMenu(), LoadMenuIndirect(), ModifyMenu(), RemoveMenu(), SetMenu(), SetMenuItemBitmaps(), and TrackPopupMenu()) can fail for several reasons. Different functions return different values to indicate failure. Read the documentation for information about each function. This article combines the causes of failure for all functions and provides a resolution or explanation. A list of affected functions follows each cause. The causes are:

1. Invalid hWnd parameter.

2. Invalid hMenu parameter.

3. The menu item is not found.

4. No space left in User's heap to hold a string or to hold an internal data structure for owner draw menu items or to create a menu or to create a window for TrackPopupMenu().

5. There are no items in the menu.

6. The menu resource could not be found (FindResource()) or loaded (LoadResource()) or locked (LockResource()) in memory.

7. TrackPopupMenu() is called while another popup menu is being tracked in the system.

8. The hMenu that has been passed to TrackPopupMenu() has been deleted.

9. MENUITEMTEMPLATEHEADER's versionNumber field is non-zero.

MORE INFORMATION

Cause 1: Invalid hWnd parameter.

Resolution 1: Validate the hWnd parameter using IsWindow(). Make sure that hWnd is not a child window.

NOTE: Resolution 1 does not apply to TrackPoupuMenu().

Explanation 1: In Windows, menus are always associated with a window. Child windows cannot have menu bars.

Affected Functions: All functions that take hWnd as a parameter except for TrackPoupuMenu().

Cause 2: Invalid hMenu parameter.

Resolution 2: Validate hMenu with IsMenu().

Affected Functions: All functions that take hMenu as a parameter.

Cause 3: The menu item is not found.

Resolution 3: If the menu item is referred to BY_POSITION, make sure that the index is lesser than the number of items. If the menu item is referred to BY_COMMAND, an application has to devise its own method of validating it.

Explanation 3: Menu items are numbered consecutively starting from 0. Remember that separator items are also counted.

Affected Functions: All functions that refer to a menu item.

Cause 4: No space left in User's heap to hold a string or to hold an internal data structure for owner draw menu items or to create a menu.

Resolution 4: Remember to delete all menus and other objects that have been created by the application when they are not needed any more. If you suspect that objects left undeleted by other applications are wasting valuable system resources, restart Windows.

Explanation 4: In Windows 3.0, menus and menu items were allocated space from User's heap. In Windows 3.1, they are allocated space from a separate heap. This heap is for the exclusive use of menus and menu items.

Affected Functions: AppendMenu(), Insertmenu(), ModifyMenu(), CreateMenu(), CreatePopupMenu(), LoadMenu(), LoadMenuIndirect(), TrackPopupMenu(), GetSystemMenu() (when fRevert = FALSE).

Cause 5: There are no items in the menu.

Resolution 5: Use GetMenuItemCount() to make sure the menu is not empty.

Explanation 5: Nothing to be deleted or removed.

Affected Functions : RemoveMenu(), DeleteMenu().

Cause 6: The menu resource could not be found (FindResource()) or loaded (LoadResource()) or locked (LockResource()) in memory.

Resolution 6: Ensure that the menu resource exists and that the hInst

parameter refers to the correct hInstance. Try increasing the number of file handles using SetHandleCount() and increasing available global memory by closing some applications. For more information about the causes of failure of resource functions, query this Knowledge Base on the following keywords:

   failure and LoadResource and FindResource and LockResource.

Explanation 6: Finding, loading, and locking a resource involves use of file handles, global memory, and the hInstance that has the menu resource.

Affected Functions: LoadMenu(), LoadMenuIndirect()

Cause 7. TrackPopupMenu() is called while another popup menu is being tracked in the system.

Explanation 7: Only one popup menu can be tracked in the system at any given time.

Affected Function: TrackPopupMenu()

Cause 8. The hMenu that has been passed to TrackPopupMenu() has been deleted. The debug mode of Windows 3.1 sends the following message :

"Menu destroyed unexpectedly by WM_INITMENUPOPUP"

Explanation 8: Windows sends a WM_INITMENUPOPUP to the application and expects the menu to not be destroyed.

Affected Function: TrackPopupMenu()

Cause 9. MENUITEMTEMPLATEHEADER 's versionNumber field is non-zero.

Explanation 9: In Windows 3.0 and 3.1, this field should always be 0.

Affected Function: LoadMenuIndirect()

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMen

Reasons Why RegisterClass() and CreateWindow() Fail

PSS ID Number: Q65257
Authored 28-Aug-1990 Last modified 10-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows, versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The RegisterClass() and CreateWindow() functions fail when the system resources are used up. The percentage of free system resources reflects the amount of available space in the USER and GDI heaps within Windows. The smaller amount of free space is reported in the Program Manager's About box because if either heap fills up, functions fail.

Under Windows NT, the USER and GDI heap resources are practically unlimited. Under Windows 95, the USER and GDI heap resources are greater than Windows 3.1, but not as great as under Windows NT.

MORE INFORMATION

If the amount of free system resources remains low after the application is exited, it is more likely that the GDI heap is filling. The main reason for the GDI heap filling is that GDI objects that are created by the application are not deleted or destroyed when they are no longer needed, or when the program terminates. Windows does not delete GDI objects (pens, brushes, fonts, regions, and bitmaps) when the program exits. Objects must be properly deleted or destroyed.

NOTE: Win32-based applications cannot cause the USER or GDI heaps to overflow when they terminate, because the system will release the resources to maximize available resources.

The following are two situations that can cause the USER heap to get full:

1. Memory is allocated for "extra bytes" associated with window classes and windows themselves. Make sure that the cbClsExtra and cbWndExtra fields in the WNDCLASS structure are set to 0 (zero), unless they really are being used.

2. Menus are stored in the USER heap. If menus are added but are not destroyed when they are no longer needed, or when the application terminates, system resources will go down.

CreateWindow() will also fail under the following conditions:

1. Windows cannot find the window procedure listed in the CreateWindow() call. Avoid this by ensuring that each window procedure is listed in the EXPORTS section of the program's DEF file.

2. CreateWindow() cannot find the specified window class.

3. The hwndparent is incorrect (make sure to use debug Windows to see the RIPs).

4. CreateWindow() cannot allocate memory for internal structures in USER heap.

5. The application returns 0 (zero) to the WM_NCCREATE message.

6. The application returns -1 to the WM_CREATE message.

Additional reference words: 3.00 3.10 3.50 4.00
KBCategory: kbprg
KBSubcategory: UsrWndw

Reducing the Count on a Semaphore Object

PSS ID Number: Q94997
Authored 28-Jan-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

ReleaseSemaphore(), which increments the count of a given semaphore object by a specified amount, will not take a negative value.

If, for some reason, you want to reduce the number of available semaphore "slots" to temporarily restrict or reduce access, you may loop calling WaitForSingleObject() with a zero timeout, counting the number of times it succeeds. When you no longer need to hold the semaphore slots, call ReleaseSemaphore() with the number of slots counted.

Note that this method does not prevent other threads from taking a semaphore slot when your thread is looping.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseMisc

Registering Multiple RPC Server Interfaces

PSS ID Number: Q129975
Authored 09-May-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, version 3.5

When registering multiple server interfaces from multiple threads or from a single thread under the same process, the RpcServerListen() function should be called only once.

An application process may register two completely separate Remote Procedure Call (RPC) server interfaces from two separate threads or from a single thread. However, when doing so, the RpcServerListen() function should be called only once from any thread.

RPC APIs are called on a per process basis. From the perspective of RPC run times, a process can register multiple interfaces using one or more protocol sequences, but must call the RpcServerListen() function only once from any one thread. Calling the RpcServerListen() function more than once results in the run time generating an exception called RPC_S_ALREADY_LISTENING.

Additional reference words: 3.50
KBCategory: kbprg kbnetwork
KBSubcategory: NtwkRpc

RegSaveKey() Requires SeBackupPrivilege

PSS ID Number: Q106383
Authored 07-Nov-1993 Last modified 20-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

The description for RegSaveKey() states the following:

> The caller of this function must possess the SeBackupPrivilege security privilege.

This means that the application must explicitly open a security token and enable the SeBackupPrivilege. By granting a particular user the right to back up files, you give that user the right only to gain access to the security token (that is, the token is not automatically created for the user but the right to create such a token is given). You must add additional code to open the token and enable the privilege.

MORE INFORMATION

The following code demonstrates how to enable SeBackupPrivilege:

```
static HANDLE           hToken;
static TOKEN_PRIVILEGES tp;
static LUID             luid;

// Enable backup privilege.

OpenProcessToken( GetCurrentProcess(),
   TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, &hToken ) ;
LookupPrivilegeValue( NULL, "SeBackupPrivilege", &luid );
tp.PrivilegeCount           = 1;
tp.Privileges[0].Luid       = luid;
tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
AdjustTokenPrivileges( hToken, FALSE, &tp,
   sizeof(TOKEN_PRIVILEGES), NULL, NULL );

// Insert your code here to save the registry keys/subkeys.

// Disable backup privilege.

AdjustTokenPrivileges( hToken, TRUE, &tp, sizeof(TOKEN_PRIVILEGES),
   NULL, NULL );
```

Note that you cannot create a process token; you must open the existing process token and adjust its privileges.

The DDEML Clock sample has similar code sample at the end of the CLOCK.C file where it obtains the SeSystemTimePrivilege so that it can set the system time.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseMisc

Removing Focus from a Control When Mouse Released Outside

PSS ID Number: Q66947
Authored 14-Nov-1990 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

Under normal circumstances, when you move the mouse cursor into the client area of a child-window control, click it, and then release the mouse button, the child window sends a WM_COMMAND message to its parent and retains the focus.

If you move the mouse into the client area of the child-window control, press the mouse button, move the mouse cursor out of the client area of the control, and then release the mouse button, the control does not send a WM_COMMAND message. However, the control retains the focus.

If you do not want the control to retain the focus, you can remove it by performing the following steps:

1. Define a static Boolean flag in the parent window function.

2. When a WM_PARENTNOTIFY message is received, set the flag to TRUE. This indicates that the mouse button has been pressed while the mouse cursor is in the client area of the control.

3. If a WM_COMMAND message is received, reset the flag to FALSE and perform normal processing.

4. Otherwise, if a WM_MOUSEMOVE message is received, the mouse button was released after the mouse cursor was moved outside the control. Reset the flag to FALSE, and use SetFocus() to move the focus to the desired window.

MORE INFORMATION

When the mouse cursor is in the client area of a control and you press the mouse button, the parent window will receive a WM_PARENTNOTIFY message and a WM_MOUSEACTIVATE message. A Boolean (BOOL) flag should be set when the message is processed to indicate that this occurred.

The parent window will receive other messages, including a number of WM_CTLCOLOR messages, when the mouse is moved around with the mouse button down. When the mouse button is released, the parent window receives only one of two messages:

1. WM_COMMAND: The mouse button was released over the control.

2. WM_MOUSEMOVE: The mouse button was released outside the control.

Note that these are not the only messages received when the button is released, but these two are mutually exclusive.

In response to either message, the following steps must take place:

1. Reset the flag indicating a mouse press.

2. Call SetFocus() or send a WM_KILLFOCUS to the control in question to move the focus as desired.

If WM_KILLFOCUS is used, the ID of the control or its handle must be known. SetFocus(NULL) or SetFocus(hWndParent) removes the focus from the control but does not set the focus to any other control in the window.

In a dialog box, SetFocus(NULL) MUST be used. SetFocus(hDlg) does not remove the focus from the button.

The following code sample is taken from the dialog box procedure of a dialog that has a single OK button. If the mouse button is pressed while the mouse cursor is over the button, the mouse is moved outside the button, and then the mouse button is released, the focus is removed from the OK button.

```
BOOL FAR PASCAL AboutProc(HWND hDlg, unsigned iMessage,
                          WORD wParam, LONG lParam)
    {
    static BOOL fMousePress;

    switch (iMessage)
        {
        case WM_INITDIALOG:
            fMousePress = FALSE;
            return TRUE;

        case WM_PARENTNOTIFY:  // or WM_MOUSEACTIVATE
            fMousePress = TRUE;
            break;

        case WM_MOUSEMOVE:
            if (fMousePress)
                SetFocus(NULL);
            fMousePress = FALSE;
            break;

        // Only command is the OK button.
        case WM_COMMAND:
            if (wParam == IDOK)
                EndDialog(hDlg, TRUE);
            break;
        }
    return FALSE;
    }
```

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbcode
KBSubcategory: UsrCtl

Replace IsTask() with GetExitCodeProcess()

PSS ID Number: Q108228
Authored 07-Dec-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

In Windows 3.1, the application programming interface (API) IsTask() can be used to determine whether a process is still running or whether it has terminated. As the help file indicates, this function is obsolete in the Win32 API.

To get this functionality through the Win32 API, use the API GetExitCodeProcess(). This function takes the handle as the first parameter and returns the exit code or STILL_ACTIVE in the second parameter:

```
BOOL GetExitCodeProcess(hProcess, lpdwExitCode)

HANDLE hProcess;
LPDWORD lpdwExitCode;
```

As an alternative, you can also use WaitForSingleObject(). Pass the process handle as the first parameter and a timeout value for the second parameter:

```
DWORD WaitForSingleObject(hObject, dwTimeout)

HANDLE hObject;
DWORD dwTimeout;
```

The process handle is signaled when the process terminates. Pass in 0 (zero) for the timeout if you would like to poll or start another thread to wait with an INFINITE timeout value.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseProcThrd

Replacing the Shell (Program Manager)

PSS ID Number: Q100328
Authored 20-Jun-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

To replace the current shell, change the following registry key:

```
HKEY_LOCAL_MACHINE\
   SOFTWARE\
   Microsoft\
   Windows NT\
   CurrentVersion\
   Winlogon\
   Shell
```

Note that Program Manager combines the functionality of Program Manager and Task Manager (the Task Manager installed is not actually run). Therefore, you must take this into account. In Windows NT 3.1, if the new shell does not replace the Task Manager functionality, the replacement string should contain both the new shell name and TASKMAN.EXE, separated by commas. In Windows NT 3.5, the new shell should either spawn TASKMAN.EXE or your own task manager, specified in

```
HKEY_LOCAL_MACHINE\
   SOFTWARE\
   Microsoft\
   Windows NT\
   CurrentVersion\
   Winlogon\
   Taskman
```

The value does not exist by default, it must be added. The value type is REG_SZ.

To update the string that is retrieved when you call GetPrivateProfileString(), change the string in the following registry key:

```
HKEY_LOCAL_MACHINE\
   SOFTWARE\
   MICROSOFT\
   Windows NT\
   Current Version\
   WOW\
   Boot\
   Shell
```

The duplicate entry is for compatibility with Windows 3.1.

MORE INFORMATION

WritePrivateProfileString() changes the following registry key:

```
   HKEY_LOCAL_MACHINE\
      SOFTWARE\
      Microsoft\
      Windows NT\
      CurrentVersion\
      WOW\
      Boot\
      Shell
```

It does not have the desired effect of actually changing the Shell.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseMisc

Replacing the Windows NT Task Manager

PSS ID Number: Q89373
Authored 21-Sep-1992 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

Under Windows 3.1, the Task Manager is an easily replaceable program. However, under the Win32 subsystem of Windows NT, it is very difficult to replace the Task Manager, due to special programming considerations. In general, it is not recommended that users attempt this.

MORE INFORMATION

Special requirements for the Task Manager make it very different from the Windows 3.1 Task Manager. The EndTask button handling is done through internal application programming interface (API) functions. These API functions are not documented. The situation is the same for handling foreground management, hung applications, and priority issues (to make sure that the Task Manager will come up as fast as possible). In addition, the Windows NT Task Manager uses shortcut ("hot") keys.

In Windows NT 3.1, the Task List has been incorporated into the Program Manager. To remove the Task List, you must also remove the Program Manager. The full functionality of the Task List (as found in TASKMAN.EXE) is now folded into the Program Manager (PROGMAN.EXE). If you completely delete the TASKMAN.EXE file from your system and from the registry location

```
HKEY_LOCAL_MACHINE\
   SOFTWARE\
   Microsoft\
   Windows NT\
   CurrentVersion\
   WinLogon\
   Shell
```

you will still be able to invoke the Task List because it is built into the Program Manager.

In Windows NT 3.5, Program Manager checks the registry for a Taskman entry. If the Taskman entry is found, Program Manager will launch the application, instead of using its built-in Taskman. The registry entry is:

```
HKEY_LOCAL_MACHINE\
   SOFTWARE\
   Microsoft\
   Windows NT\
```

```
      CurrentVersion\
      Winlogon\
      Taskman
```

This entry does not not exist by default. You will have to create this value, with type REG_SZ.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseMisc

Replacing Windows NT Control Panel's Mouse Applet

PSS ID Number: Q110704
Authored 27-Jan-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

The Control Panel includes a Mouse applet as a standard applet that is shipped with the system. In Windows 3.1, this applet can be overridden by an ISV/OEM's mouse driver or module. Windows 3.1 accomplished this by the Control Panel doing a GetModuleHandle() call on "Mouse". If MOUSE.DLL or MOUSE.EXE was already loaded in the system, Control Panel would look for the entry point "CPlApplet". If found, Control Panel would send the following messages:

CPL_NEWINQUIRE

-or-

CPL_INQUIRE (the former is preferred)

This would return the icon and strings to replace the mouse icon already in Control Panel.

In Windows NT, because of the separation of Kernel drivers from applications by the client-server interface, the GetModuleHandle() call does not work. Consequently, the same functionality must be achieved in a slightly different way. The Control Panel calls LoadLibrary("Mouse") to look for a MOUSE.DLL or a MOUSE.EXE. If this call fails, no other checks are made.

If LoadLibrary() succeeds, the Control Panel looks for the "CPlApplet" entry point, sends a CPL_INIT message, and then sends a CPL_NEWINQUIRE. If CPL_NEWINQUIRE fails, a CPL_INQUIRE is sent; however, it is preferable to have the applet implement the newer CPL_NEWINQUIRE message. The string information returned by the CPL_NEWINQUIRE message can be in either UNICODE or ANSI (UNICODE is preferred) as long as the dwSize field is set correctly. See the CPL.H public header file for these messages and structures (for example CPLINFOW or CPLINFOA, where CPLINFOW is the default).

When the User double-clicks the Mouse applet icon, the MOUSE.DLL or MOUSE.EXE will receive a CPL_DBLCLK message from the CPlApplet interface. This routine must return TRUE to the Control Panel if the routine runs its own dialog. If FALSE is returned, the internal Mouse Dialog box will be presented to the User.

The Control Panel will send the CPL_EXIT message to the Mouse applet when it wants to unload the module or terminate. The applet must use this message to perform tasks such as calling UnRegisterWindowClass(), freeing memory, and unloading DLLs.

NOTE: It is not possible to replace any of the other standard Windows NT 3.1 Control Panel applets. As of Windows NT 3.5, it is also possible to replace the Keyboard applet in the same manner.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: UsrMisc

Resource Sections are Read-only

PSS ID Number: Q126630
Authored 27-Feb-1995 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.5, 3.51, and 4.0
- Microsoft Win32s versions 1.2

SUMMARY

Resource sections are read-only by default under Windows NT and Win32s. However, under Windows NT, the resource section may appear to be read/write. If an application tries to write to the resource section, an exception occurs. Windows NT handles the exception by duplicating the page and making it read/write. Therefore, under Windows NT it is possible to write to the resource section, even though its section attribute is read-only.

In Win32s version 1.2, the resource section is read-only and you cannot write to it. To work around this, link with /SECTION:.rsrc,rw to make the resource section read/write. Or copy the resource to your own buffer and work with it from there. You cannot modify the protection of the resource section because the memory is owned by the system.

In Win32s version 1.25a and later, the resource section is read/write, regardless of what is specified in the section attributes.

Windows 95 has a handler similar to the one used in Windows NT.

MORE INFORMATION

Under Windows NT, the default top level handler detects writes to resources and will make the resource writable. If you are running outside of a debugger and you have no exception handler, your resource writes will silently work. If you are running under the debugger, your resource write will look like an access violation:

```
First-Chance Exception in msin32.exe: 0xC0000005: Access Violation
```

This allows you to "fix" your resource writes. If you have the debugger pass on the exception to your application and you have no handler, the default handler will make your resource writable.

The disadvantage of setting the attribute of the resource section to read/write is that Windows NT and Windows 95 will use a separate copy of the resource section for each process that uses this section, instead of one copy for all processes.

Additional reference words: 1.20 3.50 4.00 LoadResource LockResource

KBCategory: kbprg
KBSubcategory: UsrRes W32s

Restriction on Named-Pipe Names

PSS ID Number: Q100291
Authored 17-Jun-1993 Last modified 03-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1, 3.5, and 3.51

Named pipes are implemented in Windows NT using the same approach used for file systems. Within Windows NT, named pipes are represented as file objects.

During the design phase, one idea for the implementation was to allow subdirectories of named pipes. For example, a developer could create a named pipe subdirectory called \MYPIPES. It would then be possible to create and use pipes called \MYPIPES\PIPE1 and \MYPIPES\PIPE2, but it would not be possible to use \MYPIPES as a pipe.

In the end, this idea was not implemented, so subdirectories are not supported. This does have some effect on the named-pipe names that are allowed. If a pipe named \MYPIPES is created, it is not possible to subsequently create a pipe named \MYPIPES\PIPE1, because \MYPIPES is already a pipe name and cannot be used as a subdirectory. It is possible to create a pipe named \MYPIPES\PIPE1, but only if there is no pipe named \MYPIPES. The error recieved is ERROR_INVALID_NAME (123).

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseIpc

Results of GetFileInformationByHandle() Under Win32s

PSS ID Number: Q123813
Authored 11-Dec-1994 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32s versions 1.2 and 1.25a

The GetFileInformationByHandle() function returns information about the given file in a structure of type BY_HANDLE_FILE_INFORMATION.

The following are the BY_HANDLE_FILE_INFORMATION structure fields and the values they contain under Win32s:

   dwFileAttributes - Always 0 in version 1.2 (A bug that has been fixed.)
   ftCreationTime - Always 0. (An MS-DOS file system limitation.)
   ftLastAccessTime - Always 0. (An MS-DOS file system limitation.)
   ftLastWriteTime - Correct value.
   dwVolumeSerialNumber - Always 0. (A Win32s limitation.)
   nFileSizeHigh - Correct value.
   nFileSizeLow - Correct value.
   nNumberOfLinks - Always 1.
   nFileIndexHigh - Always 0. (A Win32s limitation.)
   nFileIndexLow - Always 0. (A Win32s limitation.)

Additional reference words: 1.20
KBCategory: kbprg
KBSubcategory: W32s

Retrieving Counter Data From the Registry

PSS ID Number: Q107728
Authored 24-Nov-1993 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, and 3.51

SUMMARY

The performance data begins with a structure of type PERF_DATA_BLOCK and is followed by PERF_DATA_BLOCK.NumObjectTypes data blocks. Each data block begins with a structure of type PERF_OBJECT_TYPE, followed by PERF_OBJECT_TYPE.NumCounters structures of type PERF_COUNTER_DEFINITION. Next, there are PERF_OBJECT_TYPE.NumInstances structures of type PERF_INSTANCE DEFINITION, each directly followed by an instance name, a structure of type PERF_COUNTER_BLOCK and PERF_OBJECT_TYPE.NumCounters counters. All of these data types are described in WINPERF.H.

MORE INFORMATION

The following steps are used to retrieve all of the counter information from the registry:

1. Allocate a buffer to obtain the performance data. Call RegQueryValueEx() to obtain the data. If the call returns ERROR_MORE_DATA, then the buffer size was not big enough. Increase the size of the buffer and try again. Repeat until the call is successful.

```
PPERF_DATA_BLOCK PerfData = NULL;

PerfData = (PPERF_DATA_BLOCK) malloc( BufferSize );

while( RegQueryValueEx( HKEY_PERFORMANCE_DATA,
                        "Global",
                            NULL,
                            NULL,
                        PerfData,
                         &dwSize ) == ERROR_MORE_DATA )
{
   BufferSize += 1024;

   PerfData = (PPERF_DATA_BLOCK) realloc( PerfData, BufferSize );
   if( PerfData == NULL )
      break;
}
```

2. Get the first object:

```
PPERF_OBJECT_TYPE PerfObj;
```

```
PerfObj = (PPERF_OBJECT_TYPE) ((PBYTE)PerfData +
                                PerfData->HeaderLength);
```

3. Get the first instance:

```
PPERF_INSTANCE_DEFINITION PerfInst;

PerfInst = (PPERF_INSTANCE_DEFINITION) ((PBYTE)PerfObj +
                                        PerfObj->DefinitionLength);
```

4. Get the first counter:

```
PPERF_COUNTER_DEFINITION PerfCntr;

PerfCntr = (PPERF_COUNTER_DEFINITION) ((PBYTE)PerfObj +
                                           PerfObj->HeaderLength);
```

5. Get the counter data:

```
PPERF_COUNTER_BLOCK PtrToCntr;
PVOID CntrData;

if( PerfInst != NULL )
   PtrToCntr = (PPERF_COUNTER_BLOCK) ((PBYTE)PerfInst +
                                      PerfInst.ByteLength );
else PtrToCntr = (PPERF_COUNTER_BLOCK) ((PBYTE)PerfObj +
                                      PerfObj.DefinitionLength );
PVOID CntrData = (PVOID)((PBYTE)PtrToCntr + PerfCntr.CounterOffset);
```

6. Get the next counter, repeat steps 5 and 6 until all NumCounters counters are retrieved:

```
PerfCntr = (PPERF_COUNTER_DEFINITION) ((PBYTE)PerfCntr +
                                        PerfCntr->ByteLength);
```

7. After all the counters are retrieved for the instance, get the next instance, repeat steps 4-6 until all NumInstances instances are retrieved:

```
PPERF_COUNTER_BLOCK PtrToCntr;
PPERF_INSTANCE_DEFINITION PerfInst;

PtrToCntr = (PPERF_COUNTER_BLOCK) ((PBYTE) PerfInst +
                                    PerfInst->ByteLength);
PerfInst = (PPERF_INSTANCE_DEFINITION) ((PBYTE) PtrToCntr +
                                    PtrToCntr->ByteLength);
```

8. After all instances of the object type are retrieved, move to the next object type, repeat steps 3-7 until all NumObjectTypes object types are handled:

```
PerfObj = (PPERF_OBJECT_TYPE)((PBYTES)PerfObj +
                                PerfObj.TotalByteLength);
```

Note that the instance names are retrieved in a fashion that is similar to

retrieving the data.

The steps above showed how to obtain all of the counters. You can retrieve only the counters that pertain to a particular object by using the titles database. The information is stored in the registry in the format index, name, index, name, and so forth.

To retrieve the titles database and store it in TitlesDatabase:

1. Open the key:

```
RegOpenKeyEx( HKEY_LOCAL_MACHINE,
   "Software\\Microsoft\\Windows NT\\CurrentVersion\\Perflib\\009",
          0,
   KEY_READ,
      &Hkey);
```

   Note that 009 is a language ID, so this value will be different on a non-English version of the operating system.

2. Query the information from the key:

```
RegQueryInfoKey(
             Hkey,
   (LPTSTR) Class,
       &ClassSize,
             NULL,
          &Subkey,
        MaxSubKey,
        &MaxClass,
          &Values,
         &MaxName,
         &MaxData,
         &SecDesc,
     &LastWriteTime );
```

3. Allocate a buffer to store the information:

```
TitlesDataBase = (PSTR) malloc( (MaxData+1) * sizeof(TCHAR) )
```

4. Retrieve the data:

```
RegQueryValueEx(     Hkey,
      (LPTSTR) "Counters",
                     NULL,
                     NULL,
  (LPBYTE) TitlesDataBase,
                  &MaxData );
```

Once you have the database, it is possible to write code that will go through all objects, searching by index (field ObjectNameTitleIndex) or by type (field ObjectNameTitle - which is initially NULL).

Or, you could obtain only the performance data for specified objects by changing the call to ReqQueryValueEx() in step 1 of the SUMMARY section

above to:

```
   RegQueryValueEx( HKEY_PERFORMANCE_DATA,
                           Indices,
                              NULL,
                              NULL,
                          PerfData,
                           &dwSize );
```

Note that the only difference here is that instead of specifying "Global" as the second parameter, you specify a string that represents the decimal value(s) for the object(s) of interest that are obtained from the titles database.

The PVIEWER and PERFMON samples in the MSTOOLS\SAMPLES\SDKTOOLS directory contain complete sample code that deals with performance data. The "Performance Overview" and Volume 3 of the Windows NT Resource Kit also contain information about performance monitoring.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseMisc

Retrieving DIBs from the Clipboard

PSS ID Number: Q106386
Authored 07-Nov-1993 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

Retrieving a DIB (device-independent bitmap) from the clipboard can take significantly more time than retrieving a bitmap from the clipboard. The difference stems from the fact that a bitmap is a GDI object and a DIB is a global memory object.

MORE INFORMATION

When SetClipBoardData() is passed a global memory handle, as it is when it is passed a handle to a DIB, all the data gets copied into the Win32 server and put into a sharable section of memory. When the DIB is retrieved with GetClipBoardData(), the shared memory is mapped into the application's virtual address space and the memory handle is cached. Any subsequent calls to GetClipBoardData() return quickly, because the memory does not have to be remapped.

In contrast, when retrieving a bitmap with GetClipBoardData(), only a handle is created, because a bitmap is a GDI object.

When CloseClipboard() is called, all of the cached handles to shared memory and GDI objects are deleted.

Rather than reopening the clipboard, it is a good idea to keep a local copy of anything retrieved from the clipboard if the item will be used again after the clipboard has been closed. In general, data should be retrieved from the clipboard only when the application is doing a paste or if the application is a clipboard viewer processing a WM_DRAWCLIPBOARD message.

The data for a GDI object exists on the server side. In other words, bitmaps and DDBs (device-dependent bitmaps) exist in the Win32 subsystem address space. Only the handles of GDI objects are private to an application. Therefore, to make a bitmap or a DDB accessible to another application, only a call to DuplicateHandle() is needed.

Note that even though it is faster to retrieve a DDB from the clipboard, it is still recommended to put a DIB on the clipboard rather than a DDB.

Additional reference words: 3.10 3.50 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrClp

Retrieving Font Styles Using EnumFontFamilies()

PSS ID Number: Q84131
Authored 29-Apr-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

Windows version 3.1 introduces the concept of a font style. In previous versions of Windows, a font could have the bold, italic, underline, and strikeout properties, which were supported by respective members in the LOGFONT and TEXTMETRIC structures. Windows 3.1 also supports these properties, as well as a style name for TrueType fonts. The article describes how to obtain the font style name during font enumeration, using the EnumFontFamilies function. For more information about obtaining style information without enumerating the fonts, query on the following words in the Microsoft Knowledge Base:

prod(winsdk) and getoutlinetextmetrics

MORE INFORMATION

In Windows 3.1, "style" refers to the weight and slant of a font. Windows supports a wide range of weights in the lfWeight member of the LOGFONT structure. (Two examples of weights are FW_BOLD, which is defined as 700, and FW_THIN, which is defined as 100). Very few applications, however, use any weights other than FW_BOLD and FW_DONTCARE (defined as 0).

Windows 3.1 builds on the support presently in Windows for these variations in weight and slant. Style names are NOT used in the LOGFONT structure except when the fonts are enumerated with EnumFontFamilies.

The ChooseFont dialog box in the common dialog boxes dynamic-link library (COMMDLG.DLL) demonstrates how style names are used. The ChooseFont dialog box has two list boxes: Font and Font Style. The Font list box lists the face names for all fonts installed and the Font Style list box lists the font styles for the currently selected face. For example, if any non-TrueType font (such as MS Sans Serif) is selected, the following styles appear in the Font Style list box:

Regular
Bold
Italic
Bold Italic

TrueType fonts may have these or more elaborate styles. For example, the "Lucida Sans" face includes the following style names:

```
Regular
Italic
Demibold Roman
Demibold Italic
```

In the case of Lucida Sans with the style of Demibold Roman or Demibold Italic, the lfWeight value is 600 (FW_DEMIBOLD).

In Windows 3.1, the EnumFontFamilies function can be used to obtain the style name of a font during font enumeration. The EnumFontFamilies function works in a manner very similar to the Windows 3.0 EnumFonts function.

EnumFontFamilies is prototyped as:

```
int EnumFontFamilies(HDC hdc, LPCSTR lpszFamily,
                     FONTENUMPROC lpfnEnumProc, LPARAM lpData)
```

The lpszFamily parameter points to a null-terminated string that specifies the family name (or typeface name) of the desired fonts. If this parameter is NULL, EnumFontFamilies selects and enumerates one font of each available font family. For example, to enumerate all fonts in the "Arial" family, lpszFamily points to a string buffer containing "Arial."

The following table illustrates the meanings of the terms, "typeface name," "font name," and "font style:"

Typeface Name	Font Name	Font Style
Arial	Arial	Regular
	Arial Bold	Bold
	Arial Italic	Italic
	Arial Bold Italic	Bold Italic
Courier New	Courier New	Regular
	Courier New Bold	Bold
	Courier New Italic	Italic
	Courier New Bold Italic	Bold Italic
Lucida Sans	Lucida Sans	Regular
	Lucida Sans Italic	Italic
	Lucida Sans Demibold Roman	Demibold Roman
	Lucida Sans Demibold Italic	Demibold Italic
MS Sans Serif	MS Sans Serif	Regular
	MS Sans Serif	Bold
	MS Sans Serif	Italic
	MS Sans Serif	Bold Italic

The first three typefaces in the above table are TrueType faces, the

remaining typeface is MS Sans Serif. The typeface name is also sometimes referred to as the family name.

When dealing with non-TrueType fonts, typeface name and font name are the same. However, it is important to recognize the distinction when dealing with a TrueType font.

For example, CreateFont takes a pointer to a string containing the typeface name of the font to create. It is not valid to use Arial Bold as this string because Arial is a TrueType font and Arial Bold is a font name, not a typeface name.

If EnumFontFamilies is called with the lpszFamily parameter pointing to a valid TrueType typeface name, the callback function, which is specified in fntenmprc, will be called once for each font name for that typeface name. For example, if EnumFontFamilies is called with lpszFamily pointing to Lucida Sans, the callback function will be called four times; once for each font name.

If the lpszFamily parameter points to the typeface name of a non-TrueType font, such as MS Sans Serif, the callback will be called once for each face size supported by the font. The number of face sizes supported by the font can vary from font to font and from device to device. Note that the callback is called for different sizes, not for different styles. This behavior is identical to that found using the EnumFonts function.

Remember that, because TrueType fonts are continuously scalable, there is no reason for the callback function to be called for each size. If the callback function was called for each size that a TrueType font supported, the callback function would be called an infinite number of times!

The EnumFontFamilies callback function is prototyped as follows:

```
int CALLBACK EnumFontFamProc(LPNEWLOGFONT lpnlf,
                             LPNEWTEXTMETRIC lpntm,
                             int FontType, LPARAM lpData)
```

The lpnlf parameter points to a LOGFONT structure that contains information about the logical attributes of the font. If the typeface being enumerated is a TrueType font [(nFontType | TRUETYPE_FONTTYPE) is TRUE], this LOGFONT structure will have two additional members appended to the end of the structure, as follows:

```
   char   lfFullName[LF_FACESIZE*2];
   char   lfStyleName[LF_FACESIZE];
```

It is important to remember that these two additional fields are used *only* during enumeration with EnumFontFamilies and nowhere else in Windows. The documentation for the EnumFontFamilies function on pages 266-268 of the "Microsoft Windows Software Development Kit: Programmer's Reference, Volume 2: Functions" manual refers to the NEWLOGFONT structure which contains the additional members listed above. However, the NEWLOGFONT structure is not defined in the

WINDOWS.H header file. To address this situation, use the ENUMLOGFONT structure which is defined in the WINDOWS.H file but is not listed in the Windows SDK documentation.

To retrieve the style name and full name of a font without using enumeration, use the GetOutlineTextMetrics function.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiTt

Retrieving Handles to Menus and Submenus

PSS ID Number: Q67688
Authored 11-Dec-1990 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

To change the contents of a menu, you must have its handle. The handle of the menu associated with a given window is available through the GetMenu() function.

To obtain a reference to a particular text item in the menu, use the GetMenuString() function. The definition for this function is

```
GetMenuString(hMenu, wIDItem, lpString, nMaxCount, wFlag)
```

where

```
hMenu     = The menu handle
wIDItem   = The ID of the item or the zero-based offset of the
            item within the menu
lpString  = The buffer that is to receive the text
nMaxCount = The length of the buffer
wFlag     = MF_BYCOMMAND or MF_BYPOSITION
```

If a menu item has a mnemonic, the text will contain an ampersand (&) character preceding the mnemonic character.

To obtain the handle to a submenu of the menu bar, use the GetSubMenu() function. The second parameter, nPos, is the zero-based offset from the beginning of the menu.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMen

Retrieving Palette Information from a Bitmap Resource

PSS ID Number: Q124947
Authored 17-Jan-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

You may sometime need to create a logical palette from a bitmap resource in order to display the bitmap with the maximum number of available colors. For example, on an 8 bit-per-pixel display, a logical palette is necessary to draw a 256-color bitmap on a device context for that display. The LoadBitmap function does not return or take a palette as one of its parameters; thus, for example, there is no way to incorporate a palette with a 256-color bitmap loaded with LoadBitmap. Therefore, an application must load the resource as a device-independent bitmap (DIB), rather than a device-dependent bitmap (DDB), in order to retrieve the bitmap's color information. An application can use the FindResource, LoadResource, and LockResource functions to do this.

MORE INFORMATION

A bitmap (.BMP file) is stored in an application's resources as a (DIB), along with a color table if one exists. When a DIB is loaded from an application's resources with the LoadBitmap function, a DDB is returned. This DDB is a bitmap compatible with the screen. Routines such as CreateDIBitmap and SetDIBits that convert DIBs to DDBs take a handle to a device context as their first parameter. This tells the routine what kind of DDB to create. If this device context currently has a palette selected into it, then CreateDIBitmap or SetDIBits can use this palette to create the DDB. Without a palette, the routines are restricted to system colors when matching the DIB's colors to the DDB's colors. For example, on an 8 bit-per-pixel display, the resulting DDB can have only up to 20 different colors. With a logical palette, the resulting bitmap could have had up to 256 different colors.

If the bitmap is loaded as a DIB from the resource, then an application can query the DIB's color table and create a logical palette for the DIB. Then, it can call either CreateDIBitmap or SetDIBits, along with a device context with that palette selected, to obtain a DDB compatible with that palette. To load a bitmap from a resource as a DIB, you can use the FindResource function with the RT_BITMAP flag set and then use the LoadResource function to load it. You can lock the resource with the LockResource function.

The following code demonstrates how to use the above technique to load a DIB from an application's resources, create a palette for it, and then create a DDB out of it. The LoadResourceBitmap function below can be used

in place of the LoadBitmap function. The only additional parameter needed is the address of a logical palette handle. The palette handle referenced will contain a handle to a logical palette after the function is called.

```
HBITMAP LoadResourceBitmap(HINSTANCE hInstance, LPSTR lpString,
HPALETTE FAR* lphPalette)
{
    HRSRC  hRsrc;
    HGLOBAL hGlobal;
    HBITMAP hBitmapFinal = NULL;
    LPBITMAPINFOHEADER  lpbi;
    HDC hdc;
    int iNumColors;

    if (hRsrc = FindResource(hInstance, lpString, RT_BITMAP))
       {
       hGlobal = LoadResource(hInstance, hRsrc);
       lpbi = (LPBITMAPINFOHEADER)LockResource(hGlobal);

       hdc = GetDC(NULL);
       *lphPalette =  CreateDIBPalette ((LPBITMAPINFO)lpbi, &iNumColors);
       if (*lphPalette)
          {
          SelectPalette(hdc,*lphPalette,FALSE);
          RealizePalette(hdc);
          }

       hBitmapFinal = CreateDIBitmap(hdc,
                   (LPBITMAPINFOHEADER)lpbi,
                   (LONG)CBM_INIT,
                   (LPSTR)lpbi + lpbi->biSize + iNumColors *
sizeof(RGBQUAD),

                   (LPBITMAPINFO)lpbi,
                   DIB_RGB_COLORS );

       ReleaseDC(NULL,hdc);
       UnlockResource(hGlobal);
       FreeResource(hGlobal);
       }
    return (hBitmapFinal);
}

HPALETTE CreateDIBPalette (LPBITMAPINFO lpbmi, LPINT lpiNumColors)
{
   LPBITMAPINFOHEADER  lpbi;
   LPLOGPALETTE     lpPal;
   HANDLE           hLogPal;
   HPALETTE         hPal = NULL;
   int              i;

   lpbi = (LPBITMAPINFOHEADER)lpbmi;
   if (lpbi->biBitCount <= 8)
       *lpiNumColors = (1 << lpbi->biBitCount);
   else
```

```
        *lpiNumColors = 0;  // No palette needed for 24 BPP DIB

   if (*lpiNumColors)
      {
      hLogPal = GlobalAlloc (GHND, sizeof (LOGPALETTE) +
                             sizeof (PALETTEENTRY) * (*lpiNumColors));
      lpPal = (LPLOGPALETTE) GlobalLock (hLogPal);
      lpPal->palVersion    = 0x300;
      lpPal->palNumEntries = *lpiNumColors;

      for (i = 0;  i < *lpiNumColors;  i++)
         {
         lpPal->palPalEntry[i].peRed   = lpbmi->bmiColors[i].rgbRed;
         lpPal->palPalEntry[i].peGreen = lpbmi->bmiColors[i].rgbGreen;
         lpPal->palPalEntry[i].peBlue  = lpbmi->bmiColors[i].rgbBlue;
         lpPal->palPalEntry[i].peFlags = 0;
         }
      hPal = CreatePalette (lpPal);
      GlobalUnlock (hLogPal);
      GlobalFree   (hLogPal);
   }
   return hPal;
}
```

Here is an example of how you might use the above function to load a bitmap from a resource and display it using a logical palette:

```
   {
   HBITMAP hBitmap,hOldBitmap;
   HPALETTE hPalette;
   HDC hMemDC, hdc;
   BITMAP bm;

   hBitmap = LoadResourceBitmap(hInst,"test", &hPalette);
   GetObject(hBitmap, sizeof(BITMAP), (LPSTR)&bm);
   hdc = GetDC(hWnd);
   hMemDC = CreateCompatibleDC(hdc);
   SelectPalette(hdc,hPalette,FALSE);
   RealizePalette(hdc);
   SelectPalette(hMemDC,hPalette,FALSE);
   RealizePalette(hMemDC);
   hOldBitmap = SelectObject(hMemDC,hBitmap);
   BitBlt(hdc,0,0,bm.bmWidth,bm.bmHeight,hMemDC,0,0,SRCCOPY);
   DeleteObject(SelectObject(hMemDC,hOldBitmap));
   DeleteDC(hMemDC);
   ReleaseDC(hWnd,hdc);
   DeleteObject(hPalette);
   }
```

REFERENCES

For more information on DIB-related functions, please review the Microsoft Windows SDK sample DIBVIEW.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiBmp GdiBmpFormat

Retrieving the Style String for a TrueType Font

PSS ID Number: Q84132
Authored 29-Apr-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

Windows version 3.1 introduces the concept of a font style. In previous versions of Windows, a font could have bold, italic, underline, and strikeout properties, which were supported by corresponding members of the LOGFONT and TEXTMETRIC structures. Windows 3.1 continues to support these properties, however, it also supports the concept of a style name for TrueType fonts.

Windows use of style names can be demonstrated by the ChooseFont dialog box in the common dialog boxes dynamic-link library (COMMDLG.DLL). The ChooseFont dialog box contains two list boxes named Font and Font Style. The Font list box contains a list of all face names and the Font Style list box contains a list of font styles for the currently selected face. For example, if any non-TrueType font (such as MS Sans Serif) is selected, the following styles will appear in the style list box:

```
Regular
Bold
Italic
Bold Italic
```

TrueType fonts may have these or more elaborate styles. For example, the "Lucida Sans" face includes the following style names:

```
Regular
Italic
Demibold Roman
Demibold Italic
```

MORE INFORMATION

As part of the TrueType support, the GetOutlineTextMetrics function can be used to retrieve metric information for TrueType fonts, including the style name.

GetOutlineTextMetrics is prototyped as follows:

```
DWORD GetOutlineTextMetrics(HDC hdc, UINT cbData,
                            LPOUTLINETEXTMETRIC lpotm);
```

The hdc parameter identifies the device context. GetOutlineTextMetrics retrieves the metric information for the font currently selected into the specified device context. For GetOutlineTextMetrics to succeed, the font must be a TrueType font. The sample code given below shows how to synthesize the style name for a non-TrueType font.

The cbData parameter specifies the size, in bytes, of the buffer in which information is returned.

The lpotm parameter points to an OUTLINETEXTMETRIC structure. If this parameter is NULL, the function returns the size of the buffer required for the retrieved metric information.

The OUTLINETEXTMETRIC structure contains most of the font metric information provided with the TrueType format. The relative parts of the structure are listed below:

```
typedef struct tagOUTLINETEXTMETRIC {
  .
  .
  .
    PSTR    otmpFamilyName;
    PSTR    otmpFaceName;
    PSTR    otmpStyleName;
    PSTR    otmpFullName;
} OUTLINETEXTMETRIC;
```

While these four members of the OUTLINETEXTMETRIC structure are defined as near pointers to strings (PSTR), they are actually offsets into the structure from the beginning of the structure. Because the length of these strings is not defined, an application must allocate space for them above and beyond the space allocated for the OUTLINETEXTMETRIC structure itself. The sample code below demonstrates this. It also demonstrates using GetOutlineTextMetrics in an application that will also work with Windows 3.0.

```
#include <windows.h>
#include <windowsx.h>
.
.
.
HFONT                hFont;
LPOUTLINETEXTMETRIC  potm;
TEXTMETRIC           tm;
int                  cbBuffer;

hFont = CreateFont( ..... );

hFont = SelectObject(hDC, hFont);

/*
 * Call the GetTextMetrics function to determine whether or not the
 * font is a TrueType font.
 */
GetTextMetrics(hDC, &tm);
```

```
/*
* GetOutlineTextMetrics is a function implemented in Windows 3.1
* and later. Assume fWin30 was determined by calling GetVersion.
*/
if (!fWin30 && tm.tmPitchAndFamily & TMPF_TRUETYPE)
{
   WORD (WINAPI *lpfnGOTM)(HDC, UINT, LPOUTLINETEXTMETRIC);

   /*
    * GetOutlineTextMetrics is exported from
    * GDI.EXE at ordinal #308
    */
   lpfnGOTM = GetProcAddress(GetModuleHandle("GDI"),
                             MAKEINTRESOURCE(308));

   /*
    * Call GOTM with NULL to retrieve the size of the buffer.
    */
   cbBuffer = (*lpfnGOTM)(hDC, NULL, NULL);

   if (cbBuffer == 0)
   {
     /* GetOutlineTextMetrics failed! */
     hFont = SelectObject(hDC, hFont);
     DeleteObject(hFont);
     return FALSE;
   }

   /*
    * Allocate the memory for the OUTLINETEXTMETRIC structure plus
    * the strings.
    */
   potm = (LPOUTLINETEXTMETRIC)GlobalAllocPtr(GHND, cbBuffer);

   if (potm)
   {
     potm->otmSize = cbBuffer;

     /*
      * Call GOTM with the pointer to the buffer. It will
      * fill in the buffer.
      */
     if (!(*lpfnGOTM)(hDC, cbBuffer, potm))
     {
        /* GetOutlineTextMetrics failed! */
        hFont = SelectObject(hDC, hFont);
        DeleteObject(hFont);
        return FALSE;
     }

     /*
      * Do something useful with the string buffers. NOTE: To access
      * the string buffers, the otmp???Name members are used as
      * OFFSETS into the buffer. They *ARE NOT* pointers themselves.
```

```
       */
      OutputDebugString((LPSTR)potm + (UINT)potm->otmpFamilyName);
      OutputDebugString((LPSTR)potm + (UINT)potm->otmpFaceName);
      OutputDebugString((LPSTR)potm + (UINT)potm->otmpStyleName);
      OutputDebugString((LPSTR)potm + (UINT)potm->otmpFullName);

      /* Don't forget to free the memory! */
      GlobalFreePtr(potm);
    }
    else
    {
      /* GlobalAllocPtr failed */
      hFont = SelectObject(hDC, hFont);
      DeleteObject(hFont);
      return FALSE;
    }
}
else
{
    /*
     * It was not a TrueType font, or Windows 3.0 is running.
     */
    LOGFONT lf;
    char    szStyle[LF_FACESIZE];
    LPSTR   p;

    GetObject(hFont, sizeof(LOGFONT), &lf);

    /*
     * Fabricate a style string. Important note! The strings
     * "Italic", "Bold", and "Regular" are only valid in English. On
     * versions of Windows localized for other countries, these
     * strings will differ.
     */
    szStyle[0] = '\0';

    if (lf.lfWeight >= FW_BOLD)
      lstrcpy(szStyle, "Bold ");

    /*
     * If it's "Bold Italic," concatenate.
     */
     p = szStyle + lstrlen(szStyle);

     if (lf.lfItalic)
       lstrcpy(p, "Italic");

     if (!lstrlen(szStyle))
       lstrcpy(szStyle, "Regular");

    /*
     * szStyle now holds what is equivalent to the otmpStyleName
     * member.
     */
    OutputDebugString(szStyle);
```

```
  }

  hFont = SelectObject(hDC, hFont);
  DeleteObject(hFont);
```

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiTt

Retrieving the Text Color from the Font Common Dialog Box

PSS ID Number: Q86331
Authored 02-Jul-1992 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

The common dialog box library (COMMDLG.DLL) provides the ChooseFont() routine that provides a common interface (Font dialog box) to specify the attributes for a font in an application. When the user chooses the Apply button in the Font dialog box, the application can format selected text using the specified font. This button is also useful in an application that allows the user to select more than one font simultaneously.

The Font dialog box provides a common method to specify a number of font attributes, including color. An application can send the CB_GETITEMDATA message to the Color combo box to retrieve the currently selected color.

This article discusses the procedure required to obtain all the information about the currently specified font.

MORE INFORMATION

The Font dialog box includes an Apply button if the application includes the CF_APPLY value in the specification for the Flags member of the CHOOSEFONT data structure. The dialog box includes the Color combo box if the CF_EFFECTS value is also specified. The remainder of this article assumes that the application has specified both of these values.

To properly process input from the Apply button, an application must install a hook function. For more information on installing a hook function from an application, query on the following words in the Microsoft Knowledge Base:

```
steps adding hook function
```

The following code illustrates one method to process input from the Apply button:

```
case WM_COMMAND:
   switch (wParam)
   {
      case psh3:  // The Apply button

      // Retrieve the font information...
         SendMessage(hDlg, WM_CHOOSEFONT_GETLOGFONT, 0,
                     (LONG)(LPLOGFONT)&lfLogFont);
```

```
         // Perform any required processing
         // (create the specified font, for example)

         // Retrieve color information...
            iIndex = (int)SendDlgItemMessage(hDlg, cmb4,
                                             CB_GETCURSEL, 0, 0L);
            if (iIndex != CB_ERR)
            {
               dwRGB = SendDlgItemMessage(hDlg, cmb4, CB_GETITEMDATA,
                                          (WORD)iIndex, 0L);
               wRed = GetRValue(dwRGB);
               wGreen = GetGValue(dwRGB);
               wBlue = GetBValue(dwRGB);

               wsprintf(szBuffer, "RGB Value is %u %u %u\r\n", wRed,
                        wGreen, wBlue);

               OutputDebugString(szBuffer);
            }
            break;

         default:
            break;
      }
      break;
```

The color information is not required to create the font; however, this information is required to accurately display the font according to the user's specification.

In an application that does not use the Apply button, the rgbColors member of the CHOOSEFONT data structure contains the selected color. In this case, no special processing to retrieve the color is required.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCmnDlg

Retrieving Time-Zone Information

PSS ID Number: Q115231
Authored 22-May-1994 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.5, 3.51, and 4.0

In Windows NT, version 3.1, the time-zone strings are compiled into a resource that is linked into the CONTROL.EXE file. For Windows NT, version 3.5 and later and Windows 95, the time-zone strings have been moved into the registry.

In Windows NT, the time-zone strings are located in the key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\
   Microsoft\
   Windows NT\
   CurrentVersion\
   Time Zones.
```

In Windows 95, the time-zone strings are located in the key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\
   Microsoft\
   Windows\
   CurrentVersion\
   Time Zones.
```

For each time zone, the registry key TZI is formatted as follows:

```
LONG       Bias;
LONG       StandardBias;
LONG       DaylightBias;
SYSTEMTIME StandardDate;
SYSTEMTIME DaylightDate;
```

You can use this information to fill out a TIME_ZONE_INFORMATION structure, which is used when calling SetTimeZoneInformation().

Additional reference words: 3.50 4.00
KBCategory: kbprg
KBSubcategory: BseMisc

Returning CBR_BLOCK from DDEML Transactions

PSS ID Number: Q102584
Authored 04-Aug-1993 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

DDEML servers are applications that provide data to client applications. For some servers, this data gathering may be a lengthy process, as when gathering data from sources such as serial ports or a network. DDEML allows a server application to process data asynchronously in these situations by returning CBR_BLOCK from the DDE callback function.

MORE INFORMATION

In DDEML-based applications, while transactions can be either synchronous or asynchronous, only DDEML client applications may choose to establish either type of transaction when requesting data from a server application. DDEML server applications do not distinguish between synchronous and asynchronous transactions.

Asynchronous transactions can be very useful when client applications know that the partner server application will take some time to gather data. This type of transaction frees up the client to do other things while waiting for a notification from the server of data availability.

Server applications have no way of determining whether the client application has requested data synchronously or asynchronously. Request transactions on the server's side are always synchronous. When a client requests data, the server's callback receives an XTYP_REQUEST transaction, where the expected return value is a data handle. If the server application has to wait for data from a serial port, for example, access to the CPU by other applications will be delayed, thereby freezing the system until data arrives.

There are a couple of ways one can enable the server to gather data in an asynchronous manner, thereby allowing it to yield to other applications on the system while it gathers data. One method is to use CBR_BLOCK; another is to change the request transaction to a one-time ADVISE loop.

Method 1

Given that DDEML callbacks are not re-entrant, and that DDEML expects a data handle as a return value from the XTYP_REQUEST transaction (and

transactions of XCLASS_DATA class), the server application can block the callback momentarily. It can do this by returning a CBR_BLOCK value after posting itself a user-defined message.

This way, the server application can gather data in the background while DDEML queues up any further transactions. The server can start gathering data when its window procedure gets the user defined message that was posted by its DDE callback function.

When a server application returns CBR_BLOCK for a request transaction, DDEML disables the server's callback function. It also queues transactions that are sent by DDEML after its callback has been disabled. This feature gives the server an opportunity to gather data while allowing other applications to run in the system.

As soon as data becomes available, then the server application can call DdeEnableCallBack() to re-enable the server callback function. Once the callback is re-enabled, DDEML will resend the same request transaction to the server's callback and this time, because data is ready, the server application can return the appropriate data handle to the client.

Transactions that were queued up because of an earlier block are sent to the server's callback function in the order they were received by DDEML.

The pseudo code to implement method 1 might resemble the following:

```
BOOL gbGatheringData = TRUE;   // Defined GLOBAlly.
HDDEDATA ghData = NULL;

HDDEDATA CALLBACK DdeServerCallBack(...)
   {
       switch(txnType)
       {
       case XTYP_REQUEST:

          // If the server takes a long time to gather data...
          // for this topic/item pair, then
          // post a user-defined message to the server app's wndproc
       // and return CBR_BLOCK... DDEML will block the callback
       // and queue transactions.

       if(bGatheringData) {
         PostMessage(hSrvWnd, WM_GATHERDATA, .....) ;
         return CBR_BLOCK;
       }
       else              // Data is ready, send back handle.
         return ghData;

       default:
         return DDE_FNOTPROCESSED;
      }
    }
```

```
LRESULT CALLBACK SrvWndProc(...)
  {
    switch (wMessage)
       {
          case WM_GATHERDATA:

             while (bGatheringData)
               {
                  // Gather data here while yielding to others
                  // at the same time!
      if(!PeekMessage(..))
        bGatheringData = GoGetDataFromSource (&ghData);
                  else {
                    TranslateMessage() ;
                    DispatchMessage ();
                  }
                }
           DdeEnableCallback (idInst, ghConv, EC_ENABLEALL);
              break ;

           default:
             return DefWndProc();
         }
     }
```

Method 2

Advise transactions in DDEML (or DDE) are just a continuous request link. Changing the transaction from a REQUEST to a "one time only" ADVISE loop on the client side allows the server to gather data asynchronously.

The client application can start an ADVISE transaction from its side and when the server receives a XTYP_ADVSTART transaction, return TRUE so that an ADVISE link is established. Once the link is established, the server can start gathering data, and as soon as it becomes available, notify the client of its availability.

This can be done by calling DdePostAdvise(). The server can use PeekMessage() to gather data if the data gathering process is a lengthy one, so that other applications on the system will get a chance to run. Once the client receives the data from the server in its callback (in its XTYP_ADVDATA transaction), it can disconnect the the ADVISE link from the server by specifying an XTYP_ADVSTOP transaction.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDde

Rich Edit Control Does Not Support Unicode

PSS ID Number: Q128558
Authored 03-Apr-1995 Last modified 04-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development KIT (SDK) for Windows NT version 3.51 and Windows 95

The Rich Edit control included with Windows 95 and Windows NT version 3.51 does not support Unicode.

For a Unicode application to use the Rich Edit control, it must convert any strings passed to the control to ASCII text. This includes strings used in the FINDTEXT structure and the TEXTRANGE structure.

Additional reference words: 3.50 4.00
KBCategory: kbprg
KBSubcategory: UsrCtl

Rotating a Bitmap by 90 Degrees

PSS ID Number: Q77127
Authored 08-Oct-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.0 and 3.1
- Microsoft Win32 SDK, version 4.0

SUMMARY

There are no Windows functions that directly rotate bitmaps. All techniques for rotating bitmaps in Windows involve copying the rows from a source bitmap into the columns of a destination bitmap. The following contains code for rotating a bitmap using GetPixel() and SetPixel(), and contains an outline of code for rotating device independent bitmaps (DIB).

MORE INFORMATION

A device dependent bitmap (DDB) can be rotated using the GetPixel() and SetPixel() functions. To rotate the bitmap, use the following code:

```
HBITMAP Rotate90(HDC hDC, HBITMAP hSourceBitmap)
{
  HBITMAP hOldSourceBitmap, hOldDestBitmap, hDestBitmap;
  HDC hMemSrc, hMemDest;
  int height, width;
  int i, j;
  BITMAP iSrcBitmap;

  // Step 1: Create a memory DC for the source and destination bitmaps
  //         compatible with the device used.

  hMemSrc = CreateCompatibleDC(hDC);
  hMemDest= CreateCompatibleDC(hDC);


  // Step 2: Get the height and width of the source bitmap.

  GetObject(hSourceBitmap, sizeof(BITMAP), (LPSTR)&SrcBitmap);
  width = SrcBitmap.bmWidth;
  height = SrcBitmap.bmHeight;


  // Step 3: Select the source bitmap into the source DC. Create a
  //         destination bitmap, and select it into the destination DC.

  hOldSourceBitmap = SelectObject(hMemSrc, hSourceBitmap);
  hDestBitmap = CreateBitmap(height, width, SrcBitmap.bmPlanes,
                   SrcBitmap.bmBitsPixel, NULL);
```

```
  if (!hDestBitmap)
    return(hDestBitmap);

  hOldDestBitmap = SelectObject(hMemDest, hDestBitmap);


  // Step 4: Copy the pixels from the source to the destination.

  for (i = 0; i < width; ++i)
      for (j = 0; j < height; ++j)
          SetPixel(hMemDest, j, width - 1 - i,
                                          GetPixel(hMemSrc, i, j));


  // Step 5: Destroy the DCs.

  SelectObject(hMemSrc, hOldSourceBitmap);
  SelectObject(hMemDest, hOldDestBitmap);
  DeleteDC(hMemDest);
  DeleteDC(hMemSrc);


  // Step 6: Return the rotated bitmap.

  return(hDestBitmap);
}
```

If the bitmap is larger, using GetPixel() and SetPixel() may be too slow. If this is the case, there are two options:

1. If the contents of the bitmap do not change, create two versions of the bitmap, the normal version and one that is rotated by 90 degrees. Load the appropriate bitmap as required.

-or-

2. Find some way to manipulate the bits of the bitmap that is faster than using SetPixel() and GetPixel(). The best way to do this is to convert the bitmap to a device independent bitmap. The following four steps detail how to create the DIB and to perform the rotation:

   a. Use GetDIBits() to convert the bitmap to a device independent format. It is necessary to create a BITMAPINFO structure appropriate for the bitmap. This will write the bitmap as a series of scanlines. Each scanline is padded so that it is DWORD aligned.

   b. Allocate memory for the destination bitmap. This bitmap requires as many scanlines as the width of the source bitmap. Each scanline is as many pixels wide as the height of the source bitmap. Also, the scanlines must be DWORD aligned.

   c. For each scanline in the source bitmap, copy the pixels to the

appropriate column in the destination bitmap. NOTE: The format for each scanline depends upon the number of bits per pixel. See the BITMAPINFO documentation for a description of the possible formats.

d. Use SetDIBits() to copy the device independent bits into a device dependent bitmap. Another BITMAPINFO structure, appropriate for the destination device is required for this step.

The four steps of this method require much more work than is required if GetPixel() and SetPixel() are used; however, this method may be faster because it directly manipulates the bits in the bitmap.

Additional reference words: 3.00 3.10 4.00 95
KBCategory: kbprg
KBSubcategory: GdiBmp

RPC CALLBACK Attribute and Unsupported Protocol Sequences

PSS ID Number: Q131495
Authored 12-Jun-1995 Last modified 15-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 3.5

SUMMARY

The CALLBACK attribute of RPC does not support connection-less protocol sequences.

MORE INFORMATION

If an RPC interface has a procedure with the CALLBACK attribute, it can only make use of connection-oriented protocol sequences. The following protocol sequences are not supported:

- ncadg_ip_udp
- ncadg_ipx

If an RPC client tries to call a remote procedure that in turn calls a procedure back on client, the client will be able make the initial call, but when the server tries to call the procedure on client, the RPC run time generates exception 1726: The remote procedure call failed.

Additional reference words: 3.50
KBCategory: kbprg kbnetwork
KBSubcategory: NtwkRpc

RPC Can Use Multiple Protocols

PSS ID Number: Q100009
Authored 14-Jun-1993 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, and 4.0

Microsoft Remote Procedure Call (RPC) does not rely on a single protocol. An RPC server can be written to use all available protocols on the server, and a client can be written in the same manner. Thus, the server or client does not have to know which protocols it supports explicitly.

RPC protocols supported by Windows 3.1 and 3.5 are:

ncacn_ip_tcp	(TCP/IP)
ncacn_nb_nb	(NetBIOS over NetBEUI)
ncacn_nb_tcp	(NetBIOS over TCP)
ncacn_np	(Named Pipes)
ncalrpc	(LPC)

RPC protocols supported by Windows 3.5 only are:

ncadg_ipx	(Datagram - IPX)
ncacn_spx	(SPX)
ncadg_ip_udp	(Datagram - UDP)
ncacn_nb_ipx	(Netbios over IPX)

RPC protocols supports by Windows 95 are:

ncacan_ip_tcp
ncacn_nb_nb
ncacn_np
ncacn_spx
ncalrpc

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: NtwkRpc

RpcNsxxx() APIs Not Supported by Windows NT Locator

PSS ID Number: Q104318
Authored 13-Sep-1993 Last modified 23-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

The RpsNSxxx group and profile name service application programming interfaces (APIs), such as RpcNsProfileEltAdd(), are supported by our RPC run time; however, they are not supported by the Windows NT Locator, which is the default RPC name service provider. If you attempt to make a call to one of these APIs, an error 1764, "request not supported," will be returned. Because the RpcNSxxx APIs are supported in the run time, name service providers other than the Locator, such as the DCE CDS, can be accessed.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: NtwkRpc

Running a Windows-Based Application in its Own VDM

PSS ID Number: Q115235
Authored 22-May-1994 Last modified 04-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, version 3.5

SUMMARY

The Windows NT, version 3.1, Windows on Win32 (WOW) supports running all 16-bit Windows-based applications in one virtual machine (VM). These applications share an address space, just as they do on Windows. The Windows NT, version 3.5, WOW supports running a Windows-based application in its own VM, which gives the application its own address space.

In order to programmatically start a Windows-based application in its own VM, start the application with CreateProcess() and then specify the flag CREATE_SEPARATE_WOW_VDM.

To specify a Win16 application started from the command prompt to run in its own address space, use the following syntax:

```
start /SEPARATE <filename>
```

To specify a Windows-based application started from the Program Manager to run in its own address space, check the following item in the Program Item Properties for the application or in the Run dialog box, which can be selected from the Program Manager's File menu:

```
Run in Separate Memory Space
```

NOTE: This option is not the default, nor is there any way to make it the default.

MORE INFORMATION

Allowing a Windows-based application to run in a separate address space provides for more robust operation, because the application is isolated from other Windows-based applications. However, the downside is twofold:

- WOW VMs require approximately 2500K of private memory on x86 machines.

- There is no shared memory between WOW VMs. Therefore, Windows-based applications that rely on shared memory cannot be run in separate VMs. As an alternative, use DDE or OLE, because they can be used by an application in one VM to communicate with an application in another VM.

Additional reference words: 3.50

KBCategory: kbprg
KBSubcategory: Subsys

Running Bound Applications Under Windows NT

PSS ID Number: Q90913
Authored 26-Oct-1992 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

Bound applications are designed and built so they can be run under either OS/2 or MS-DOS. The OS/2 subsystem is not available on MIPS; therefore, bound applications will not run as OS/2 applications on MIPS.

When a bound application is run under Windows NT on an 80x86 system, the application will automatically run under the OS/2 subsystem if it is available.

MORE INFORMATION

The OS/2 subsystem is available by default on an 80x86 system. To force bound applications to run as an MS-DOS-based application, you can disable the OS/2 subsystem, but this is not recommended. Instead use the FORCEDOS facility. Type "FORCEDOS /?" at the command line to get help on FORCEDOS. It is not advised that you disable the OS/2 subsystem unless there is a very specific need that FORCEDOS does not address.

To disable the OS/2 subsystem using RegEdit:

   Go to HKEY_LOCAL_MACHINE
   Go to SYSTEM
   Go to Current Control Set
   Go to Control
   Go to Session Manager
   Go to SubSystems

   Modify 'Optional: REG_MULTI_SZ OS/2 Posix'. Specifically, remove OS/2 and reboot.

Once this is done, bound applications will run as MS-DOS-based applications. Running an OS/2 application results in the following message:

   Cannot connect to OS/2 subsystem

WARNING: REGEDT32 is a very powerful utility that facilitates directly changing the Registry database. Using REGEDT32 incorrectly can cause serious problems, including hard disk corruption. It may be necessary to reinstall the software to correct some problems. Microsoft does not support changes made with REGEDT32. Use this tool AT YOUR OWN RISK.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: SubSys

RW2002 Error "Cannot Reuse String Constants" in RC.EXE

PSS ID Number: Q21569
Authored 31-Oct-1988 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The error "Cannot Reuse String Constants" will be returned by the Resource Compiler if you have used the same ID value to define two different string constants.

For example, the following error is returned when compiling the resource file:

```
Cannot Reuse String Constants
```

The file MY.RC may contain the following lines:

MORE INFORMATION

The following sample code can be used to demonstrate the problem.

Sample Code

```
StringTable
begin
   1, "one"
   2, "two"
   3, "three"
   1, "four"
end
```

Note that "one" and "four" have the same value. This error may be less noticeable if you are using both decimal and hexadecimal notation in your RC file. In the following example, 0x010 and 16 have the same value and generate the error:

```
 0x010, "hex 10"
 10, "ten"
 11, "eleven"
 15, "fifteen"
 16, "sixteen"
```

Additional reference words: 3.00 3.10 3.50 4.00 95 RW2002 RC2151
KBCategory: kbtool

KBSubcategory: TlsRc

SAMPLE: 16 and 32 Bits-Per-Pel Bitmap Formats

PSS ID Number: Q94326
Authored 04-Jan-1993 Last modified 21-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

Windows NT supports the same bitmap formats as Microsoft Windows version 3.1, but includes two new formats: 16 and 32 bits-per-pel.

SEEDIB.EXE is a file in the Microsoft Software Library that contains the source code to an application that demonstrates how to load, display, and save 1, 4, 8, 16, 24, and 32-bits-per-pixel DIB formats. In addition, it demonstrates a simple method of creating an optimized palette for displaying DIBSs with more than 8-bits-per-pixel on 8-bits-per-pixel devices.

NOTE: In order to minimize color loss, SeeDIB uses CreateDIBSection() to do conversions between uncompressed DIBs which have more than 8-bits-per-pixel. This function is not available on Windows NT 3.1.

You can download SEEDIB.EXE from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for SEEDIB.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download SEEDIB.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get SEEDIB.EXE

MORE INFORMATION

For DIBs (device independent bitmaps), the 16 and 32-bit formats contain three DWORD masks in the bmiColors member of the BITMAPINFO structure. These masks specify which bits in the pel correspond to which color.

The three masks must have contiguous bits, and their order is assumed to be R, G, B (high bits to low bits). The order of the three masks in the color table must also be first red, then green, then blue (RGB). In this manner,

the programmer can specify a mask indicating how many shades of each RGB color will be available for bitmaps created with CreateDIBitmap(). For 16-bits-per-pixel DIBs, CreateDIBitmap() defaults to the RGB555 format. For 32-bits-per-pixel DIBs, CreateDIBitmap() defaults to an RGB888 format.

NOTE: The DIB engine in Windows 95 supports only RGB555 and RGB565 for 16-bit DIBs and only RGB888 for 32-bit DIBs.

Example

The RGB555 format masks would look like:

```
   0x00007C00  red   (0000 0000 0000 0000 0111 1100 0000 0000)
   0x000003E0  green (0000 0000 0000 0000 0000 0011 1110 0000)
   0x0000001F  blue  (0000 0000 0000 0000 0000 0000 0001 1111)
```

NOTE: For 16 bits-per-pel, the upper half of the DWORDs are always zeroed.

The RGB888 format masks would look like:

```
   0x00FF0000  red   (0000 0000 1111 1111 0000 0000 0000 0000)
   0x0000FF00  green (0000 0000 0000 0000 1111 1111 0000 0000)
   0x000000FF  blue  (0000 0000 0000 0000 0000 0000 1111 1111)
```

Usage

When using 16 and 32-bit formats, there are also certain fields of the BITMAPINFOHEADER structure that must be set to the correct values:

1. The biCompression member must be set to either BI_RGB or BI_BITFIELDS. Using BI-RGB indicates that no bit masks are included in the color table and that the default (RGB555 for 16bpp and RGB888 for 32bpp) format is implied. Using BI_BITFIELDS indicates that there are masks (bit fields) specified in the color table.

2. As with 24-bits-per-pixel formats, the biClrUsed member specifies the size of the color table used to optimize performance of Windows color palettes. If the biCompression is set to BI_BITFIELDS, then the optimal color palette starts immediately following the three DWORD masks. Note that an optimal color palette is optional and many applications will ignore it.

A technical note related to this subject from the Microsoft Multimedia group is also available. It can be obtained from CompuServe in the WINEXT and WINSDK forums. The filename is VFW.ZIP. In addition, the technote is available by calling Microsoft Developer Services at (800) 227-4679, extension 11771. The technical note is part of the Video for Windows technical notes and describes how to create a display driver that supports these new DIB formats, which are used by Video for Windows. The technical note also includes definitions of installable image codecs.

Windows 95

In Windows 95, if the BI_BITFIELDS flag is set, then a color mask must be specified and it must be one of the following:

Resolution	Bits per color	Color Mask
16bpp	5,5,5	0x00007c00 0x000003e0 0x0000001f
16bpp	5,6,5	0x0000f800 0x000007e0 0x0000001f
32bpp	8,8,8	0x00ff0000 0x0000ff00 0x000000ff

User-defined color masks are not available in Windows 95.

Additional reference words: 3.10 3.50 4.00 bpp bmp
KBCategory: kbprg kbfile
KBSubcategory: GdiBmp

SAMPLE: Adding TrueType, Raster, or Vector Fonts to System

PSS ID Number: Q130459
Authored 21-May-1995 Last modified 20-Jun-1995

--
The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), versions 3.1 and 3.11
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0
--

FONTINST is a sample application in the Microsoft Software Library that demonstrates how to programmatically add a TrueType, raster, or vector font to the system.

When working with a TrueType font file (.TTF file), FONTINST creates a .FOT file and moves the specified .TTF file to the Windows system directory. For a raster or vector font file (.FON file), FONTINST moves the .FON file to the Windows system directory. After the font file has been moved to the system directory, FONTINST adds the font to the system by using the AddFontResource() API. Then it adds font information to the [fonts] section of the WIN.INI file so that the font is automatically loaded every time Windows starts. For example, the following line is added to the WIN.INI file when FONTINST adds the ARIAL.TTF file to the system:

```
   Arial (TrueType)=ARIAL.FOT
```

FONTINST also demonstrates how to retrieve the facename of a font given a .TTF or .FON file. In the case of a TrueType font, FONTINST opens up the file and reads the naming table of the .TTF file. FONTINST also shows how to read the FONTINFO structure (as described on pages 49-50 of the "Microsoft Windows Software Development Kit: Programmer's Reference, Volume 4: Resources") of a .FON file. The facename of a font in a .FON file can also be found in this manner.

After a font is installed by FONTINST, information about the font is displayed in the window. Information such as the TEXTMETRIC structure and font type, as well as sample font text, is displayed. Information added to the WIN.INI file is also displayed in this window.

Download FONTINST.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for FONTINST.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download FONTINST.EXE

- Internet (anonymous FTP)

```
      ftp ftp.microsoft.com
      Change to the \SOFTLIB\MSLFILES directory
      Get FONTINST.EXE

Additional reference words: 3.10 3.50 4.00 95 softlib
KBCategory: kbprg kbfile
KBSubcategory: GdiFnt GdiFntCreate
```

SAMPLE: AngleArc in Windows 3.1, Win32s, and Windows 95

PSS ID Number: Q125693
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK
- Microsoft Win32s, version 1.3

SUMMARY

In Windows version 3.1, Win32s, and Windows 95, you may find it useful to get the functionality provided by the Win32 API AngleArc(). AngleArc() is only supported on Windows NT.

MORE INFORMATION

The AngleArc() function draws a line segment and an arc. The line segment is drawn from the current position to the beginning of the arc. The arc is drawn along the perimeter of a circle with the given radius and center. The length of the arc is defined by the given start and sweep angles. The starting point of the sweep is determined by measuring counterclockwise from the x-axis of the circle by the number of degrees in the start angle. The ending point is similarly located by measuring counterclockwise from the starting point by the number of degrees in the sweep angle.

The code below provides two possible ways of getting functionality similar to that of the AngleArc() function. While both of these methods will work on any Windows platform, the second (AngleArc2) will be substantially faster due to the fact that it uses the Arc() function to draw the sweep rather than calculating each of the segments on the perimeter of the arc.

NOTE: One limitation of the second method is that if the sweep angle is greater than 360 degrees, the arc will not be swept multiple times. In most cases this will not be a problem but in certain cases (constructing paths, for example) this can be a problem.

SAMPLE CODE #1

```
BOOL AngleArc1(HDC hdc, int X, int Y, DWORD dwRadius,
               float fStartDegrees, float fSweepDegrees)
{
  float fCurrentAngle;          // Current angle in radians
  float fStepAngle = 0.03f;  // The sweep increment value in radians
  float fStartRadians;       // Start angle in radians
  float fEndRadians;         // End angle in radians
  int ix, iy;                // Current point on arc
  float fTwoPi = 2.0f * 3.141592f;
```

```
  /* Get the starting and ending angle in radians */
  if (fSweepDegrees > 0.0f) {
    fStartRadians = ((fStartDegrees / 360.0f) * fTwoPi);
    fEndRadians = (((fStartDegrees + fSweepDegrees) / 360.0f) *
           fTwoPi);
  } else {
    fStartRadians = (((fStartDegrees + fSweepDegrees)  / 360.0f) *
          fTwoPi);
    fEndRadians =  ((fStartDegrees / 360.0f) * fTwoPi);
  }

  /* Calculate the starting point for the sweep via */
  /* polar -> cartesian conversion */
  ix = X + (int)((float)dwRadius * (float)cos(fStartRadians));
  iy = Y - (int)((float)dwRadius * (float)sin(fStartRadians));

  /* Draw a line to the starting point */
  LineTo(hdc, ix, iy);

  /* Calculate and draw the sweep */
  for (fCurrentAngle = fStartRadians;
       fCurrentAngle <= fEndRadians;
       fCurrentAngle += fStepAngle) {

    /* Calculate the current point in the sweep via */
    /* polar -> cartesian conversion */
    ix = X + (int)((float)dwRadius * (float)cos(fCurrentAngle));
    iy = Y - (int)((float)dwRadius * (float)sin(fCurrentAngle));

    /* Draw a line segment to current point */
    LineTo(hdc, ix, iy);
  }

  return TRUE;
}
```

SAMPLE CODE #2

```
BOOL AngleArc2(HDC hdc, int X, int Y, DWORD dwRadius,
               float fStartDegrees, float fSweepDegrees)
{
  int iXStart, iYStart;  // End point of starting radial line
  int iXEnd, iYEnd;        // End point of ending radial line
  float fStartRadians;   // Start angle in radians
  float fEndRadians;     // End angle in radians
  BOOL bResult;            // Function result
  float fTwoPi = 2.0f * 3.141592f;

  /* Get the starting and ending angle in radians */
  if (fSweepDegrees > 0.0f) {
    fStartRadians = ((fStartDegrees / 360.0f) * fTwoPi);
    fEndRadians = (((fStartDegrees + fSweepDegrees) / 360.0f) *
          fTwoPi);
```

```
  } else {
    fStartRadians = (((fStartDegrees + fSweepDegrees)  / 360.0f) *
          fTwoPi);
    fEndRadians =  ((fStartDegrees / 360.0f) * fTwoPi);
  }

  /* Calculate a point on the starting radial line via */
  /* polar -> cartesian conversion */
  iXStart = X + (int)((float)dwRadius * (float)cos(fStartRadians));
  iYStart = Y - (int)((float)dwRadius * (float)sin(fStartRadians));

  /* Calculate a point on the ending radial line via */
  /* polar -> cartesian conversion */
  iXEnd = X + (int)((float)dwRadius * (float)cos(fEndRadians));
  iYEnd = Y - (int)((float)dwRadius * (float)sin(fEndRadians));

  /* Draw a line to the starting point */
  LineTo(hdc, iXStart, iYStart);

  /* Draw the arc */
  bResult = Arc(hdc, X - dwRadius, Y - dwRadius,
                     X + dwRadius, Y + dwRadius,
                     iXStart, iYStart,
                     iXEnd, iYEnd);

  /* Move to the ending point - Arc() wont do this and ArcTo() */
  /* wont work on Win32s or Win16 */
  MoveToEx(hdc, iXEnd, iYEnd, NULL);

  return bResult;
}
```

Additional reference words: 1.30 3.10 4.00
KBCategory: kbprg kbcode
KBSubcategory: GdiMisc

SAMPLE: Changing Text Alignment in an Edit Control Dynamiclly

PSS ID Number: Q66942
Authored 14-Nov-1990 Last modified 22-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

A Microsoft Windows edit control aligns its contents to the left or right margins, or centers its contents, depending on the window style of the control. The control styles ES_LEFT, ES_CENTER, and ES_RIGHT specify left-, center-, and right-alignment, respectively.

Only multiline edit controls can be right-aligned or centered. Single-line edit controls are always left-aligned, regardless of the control style given.

Windows does not support altering the alignment style of a multiline edit control after it has been created. However, there are two methods that you can use to cause a multiline edit control in a dialog box to appear to change alignment. Note that in each of these methods, the dialog box that contains the control must be created with the DS_LOCALEDIT style.

MORE INFORMATION

The first method applies to all platforms. The second method does not apply to Windows 95. Under Windows 95, EM_SETHANDlE and EM_GETHANDLE are not supported. For more information, please see the following articles in the Microsoft Knowledge Base:

```
   ARTICLE-ID: Q130759
   TITLE     : EM_SETHANDLE and EM_GETHANDLE Messages Not Supported
```

Method 1

Create three controls: one left-aligned, one centered, and one right-aligned. Each has the same dimensions and position in the dialog box, but only one is initially made visible.

When the alignment is to change, call ShowWindow() to hide the visible control and to make one of the other controls visible.

To keep the text identical in all three controls, use the EM_GETHANDLE and EM_SETHANDLE messages to share the same memory among all three controls.

Method 2

Initially create a single control. When the text alignment is to change, retrieve location, size, and style bits for the existing edit control. Create a new control with the same size and in the same location, but change the style bits to reflect the new alignment.

Send the EM_GETHANDLE to each control to retrieve a handle to the memory that stores the contents. Send an EM_SETHANDLE to each control to exchange the memory used by each. Finally, destroy the original control.

There is a sample application named EDALIGN in the Microsoft Software Library that demonstrates each of these methods. Note, however, that this sample is a Windows 3.1 sample only.

Download EDALIGN.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for EDALIGN.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download EDALIGN.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get EDALIGN.EXE

Additional reference words: 3.00 3.10 3.50 4.00 95 EDALIGN.EXE
KBCategory: kbprg kbfile
KBSubcategory: UsrCtl

SAMPLE: Customizing the TOOLBAR Control

PSS ID Number: Q125683
Authored 01-Feb-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

The sample BARSDI demonstrates how to provide Customization features for the Toolbar Common Control. The Toolbar Common Control under Windows 95 provides Customization features that are useful when the user needs to change the toolbar control's buttons dynamically (add, delete, interchange, etc. buttons).

Download BARSDI.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for BARSDI.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download BARSDI.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the SOFTLIB\MSLFILES directory
  Get BARSDI.EXE

MORE INFORMATION

There are two ways the user can cusotmize the toolbar.

First, the user can use the Drag Drop Customization process to delete or change the position of buttons on the toolbar. This method does not allow the user to add buttons to the toolbar dynamically.

The second method involves displaying the Customize dialog box through which the user can add, remove, interchange buttons on the toolbar.

To provide Customization, the toolbar control has to be created with the CCS_ADJUSTABLE style, and the parent of the toolbar control has to process a series of TBN_XXXX notifications. The BARSDI sample implements both methods of Customization.

Method 1: Drag Drop Customization

This method of toolbar customization allows the user to reposition or delete buttons on the toolbar. The user initiates this operation by holding down the SHIFT key and begins dragging a button. The toolbar control handles all of the drag operations automatically, including the cursor changes.

To delete a button, the user has to release the drag operation outside the Toolbar control. The Toolbar control sends the TBN_QUERYDELETE message to its parent window. The parent window can return TRUE to allow the button to be deleted and FALSE to prevent the button from being deleted.

If the application wants to do custom dragging, it has to process the TBN_BEGINDRAG and TBN_ENDDRAG notifications itself and perform the drag/drop process, which involves more coding.

Method 2: Customization Dialog Box

This method of customization allows users to add buttons to the toolbar dynmacally in addition to deleting and rearranging buttons on the toolbar. For example, if the toolbar has N total buttons, and displays only 10 of those buttons initially, the bitmap that was used to create the toolbar, should contian all N buttons (where N > 10).

There are two ways in which the Toolbar control dislpays the customize dialog box. The user can bring up the Customization dialog box by double-clicking the left mouse button on the toolbar control or the application can send the TB_CUSTOMIZE message to the toolbar control.

The Customize dialog box displayed by the Toolbar control has two list boxes. One, on the left contains the list of N-10 Buttons that were not displayed on the initial toolbar, and the one on the right will have the currently displayed buttons on the toolbar. The toolbar control provides the add, remove and other features in the Customize dialog box.

Here is a code sample that shows how the Customization feature is implemented:

SAMPLE CODE

```
// The initial set of toolbar buttons.

TBBUTTON tbButton[] =
{
    {0,   IDM_FILENEW,     TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
    {1,   IDM_FILEOPEN,    TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
    {2,   IDM_FILESAVE,    TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
    {3,   IDM_EDITCUT,     TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
    {0,   0,               TBSTATE_ENABLED, TBSTYLE_SEP,    0, 0},
    {4,   IDM_EDITCOPY,    TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
    {5,   IDM_EDITPASTE,   TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
    {6,   IDM_FILEPRINT,   TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
```

```
    {0,   0,               TBSTATE_ENABLED, TBSTYLE_SEP,    0, 0},
    {7,   IDM_ABOUT,       TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
};

// Buttons that can be added at a later stage.

TBBUTTON tbButtonNew[] =
{
    { 8,  IDM_ERASE,       TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
    { 9,  IDM_PEN,         TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
    {10,  IDM_SELECT,      TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
    {11,  IDM_BRUSH,       TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
    {12,  IDM_AIRBRUSH,    TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
    {13,  IDM_FILL,        TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
    {14,  IDM_LINE,        TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
    {15,  IDM_EYEDROP,     TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
    {16,  IDM_ZOOM,        TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
    {17,  IDM_RECT,        TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
    {18,  IDM_FRAME,       TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
    {19,  IDM_OVAL,        TBSTATE_ENABLED, TBSTYLE_BUTTON, 0, 0},
};

// The bitmap that is used to create the toolbar should have all
// tbButtonNew + tbButton buttons = 20 in this case.

// Use tbButtons array to create the initial toolbar control.

// Once the user starts to customize the toolbar, process the WM_NOTIFY
// message and the following notifications.
// The toolbar control sends a WM_NOTIFY message to the parent window
// during each process of the customization.

LRESULT OnMsgNotify(HWND hwnd, UINT uMessage, WPARAM wparam, LPARAM lparam)
{
    LPNMHDR         lpnmhdr;
    lpnmhdr =       (LPNMHDR)lparam;

// process the QUERYINSERT And QUERYDELETE notifications
// to allow the drag/drop operation to succeeed.
    if (lpnmhdr->code == TBN_QUERYINSERT)
        return TRUE;
    else if (lpnmhdr->code == TBN_QUERYDELETE)
        return TRUE;
    else if (lpnmhdr->code == TBN_GETBUTTONINFO)
// The user has brought up the customization dialog box,
// so provide the the control will button information to
// fill the listbox on the left side.
    {
        LPTBNOTIFY lpTbNotify = (LPTBNOTIFY)lparam;
        char  szBuffer [20];
        if (lpTbNotify->iItem < 12) // 20 == the total number of buttons
        {                           // tbButton and tbButtonNew
                                    // Since initially we displayed
                                    // 8 buttons
                  // send back information about the rest of
```

```
                    // 12 buttons that can be added the toolbar.

            lpTbNotify->tbButton = tbButtonNew[lpTbNotify->iItem];

            LoadString(hInst,
                       NEWBUTTONIDS + lpTbNotify->iItem, // string
                                                         //ID == command ID
                       szBuffer,
                       sizeof(szBuffer));

            lstrcpy (lpTbNotify->pszText, szBuffer);
            lpTbNotify->cchText = sizeof (szBuffer);
            return TRUE;
        }
        else
        return 0;
    }
}
```

Additional reference words: 4.00 BARSDI
KBCategory: kbprg kbcode kbfile
KBSubcategory: UsrCtl

SAMPLE: Drawing Three-Dimensional Text in OpenGL Applications

PSS ID Number: Q131024
Authored 02-Jun-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 3.51

SUMMARY

GDI operations, such as TextOut, can be performed on an OpenGL window only if the window is single-buffered. The Windows NT implementation of OpenGL does not support GDI graphics in a double-buffered window. Therefore, you can not use GDI functions to draw text in a double-buffered window, for example. To draw text in a double-buffered window, an application can use the wglUseFontBitmaps and wglUseFontOutlines functions to create display lists for characters in a font and then draw the characters in the font with the glCallLists function.

The wglUseFontOutlines function is new to Windows NT 3.51 and can be used to draw 3-D characters of TrueType fonts. These characters can be rotated, scaled, transformed, and viewed like any other OpenGL 3-D image. This function is designed to work with TrueType fonts.

The GLFONT sample shows how to use the wglUseFontOutlines function to create display lists for characters in a TrueType font and how to draw, scale, and rotate the glyphs in the font by using glCallLists to draw the characters and other OpenGL functions to rotate and scale them. You will need the Win32 SDK for Windows NT 3.51 to compile this sample and to incorporate wglUseFontOutlines in your own application. You will need Windows NT 3.51 to execute the application.

Download GLFONT.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for GLFONT.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download GLFONT.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the SOFTLIB\MSLFILES directory

  Get GLFONT.EXE

MORE INFORMATION

To specify which TrueType font you want wglUseFontOutlines to create display lists for, you must first create the desired logical font with CreateFont or CreateFontIndirect. Then, you must select the HFONT created into a screen device context (HDC) with SelectObject and send the HDC to the wglUseFontOutlines function. Each characer is mapped in the "x" and "y" directions in the display lists and you specify the depth in the negative "z" direction in the "extrusion" parameter of wglUseFontOutlines.

You can also specify whether you want the 3-D glyphs to be created with line segments or polygons. To instruct wglUseFontOutlines to create the 3-D glyphs with lines segments, specify WGL_FONT_LINES in the "format" parameter and to create them with polygons, you need to specify WGL_FONT_POLYGONS.

Here is an example of how one could create a set of display lists to draw the characters of the "Arial" TrueType font as a set of polygons:

```
LOGFONT     lf;
HFONT       hFont, hOldFont;
GLYPHMETRICSFLOAT agmf[256];

// An hDC and an hRC have already been created.
wglMakeCurrent( hDC, hRC );

// Let's create a TrueType font to display.
memset(&lf,0,sizeof(LOGFONT));
lf.lfHeight               =   -20 ;
lf.lfWeight               =   FW_NORMAL ;
lf.lfCharSet              =   ANSI_CHARSET ;
lf.lfOutPrecision         =   OUT_DEFAULT_PRECIS ;
lf.lfClipPrecision        =   CLIP_DEFAULT_PRECIS ;
lf.lfQuality              =   DEFAULT_QUALITY ;
lf.lfPitchAndFamily       =   FF_DONTCARE|DEFAULT_PITCH;
lstrcpy (lf.lfFaceName, "Arial") ;

hFont = CreateFontIndirect(&lf);
hOldFont = SelectObject(hDC,hFont);

// Create a set of display lists based on the TT font we selected
if (!(wglUseFontOutlines(hDC, 0, 255, GLF_START_LIST, 0.0f, 0.15f,
   WGL_FONT_POLYGONS, agmf)))
     MessageBox(hWnd,"wglUseFontOutlines failed!","GLFont",MB_OK);

DeleteObject(SelectObject(hDC,hOldFont));
.  .  .  .
.  .  .  .
.  .  .  .
.  .  .  .
```

To display these 3-D characters in a string, use the following code:

```
// Display string with display lists created by wglUseFontOutlines()
glListBase(GLF_START_LIST); // indicate start of display lists
```

```
   // Draw the characters
   glCallLists(6, GL_UNSIGNED_BYTE, "OpenGL");
```

Additional reference words: graphics
KBCategory: kbprg kbcode kbfile
KBSubcategory: GdiOpenGL

SAMPLE: Drawing to a Memory Bitmap for Faster Performance

PSS ID Number: Q130805
Authored 28-May-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 Software Development Kit (SDK) versions 3.5, 4.0
- Microsoft Win32s version 1.2

An application whose client area is a complex image can realize a performance benefit from drawing to a memory bitmap. The complex, time consuming drawing code need only be performed once - to initialize the offscreen bitmap. During the handling of the WM_PAINT message, the only work that needs to be done is a simple BitBlt from the memory bitmap to the screen.

Sample code demonstrating this technique is available in the Microsoft Software Library. The MemDC sample code draws a complex pattern on its client area. A menu option toggle allows the user to see the speed difference between using and not using the offscreen bitmap.

Download MEMDC.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for MEMDC.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download MEMDC.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get MEMDC.EXE

Additional reference words: 1.20 3.10 3.50 4.00 95 device context memory DC speed fast buffer
KBCategory: kbgraphic kbfile
KBSubcategory: GdiDc

SAMPLE: Fade a Bitmap Using Palette Animation

PSS ID Number: Q130804
Authored 28-May-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.5, 3.51, and 4.0

SUMMARY

PALFADE is a sample application available in the Microsoft Software Library. It demonstrates:

- How to use the AnimatePalette function to fade a bitmap to black.

- How to use the DIBAPI32.DLL library that can be built by the WINCAP32 sample that ships with the Microsoft Win32 SDK.

To perform palette animation, the sample creates a logical palette for a device-independent bitmap (DIB) with the PC_RESERVED flag set for each palette entry. PALFADE loads, displays, and animates both Windows-style and OS/2-style DIB files.

Download PALFADE.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for PALFADE.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download PALFADE.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the SOFTLIB\MSLFILES directory
  Get PALFADE.EXE

MORE INFORMATION

Before performing palette animation on a logical palette entry, ensure that the palette entry has the PC_RESERVED flag set. To fade a bitmap drawn on a device context with a PC_RESERVED palette selected, you can lower the RGB values for each color in the palette in a loop until all colors are black.

The default system palette contains 20 static colors. These static colors take up the first ten and last ten colors of the system palette; these palette entries are not available for animation. If you try to fade a

bitmap that has 256 unique colors by creating a 256-color palette with each palette entry set to PC_RESERVED, you are not guaranteed that every logical palette entry will map to an entry in the system palette that is available for palette animation.

One solution to this is to create a logical palette that contains only 236 colors. The PALFADE sample demonstrates one way to create an optimal palette of 236 colors given a device-independent bitmap with 256 colors in its color table.

Given a 256-color DIB, PALFADE traverses through every bit in the bitmap to find the least-used 20 colors in the color table. Then it creates a logical palette out of the 236 most-used colors. This ensures that all entries in the logical palette will animate.

This sample uses many of the DIB support functions included with the DIBAPI32.DLL library. It does not use the LoadDIB() function, because it was not written to handle OS/2-style DIB files. Instead, PALFADE implements the DIB-loading routines found in the Win32 SDK SHOWDIB sample.

NOTE: DIBAPI32.DLL is included with this sample.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg kbfile kbgraphic
KBSubcategory: GdiPal

SAMPLE: FASTBLT Implements Smooth Movement of a Bitmap

PSS ID Number: Q40959
Authored 07-Feb-1989 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

There is a sample application called FASTBLT in the Microsoft Software Library that demonstrates how to implement the smooth movement of a bitmap around the screen. Basically, it sets up a pair of BitBlt() calls: one that erases the image and another that redisplays the image. The necessary ROP codes for BitBlt() that should be used are SRCCOPY and SRCINVERT.

Download FASTBLT.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for FASTBLT.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download FASTBLT.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get FASTBLT.EXE

Additional reference words: 3.00 3.10 3.50 4.00 95 softlib FASTBLT.EXE
KBCategory: kbprg kbfile
KBSubcategory: GdiBmp

SAMPLE: Highlighting an Entire Row in a ListView Control

PSS ID Number: Q131788
Authored 20-Jun-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.51, 4.0
- Microsoft Win32s version 1.3

SUMMARY

One of the limitations of the ListView common control is that when the control is in report view, the control only highlights the first column when a row is selected. To work around this limitation, you can create the ListView as an owner draw control (using the LVS_OWNERDRAWFIXED style) and perform all the painting yourself.

The ODLISTVW sample demonstrates how to create an owner draw ListView control that highlights an entire row.

Download ODLISTVW.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for ODLISTVW.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download ODLISTVW.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the SOFTLIB\MSLFILES directory
  Get ODLISTVW.EXE

MORE INFORMATION

One of the supported styles for the ListView control is LVS_OWNERDRAWFIXED, which allows the program to perform all the drawing of items in the ListView control. Whenever the ListView control needs to have a portion of the control repainted, it calculates which items are in that area and sends a WM_DRAWITEM message to its parent for each item. If any part of an item needs to be redrawn, the ListView sends the WM_DRAWITEM with the update rectangle set to the entire item.

The ListView control handles owner drawn items differently from other owner drawn controls. Previous owner drawn controls use the DRAWITEMSTRUCT's itemAction field to let the parent know if it needs to draw the item, change the selected state, or change the focus state. The ListView control

always sends the WM_DRAWITEM message with the itemAction set to ODA_DRAWENTIRE. The parent needs to check the itemState to see if the focus or selection needs to be updated.

Additional reference words: 1.30 4.00 95 3.51
KBCategory: kbui kbprg kbfile
KBSubcategory: UsrCtl

SAMPLE: How to Simulate Multiple-Selection TreeView Control

PSS ID Number: Q125587
Authored 31-Jan-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

The TVWSTATE sample demonstrates how to simulate a multiple-selection TreeView control. The Windows 95 TreeView control does not support multiple selection. If you want a multiple-selection TreeView, you can use state images to simulate it in your application.

The TVWSTATE sample accomplishes this by using a checkbox type of state image to indicate that the item is selected or cleared (de-selected). These checkboxes will retain their state even if the TreeView loses focus.

Download TVWSTATE.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for TVWSTATE.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download TVWSTATE.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the SOFTLIB\MSLFILES directory
  Get TVWSTATE.EXE

MORE INFORMATION

The multiple-selection TreeView control simulation is implemented by setting the state image list of the TreeView control to an image list that contains the checked and unchecked checkbox bitmaps. This image list is set by using the TVM_SETIMAGELIST message with lParam == TVSIL_STATE (see InitImageList in TVWSTATE.C). A TreeView control can have two image lists, a normal image list and a state image list. In the TreeView, the display order from left to right is: the expansion button, the state image (if present), the normal image (if present), and then the item text.

When processing the WM_NOTIFY message where (LPNMHDR)lParam->code == NM_CLICK (see MsgNotifyTreeView in TVWSTATE.C), the code checks to see if the user clicked the left mouse button in the checkbox. If this is the case, the state image index of the item is retrieved, the index is toggled

between the checked and unchecked image list items, and then the new index is saved.

The state image index identifies which member of the state image list should be displayed. The state image index is stored in bits 12-16 of the item state value. Either TVIS_STATEIMAGEMASK or TVIS_USERMASK can be used to mask off the lower bits. To access just the state image index, use a statement similar to this:

```
StateIndex = tvi.state & TVIS_STATEIMAGEMASK;
```

The INDEXTOSTATEIMAGEMASK macro offsets a value to the correct bits for the state image index. This is accomplished by shifting the given value left 12 places. If the desired state image index is 1, the state can be set using a statement similar to this:

```
tvi.state = INDEXTOSTATEIMAGEMASK(1);
```

This sample can also be modified to implement selection methods similar to those of an extended-selection listbox where the user uses the SHIFT key to select a range of items and/or the CTRL key to select or clear individual items.

Additional reference words: 3.51 4.00
KBCategory: kbprg kbcode kbfile
KBSubcategory: UsrCtl

SAMPLE: How to Use File Associations

PSS ID Number: Q122787
Authored 13-Nov-1994 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK), version 3.1
- Microsoft Win32 SDK, versions 3.1 and 3.5

SUMMARY

Windows provides file associations so that an application can register the type of documents it supports. The benefit of doing this is that it allows the user to double-click or select a document in File Manager to edit or print it. File association is also supported by the ShellExecute() API. File associations also allow the user to open multiple documents with a single instance of the application via the File Manager.

ShellExecute() has even more benefit in Windows 95.

The sample FILEASSO.EXE demonstrates how to use file associations. Download FILEASSO.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for FILEASSO.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download FILEASSO.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the SOFTLIB\MSLFILES directory
  Get FILEASSO.EXE

MORE INFORMATION

The following information applies to both File Open and File Print. For ease of reading, this article will discuss File Open to explain how File Associations work.

When the user double-clicks a document, the File Manager calls ShellExecute() with filename. ShellExecute() checks the Registration Database for an entry that associates that file extension with a particular application. If an entry exists and does not specify DDE commands, then ShellExecute() launches the application as specified in the registry. If the registry specifies to use DDE commands, ShellExecute() attempts to establish a DDE conversation with that application using the application

topic. If an application responds to the DDE connections, ShellExecute() sends a DDE Execute command, as specified in the registry. It is up to the application to define the specifics on this conversation, particularly the service and topic name to connect to, and also the correct DDE execute command syntax to use. However, if attempts to establish the conversation fail, ShellExecute() launches the application specified in the regiistry and tries to establish the DDE connection again.

There is one more option available when the application is not running, which seems to be appropriate for File Print. In this option, ShellExecute() sends a different Execute statement, the application needs to Open and Print the document. When the Printing is done, it exits.

There are two steps for an application to open multiple documents through single application instance via File association. As an example, assume MyApp is the application and AssocSupport is the topic. Most applications use MyApp as their application name and System as the topic.

1. When the application starts, register a DDE Server with the application name and topic (for example MyApp, and AssocSupport). The application also has to support DDE Execute Statements. The execute statement could be any format; at minimum, it should be:

   <Action> <fileName> <options>

   Here <Action> is anything specifying unique identification of the action, such as Open or Print. The <fileName> is the file that should be operated on. Finally, <options> can be any options that need to be passed on.

   A typical Execute Statement follows this format:

   [<Action>(<FileName>)]

   For example, Microsoft Word uses:

   [Open("%1")]

   The Application has to support the required functionality for executed statements.

2. File association can be done in Windows NT via File Manager or regedit.

Using the File Manager to Set File Associations

When associating a file type using the File Manager, choose Associate from the File menu. The Associate dialog presents the list of existing file associations. This dialog allows you to add a new file type (or file association), modify an existing file type, or delete an existing file type. The New File Type button allows the user to add an association for a new file extension. Here are the steps:

1. Add a File type name. For example, name it "Microsoft Word 6.0 Document."

2. Choose an action (Open or Print). For example, select the Uses DDE check box.

3. Add the directory path and application name. For example, enter WINWORD as the application.

4. Select the option Uses DDE.

5. Set the Application as the DDE Server Name.

6. Set the Topic as the DDE Server. For example enter System as the Topic.

7. Set the DDE Message <Action> <fileName> <options> to be the same as your application's Execute Statement. However the <fileName> and <options> should be replaced by whatever the command line arguments are. For example use:

```
DDE Message : [FileOpen("%1")]
```

Using Regedit in Windows NT to Set File Associations

NOTE: Regedit is available only in Windows NT, not in Windows version 3.1.

The user can also associate files with an application by using regedit. From the Edit menu, choose Add File Type or Modify File Type (to modify an existing file type). A dialog similar to File Manager Associate dialog appears. Follow the same steps as described for File Manager. In Windows version 3.1, once you have defined a File Type via this method, go to the File Manager associate dialog and attach the file type to the extension.

Using a Program to Set File Assoications

You can also set the associations programmatically. This is useful when setting up your application on other machines. You would provide this functionality through your installation program. The first way to do this (the simplier method) is to use regedit to merge the changes from a file. The syntax for this is:

```
regedit <filename>.reg
```

An example of a <filename>.reg is:

```
REGEDIT
HKEY_CLASSES_ROOT\.riy = FMA000_File_assoc
HKEY_CLASSES_ROOT\FMA000_File_assoc = File_assoc
HKEY_CLASSES_ROOT\FMA000_File_assoc\shell\open\command = fileasso.EXE
HKEY_CLASSES_ROOT\FMA000_File_assoc\shell\open\ddeexec = [Open(%1)]
HKEY_CLASSES_ROOT\FMA000_File_assoc\shell\open\ddeexec\application
= Myserver
HKEY_CLASSES_ROOT\FMA000_File_assoc\shell\open\ddeexec\topic = system
HKEY_CLASSES_ROOT\FMA000_File_assoc\shell\print\command = fileasso.EXE
```

```
   HKEY_CLASSES_ROOT\FMA000_File_assoc\shell\print\ddeexec = [Open(%1)]

HKEY_CLASSES_ROOT\FMA000_File_assoc\shell\print\ddeexec\application =
MYServer
   HKEY_CLASSES_ROOT\FMA000_File_assoc\shell\print\ddeexec\topic = System
   HKEY_CLASSES_ROOT\FMA000_File_assoc\shell\print\ddeexec\ifexec =
[Test(%1)
]
```

In the program, you can also add keys to the registry by using the registry APIs. The developer needs to add the following keys to the registration database:

```
   // Your extensions.
   HKEY_CLASSES_ROOT\.riy = FMA000_File_assoc

   //File type name.
   HKEY_CLASSES_ROOT\FMA000_File_assoc = File_assoc

   // Command to execute when application is not running or dde is not
   // present and Open command is issued.
   HKEY_CLASSES_ROOT\FMA000_File_assoc\shell\open\command = fileasso.EXE

   // DDE execute statement for Open.
   HKEY_CLASSES_ROOT\FMA000_File_assoc\shell\open\ddeexec = [Open(%1)]

   // The server name your application responds to.
   HKEY_CLASSES_ROOT\FMA000_File_assoc\shell\open\ddeexec\application =
   Myserver

   // Topic name your application responds to.
   HKEY_CLASSES_ROOT\FMA000_File_assoc\shell\open\ddeexec\topic = system

   // Command to execute when application is not running or dde is not
   // present and print command is issued.
   HKEY_CLASSES_ROOT\FMA000_File_assoc\shell\print\command = fileasso.EXE

   // DDE execute statement for Print.
   HKEY_CLASSES_ROOT\FMA000_File_assoc\shell\print\ddeexec = [Open(%1)]

   // The server name your application responds to.
   HKEY_CLASSES_ROOT\FMA000_File_assoc\shell\print\ddeexec\application =
   MYServer

   // Topic name your application responds to.
   HKEY_CLASSES_ROOT\FMA000_File assoc\shell\print\ddeexec\topic = System

   // DDE execute statement for print if the application is not already
   // running. This gives the options for a an application to Run, Print
   // and Exit.
   HKEY_CLASSES_ROOT\FMA000_File assoc\shell\print\ddeexec\ifexec =
   [Test(%1)]
```

REFERENCES

Windows SDK Programmers Reference, Volume 1, chapter 7, Shell Library or Books Online.

Window 3.1 SDK Help file, Registration Database, Shell Library Functions.

Win32 Programmers Reference, Volume 2, chapter 52, Registry and Initialization Files or Books Online.

Win32 SDK Help file Registry and Initialization

File Manager Help File.

REGEDIT.HLP

REGEDT32.HLP

Additional reference words: 3.10 3.50
KBCategory: kbprg kbfile
KBSubcategory: UsrMisc

SAMPLE: How to Use Paths to Create Text Effects

PSS ID Number: Q128091
Authored 26-Mar-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

This article defines the term "path" for the purposes of this article, and it explains how you can get sample code (provided in TEXTFX.EXE, a self-extracting file) that shows by example how to use paths to draw text at varying angles, orientations, and sizes. In addition, the sample code gives useful routines for displaying path data.

Download TEXTFX.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for TEXTFX.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download TEXTFX.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get TEXTFX.EXE

MORE INFORMATION

A path is one or more figures (or shapes) that are filled, outlined, or both filled and outlined. Computer-aided design (CAD) applications use paths to create unique clipping regions, to draw outlines of irregular shapes, and to fill the interiors of irregular shapes.

A path is associated with a Device Context (DC) but unlike other objects associated with a DC, such as pens and brushes, a path has no default object.

To create a path, you first call BeginPath(). Then use the drawing functions in the table below to add to the path. Any drawing done using these functions is recorded as part of the path. When you finish building the path, call EndPath(). The new path can then be converted to a region by using PathToRegion(), selected as a clipping region for a device context by using SelectClipPath(), and rendered by using StrokePath() or FillPath().

In addition, as this sample illustrates, the path can be retrieved by using GetPath() and manipulated programmatically.

Functions supported in paths:

AngleArc	LineTo	Polyline
Arc	MoveToEx	PolylineTo
ArcTo	Pie	PolyPolygon
Chord	PolyBezier	PolyPolyline
CloseFigure	PolyBezierTo	Rectangle
Ellipse	PolyDraw	RoundRect
ExtTextOut	Polygon	TextOut

Functions not supported under Windows 95:

AngleArc
ArcTo
PolyDraw

Functions not supported in a path under Windows 95:

Arc
Chord
Ellipse
Pie
Rectangle
RoundRect

MORE INFORMATION

The following path functions are used in the TextFX sample:

- BeginPath
- EndPath
- GetPath
- FillPath
- StrokePath

To use TextFX, run it, and then draw two lines into the client area (they don't have to be straight). The first line appears as blue and the second appears as red. These lines serve as guides for how the text will be rendered. After completing the second line, the text "This is a test" will be drawn so that it appears between the two guide lines.

To remap the text so that it appears between the two lines, TextFX first breaks down the guide lines (which are composed of line segments) into distinct adjacent points. The result is that the x,y position of each point in the lines is adjacent to its neighboring points x,y position.

Next, the text "This is a test" is drawn into a device context as a path. The points that make up the lines and curves in this path are then retrieved from the device context by using the GetPath() function.

To reposition the points in the path data, the code must establish a

relationship between the relative position of the points in the original text and the position defined by the guide lines. To establish this relationship, the code calculates the x and y positions of each point in the path data relative to the overall extent of the text string. The relative x position is used to calculate a corresponding point on each of the two the guide lines, while the relative y value is used as a weight to determine how far along on a line between the two guide line points the remapped position should be.

For example, if the point in the upper left corner of the "T" in the string "This is a test" is 2% of the total x extent of the string and 10% of the total y extent, then TextFX would find the point in each guide line that corresponds to 2% of the total number of points in that guide line. Then TextFX would reposition the point in the path data representing the upper left corner of the "T" so that it would be 10% of the way along an imaginary line extending from the point on the top guide line to the point on the bottom guide line.

Two different methods can be selected for drawing the remapped data, one draws just the outline of the characters, while the other fills in the characters.

To draw the outline of the characters, TextFX converts the remapped data back into a path and uses StrokePath() to display the outlines. To do the conversion, TextFX begins a new path, and then loops through the remapped data and uses the vertex types returned from GetPath() to determine how to draw the points. After drawing all the data, TextFX ends the path and calls StrokePath().

To draw the solid characters, instead of using StrokePath(), TextFX uses FillPath(). However, in order to get the interior areas of characters like "O", "A", "D", and so on, TextFX sets the ROP2 code to R2_MERGEPENNOT before calling FillPath(). This is done so that characters like "O" that consist of two separate polygons (one representing the outer perimeter and one representing the inner perimeter) will not be drawn as a solid blob. By drawing the polygons with the R2_MERGEPENNOT code, the code ensures that the second polygon will cancel the effects of the first in the area of the inner polygon.

REFERENCES

For additional information on paths, please see the PATHS sample included with the Win32 SDK.

Additional reference words: 3.10 3.50 4.00 stones effects effect font fx
KBCategory: kbprg kbfile kbcode
KBSubcategory: GdiDrw

SAMPLE: Implementing Multiple Threads in an OpenGL

PSS ID Number: Q128122
Authored 27-Mar-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

It is possible to create multiple threads in an OpenGL application and have each thread call OpenGL functions to draw an image. You might want to do this when multiple objects need to be drawn at the same time or when you want to have certain threads perform the rendering of specific types of objects.

This article explains how to obtain GLTHREAD, a sample that demonstrates how to implement multiple threads in an OpenGL application. The main process default thread creates two threads that each draw a three-dimensional wave on the main window. The first thread draws a wave on the left side of the screen. The second thread draws a wave on the right side of the screen. Both objects are drawn simultaneously, demonstrating OpenGL's ability to handle multiple threads.

How to Get the GLTHREAD Sample

Download GLTHREAD.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for GLTHREAD.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download GLTHREAD.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get GLTHREAD.EXE

MORE INFORMATION

When implementing multiple threads in any type of application, it is important to have adequate communication between threads. In OpenGL, it is important for two threads to know what the other thread is doing. For example, it is common practice to clear the display window before drawing

an OpenGL scene. If both threads are called to draw portions of a scene and they both try to call glClear before drawing, one thread's object may get erased by another thread's call to glClear.

The GLTHREAD sample assigns the glClear function to a single thread, and ensures that the other thread does not perform any drawing until glClear has been called. When a menu command message is sent to the main window, the application calls CreateThread twice to create two threads. Each thread calls GetDC(hwndMain) to obtain its own device context to the main window.

Then, each thread calls GLTHREAD's bSetupPixelFormat function to set up the pixel format and calls wglCreateContext to create a new OpenGL Rendering Context. Now, each thread has its own Rendering Context and both can call wglMakeCurrent to make its new OpenGL rendering context its (the calling thread's) current rendering context.

All subsequent OpenGL calls made by the thread are drawn on the device identified by the HDC returned from each thread's call to GetDC(). Now, because only one thread should call glClear, GLTHREAD has thread number one call it. The second thread is created "suspended" so it does nothing until a call to ResumeThread is made. After thread one has called glClear, it enables thread two to resume by calling ResumeThread with a handle to the second thread.

The procedure in the main thread that created the two other threads waits until both threads are finished before returning from the processing of the menu command message that is sent when the user selects the "Draw Waves" menu selection from the "Test Threads" menu. It will use the WaitForMultipleObjects function to do this.

Additional reference words: 3.10 3.50 4.00 95 GDI GRAPHICS THREADS
KBCategory: kbprg kbgraphic kbfile kbcode
KBSubcategory: codesam GdiDrwOpenGL

SAMPLE: MFCOGL a Generic MFC OpenGL Code Sample

PSS ID Number: Q127071
Authored 12-Mar-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.5, 3.51, and 4.0

- The Microsoft Foundation Classes (MFC) included with:
  - Microsoft Visual C++ 32-bit Edition, versions 2.0 and 2.1

SUMMARY

Microsoft Windows NT's OpenGL can be used with the Microsoft Foundation Class (MFC) library. This article gives you the steps to follow to enable MFC applications to use OpenGL.

The companion sample (MFCOGL) is a generic sample that demonstrates using OpenGL with MFC. Download MFCOGL.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for MFCOGL.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download MFCOGL.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the SOFTLIB\MSLFILES directory
  Get MFCOGL.EXE

MORE INFORMATION

Step-by-Step Example to Use OpenGL in MFC Application

1. Include the necessary header files to use OpenGL including:

   - "gl\gl.h" for all core OpenGL library fuctions. These functions have the "gl" prefix such as glBegin().

   - "gl\glu.h" for all OpenGL utility library functions. These functions have the "glu" prefix such as gluLookAt().

   - "gl\glaux.h" for all Windows NT OpenGL auxilary library functions.

These functions have the "aux" prefix such as auxSphere().

You don't need to add a header file for functions with the "wgl" prefix.

2. Add necessary library modules to the link project settings. These library modules include OPENGL32.LIB, GLU32.LIB, and GLAUX.LIB.

3. Add implementations for OnPaletteChanged() and OnQueryNewPalette() in CMainFrame class for palette-aware applications.

```
void CMainFrame::OnPaletteChanged(CWnd* pFocusWnd)
{
    CFrameWnd::OnPaletteChanged(pFocusWnd);

    if (pFocusWnd != this)
        OnQueryNewPalette();

}

BOOL CMainFrame::OnQueryNewPalette()
{
    WORD   i;
    CPalette   *pOldPal;
    CMfcOglView *pView = (CMfcOglView *)GetActiveView();
    CClientDC   dc(pView);


    pOldPal = dc.SelectPalette(&pView->m_cPalette, FALSE);
    i = dc.RealizePalette();
    dc.SelectPalette(pOldPal, FALSE);

    if (i > 0)
        InvalidateRect(NULL);

    return CFrameWnd::OnQueryNewPalette();
}
```

4. Use the one or more of the following classes derived from CWnd, including view classes, that will use OpenGL for rendering onto:

   - Implement PreCreateWindow() and add WS_CLIPSIBLINGS and WS_CLIPCHILDREN to the windows styles:

```
BOOL CMfcOglView::PreCreateWindow(CREATESTRUCT& cs)
{
    cs.style |= WS_CLIPSIBLINGS | WS_CLIPCHILDREN;

    return CView::PreCreateWindow(cs);
}
```

   - Implement OnCreate() to initialize a rendering context and make it current. Also, initialize any OpenGL states here:

```
int CMfcOglView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
```

```
    if (CView::OnCreate(lpCreateStruct) == -1)
        return -1;

    Init(); // initialize OpenGL

    return 0;
}
```

- Implement OnSize() if the window is sizeable:

```
void CMfcOglView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);

    if (cy > 0)
    {
        glViewport(0, 0, cx, cy);

        if ((m_oldRect.right > cx) || (m_oldRect.bottom > cy))
            RedrawWindow();

        m_oldRect.right = cx;
        m_oldRect.bottom = cy;

        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(45.0f, (GLdouble)cx / cy, 3.0f, 7.0f);
        glMatrixMode(GL_MODELVIEW);
    }
}
```

- Implement OpenGL rendering code. This can be done in OnDraw() or other application-specific places such as OnTimer().

- Implement clean-up code, which is typically done in OnDestroy():

```
void CMfcOglView::OnDestroy()
{
    HGLRC   hrc;

    if (m_nTimerID)
        KillTimer(m_nTimerID);

    hrc = ::wglGetCurrentContext();

    ::wglMakeCurrent(NULL,  NULL);

    if (hrc)
        ::wglDeleteContext(hrc);

    CPalette    palDefault;

    // Select our palette out of the dc
    palDefault.CreateStockObject(DEFAULT_PALETTE);
    m_pDC->SelectPalette(&palDefault, FALSE);
```

```
        if (m_pDC)
            delete m_pDC;

        CView::OnDestroy();
    }
```

Additional reference words: 3.50 3.51 2.00 2.10 3.00 3.10 4.00 95 graphics
KBCategory: kbprg kbcode kbfile
KBSubcategory: GdiOpenGL

SAMPLE: RASberry - an RAS API Demonstration

PSS ID Number: Q118983
Authored 03-Aug-1994 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows, version 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

RASberry is a sample application that demonstrates the Remote Access Service (RAS) API. RASberry allows the user to enumerate current RAS connections, display the status of a selected connection, dial entries from the default phone book, and hang up an active connection. The sample may be built for both Windows version 3.1 and Win32 environments.

RASBRY.EXE can be downloaded as a self-extracting file from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL and download RASBRY.EXE

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download RASBRY.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get RASBRY.EXE

MORE INFORMATION

The RAS APIs used in this sample are:

- RasDial
- RasEnumConnections
- RasEnumEntries
- RasGetConnectStatus
- RasGetErrorString
- RasHangUp

The RAS SDK for Win32 is included as part of the Win32 SDK.

The RAS SDK for Windows version 3.1 (which contains additional files that can be used with the Windows version 3.1 SDK for RAS development) is available on the Microsoft Developer's Network Level 2 CD set, beginning with the April 1994 edition.

Additional reference words: 3.10 3.50 4.00 95 softlib
KBCategory: kbnetwork kbprg kbfile
KBSubcategory: NtwkRAS

SAMPLE: RESIZE App Shows How to Resize a Window in Jumps

PSS ID Number: Q123605
Authored 05-Dec-1994 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

Sometimes it's useful to have a window that can only be certain sizes. For example, Microsoft Word and Microsoft Visual C++ have toolbars that are resizable only to particular sizes that best fit the controls in the toolbar. When you do this, it's a good idea to give the user visual cues about the available window sizes. The RESIZE sample code shows by example how to modify the way Windows resizes a window so that when a user uses the mouse to resize the window the border jumps automatically to the next available size.

To obtain the RESIZE sample code, download RESIZE.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for RESIZE.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download RESIZE.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the SOFTLIB\MSLFILES directory
  Get RESIZE.EXE

MORE INFORMATION

When a user clicks the resizing border of a window, Windows enters a PeekMessage loop to capture all the mouse messages that occur until the left mouse button is released. While inside this loop, every time the mouse moves it moves the rectangle that shows the new window size to provide a visual cue to the user as to what the new window size will be.

The RESIZE sample code modifies the resizing operation by entering it's own message loop to capture the mouse messages until the left button is released. Instead of updating the rectangle every time a mouse move is received, the RESIZE code checks to see if the current mouse position would make the window size one of the possible window width and height sizes as defined by the application. By doing this, the RESIZE application provides

more accurate visual cues about what the resizing operation will do.

The resizing operation is triggered by the WM_NCLBUTTONDOWN message both for Windows and the RESIZE application. When this message is received, a message loop is entered to filter out all the mouse messages except for two, WM_MOUSEMOVE and WM_LBUTTONUP. When the WM_MOUSEMOVE message is received, the RESIZE application checks to see if the current mouse position would make the window larger or smaller. If the window would be smaller, the resizing rectangle is moved to the next smaller dimension defined by the application. If the window would be larger, the program checks to see if the new size would be large enough for the next possible dimension and updates the rectangle accordingly. When the WM_LBUTTONUP message is received, the resizing operation is completed by updating the window size to the current position defined by the mouse and the rectangle is removed.

The RESIZE application also takes advantage of some of the flexibility provided by processing the WM_NCHITTEST message. Windows sends this message to an application with a mouse position and expects the application to describe which part of the window that mouse position covers. Frequently, applications pass this message on to DefWindowProc() and let the default calculations take care of telling the system what the mouse is on top of. The RESIZE application allows DefWindowProc() to process the message, but then checks to see if the mouse is over one of the resizing corners or in the client area. To simplify the resizing operation, RESIZE doesn't let the user resize from a window corner, so the application overrides the HTBOTTOMLEFT, HTBOTTOMRIGHT, HTTOPLEFT, and HTTOPRIGHT hit test codes and returns HTBOTTOM or HTTOP. By doing this, the mouse cursor accurately reflects the direction of the resize. When the HTCLIENT hit test code is returned, RESIZE changes this to HTCAPTION to allow the window to be moved even though it doesn't have a title bar.

Although this technique will work in Windows 95, it is not necessary. Windows 95 provides a new message WM_SIZING that will enable the program to do exactly the same thing without processing the WM_NCxxx messages or entering a PeekMessage() loop.

Additional reference words: 3.10 3.50 4.00
KBCategory: kbprg kbui kbcode kbfile
KBSubcategory: UsrWndw

SAMPLE: SCLBLDLG - Demonstrates Scaleable Controls in Dialog

PSS ID Number: Q112639
Authored 15-Mar-1994 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, 4.0

SUMMARY

In certain circumstances it is desirable to dynamically scale the controls in a dialog box to the size of the dialog box window. SCLBLDLG.EXE is a file in the Microsoft Software Library that contains sample code implementing scaleable controls in a dialog box.

SCLBLDLG can be downloaded as a self-extracting file from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL and download SCLBLDLG.EXE

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download SCLBLDLG.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get SCLBLDLG.EXE

MORE INFORMATION

To accomplish scaleable controls in a dialog box, the following messages are processed:

WM_INITDIALOG - To store original dimensions of the dialog box and all its controls, together with the font the dialog box uses. The original dimensions are stored using SetProp() in this sample. The font handle is stored in a static variable to be used with in WM_SIZE.

WM_SIZE - To calculate the scaling factor, and then scale up or down the font and all the controls in the dialog box.

WM_GETMINMAXINFO - To set the minimum size of the dialog box so the controls are not truncated.

WM_COMMAND - To clean up when closing the dialog box. RemoveProp() is called to remove the stored dimensions from the property list of the dialog box window and all its child control windows.

NOTE: Special processing is required for calculating the dimensions of CBS_DROPDOWN and CBS_DROPDOWNLIST style combo boxes. GetWindowRect() returns the dimensions of the edit portion of the combo box, excluding the drop-down list. To get the correct height for such combo boxes, the value returned by CB_GETDROPPEDCONTROLRECT is used instead of GetWindowRect().

Additional reference words: 3.10 3.50 3.51 4.00 95 proportional sizing softlib
KBCategory: kbprg kbfile
KBSubcategory: UsrDlgs

SAMPLE: ServerEnumDialog DLL

PSS ID Number: Q118327
Authored 14-Jul-1994 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

SUMMARY

SERVENUM is a dynamic-link library (DLL) that implements a multithreaded dialog box, allowing a user to browse the network for servers. The dialog box is instantiated by calling the ServerEnumDialog application programming interface (API) implemented in the SERVENUM.DLL.

Download SRVENM.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL and download SRVENM.EXE

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download SRVENM.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \softlib\mslfiles directory
  Get SRVENM.EXE

MORE INFORMATION

The Win32 application programming interface (API) implements several WNet dialog boxes that allow users to connect to and browse network resources. However, an API allowing a user to browse the network for servers is currently not implemented.

Because such a dialog box is often useful in network programs, the SERVENUM sample implements such a dialog box within a DLL. The sample also includes a simple Windows program that calls the ServerEnumDialog API implemented in SERVENUM.DLL of the sample. For redistribution issues of the DLL, please see the README.TXT file included with the sample.

This sample demonstrates the following Win32 programming issues:

DLLs
thread local storage
multithreading and thread synchronization
owner-draw list boxes
Unicode

WNet APIs

API Interface

```
ServerEnumDialog(
   HWND hwnd,              // Handle of calling window
   LPWSTR lpszServer,      // Buffer to store chosen server name
   LPDWORD lpcchServer,    // Pointer to size of buffer in bytes
   FARPROC lpfnHelpProc    // User-defined help procedure
   )
```

Purpose:

Display a dialog box that allows a user to select a computer on the network. The user may also type in a string. The result is not guaranteed to be a valid computer name.

Parameters:

hwnd - Handle of calling window

lpszServer - Buffer to store resultant server name in. This should be of length MAX_COMPUTERNAME_LENGTH+3

lpcchServer - Pointer to size of lpszServer, including the NULL terminating character. The resultant length of the string is stored here. If ERROR_MORE_DATA is return, the value is the required size of the buffer.

lpfnHelpProc - Pointer to user-defined help procedure. The procedure should be prototyped as int HelpProc(VOID);

Return Value:

ERROR_SUCCESS - User selected or typed in a server name
ERROR_CANCELLED - Dialog box was canceled
ERROR_NOT_ENOUGH_MEMORY - Unable to initialize dialog box
ERROR_MORE_DATA - lpszServer not large enough

Additional reference words: 3.10 WNetServerBrowseDialog softlib SRVENM.EXE
KBCategory: kbprg kbfile
KBSubcategory: NtwkWinnet

SAMPLE: Setting Tab Stops in a Windows List Box

PSS ID Number: Q66652
Authored 01-Nov-1990 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

Tab stops can be used in a list box to align columns of information. This article describes how to set tab stops in a list box and provides a code example that demonstrates the process.

MORE INFORMATION

To set tab stops in a list box, perform the following three steps:

1. Specify the LBS_USETABSTOPS style when creating the list box.

2. Assign the desired tab stops to an integer array.

   a. The tab stop values must be in increasing order -- back tab stops are not allowed. The tabs work the same as typewriter tabs: once a tab stop is overrun, a tab character will move the cursor to the next tab stop. If the tab stop list is overrun (that is, the current position is greater than the last tab stop value), the default tab of eight characters is used.

   b. The tab stops should be specified in dialog units. On the average, each character is about four horizontal dialog units in width.

   c. It is possible to hide columns of text from the user by specifying tab stops beyond the right side of the list box. This can be a useful way to hide information used for the application's internal processing.

3. Send an LB_SETTABSTOPS message to the list box to set the tab stops. For example, in Windows 3.1:

```
SendMessage(GetDlgItem(hDlg, IDD_LISTBOX),
            LB_SETTABSTOPS,
            TOTAL_TABS,
            (LONG)(LPSTR)TabStopList);
```

   a. If wParam is set to 0 (zero) and lParam to NULL, the tab stops are set to two dialog units by default.

   b. SendMessage() will return TRUE if all of the tab stops are set
      successfully; otherwise, SendMessage() returns FALSE.

Example

Below is an example of the process. Tab stops are set at character
positions 16, 32, 58, and 84.

```
   int     TabStopList[TOTAL_TABS]; /*  Array to store tabs */

   TabStopList[0] = 16 * 4;          /* 16 spaces */
   TabStopList[1] = 32 * 4;          /* 32 spaces */
   TabStopList[2] = 58 * 4;          /* 58 spaces */
   TabStopList[3] = 84 * 4;          /* 84 spaces */

   SendMessage(GetDlgItem(hDlg, IDD_LISTBOX),
               LB_SETTABSTOPS,
               TOTAL_TABS,
               (LONG)(LPSTR)TabStopList);
```

NOTE: For Win32, use LPARAM instead of LONG.

If the desired unit of measure is character position, then specifying
tab positions in dialog units is recommended. Dialog units are
independent of the current font; they are loosely based on the average
width of the system font. Each character takes approximately four
dialog units.

NOTE: Under Windows 95, dialog base units for dialogs based on non-system
fonts are calculated in a different way than under Windows 3.1. For more
information, please see the following article in the Microsoft Knowledge
Base:

```
   ARTICLE-ID: Q125681
   TITLE     : How to Calculate Dialog Base Units with Non-system-based
               Font
```

For more control over the exact placement of a tab stop, the desired
position should be converted to a pixel offset and this offset should
be converted into dialog units. The following formula will take a
pixel position and convert it into the first tab stop position before
(or at) the desired pixel position:

```
   TabStopList[n] = 4 * DesiredPixelPosition /
                    LOWORD(GetDialogBaseUnits());
```

There is a sample application named TABSTOPS in the Microsoft Software
Library that demonstrates how tab stops are set and used in a list
box.

Download TABSTOPS.EXE, a self-extracting file, from the Microsoft Software
Library (MSL) on the following services:

 - CompuServe

GO MSL
Search for TABSTOPS.EXE
Display results and download

- Microsoft Download Service (MSDL)
Dial (206) 936-6735 to connect to MSDL
Download TABSTOPS.EXE

- Internet (anonymous FTP)
ftp ftp.microsoft.com
Change to the \SOFTLIB\MSLFILES directory
Get TABSTOPS.EXE

Additional reference words: 3.00 3.10 3.50 4.00 softlib
KBCategory: kbprg kbfile
KBSubcategory: UsrCtl

SAMPLE: Simulating Palette Animation on Non-Palette Displays

PSS ID Number: Q130476
Authored 21-May-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, version 3.5

SUMMARY

LAVALAMP is a sample application in the Microsoft Software Library that demonstrates how to simulate the effects of the AnimatePalette() function on devices that may not support palettes. This program also demonstrates how to create and manipulate dibsections. The following dibsection functions are used in LAVALAMP:

```
CreateDIBSection()
GetDIBColorTable()
SetDIBColorTable()
```

Download LAVALAMP.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for LAVALAMP.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download LAVALAMP.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get LAVALAMP.EXE

MORE INFORMATION

When running in display modes that are not palette-based, many of the effects that can be performed easily with palette animation need to be reprogrammed. A simple method of simulating palette animation can be achieved by "animating" a device-independent bitmap's (DIBs) color table and redisplaying the DIB with the new colors. To demonstrate this technique, LAVALAMP creates an 8-bits-per-pixel (bpp) dibsection. Then it shifts each of the RGBQUAD data structures in the color table by one position to the left, and recycles the first entry in the color table to the last position. After each modification to the color table, the DIB is redisplayed.

Because the entire DIB must be redisplayed after each modification to the color table, this technique is not recommended for large bitmaps.

Additional reference words: 3.50 technote BMP softlib
KBCategory: kbprg kbfile
KBSubcategory: GdiBmp GdiPal

SAMPLE: Using Blinking Text in an Application

PSS ID Number: Q11787
Authored 17-Dec-1987 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

It is possible to create blinking text in a Windows-based application. Because there are no character attributes similar to the normal MS-DOS text environment, the application must repeatedly paint the screen to implement the flash. This article, and an accompanying file in the Microsoft Software Library, demonstrate how this is done.

MORE INFORMATION

A timer is used to determine the rate at which the text flashes. Timer messages are processed by inverting the appropriate area in the window using the DSTINVERT action of the PatBlt function. The second time that the PatBlt function is called, the text returns to its original state. Alternatively, the PATINVERT action of the PatBlt function may be used. To use this method, an appropriate brush must be selected into the display context as the current pattern. This method requires more work, however, it is more flexible.

The rate at which the text blinks can be set to match the cursor blink time set in the Control Panel. To do this, the following code should be run when the application starts and in response to WM_WININICHANGE messages:

```
   nRate = GetProfileInt(
     (LPSTR)"windows",         /* heading in [] */
     (LPSTR)"CursorBlinkRate", /* string to match */
      550);                    /* default value */
```

Be sure to delete the timer when the application terminates.

There is a sample program in the Microsoft Software Library named BLINK that uses this technique to demonstrate blinking text.

Download BLINK.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for BLINK.EXE
  Display results and download

- Microsoft Download Service (MSDL)
    Dial (206) 936-6735 to connect to MSDL
    Download BLINK.EXE

- Internet (anonymous FTP)
    ftp ftp.microsoft.com
    Change to the \SOFTLIB\MSLFILES directory
    Get BLINK.EXE

Additional reference words: 3.00 3.10 3.50 4.00 95 softlib BLINK.EXE
KBCategory: kbprg kbfile
KBSubcategory: UsrPnt

SAMPLE: Win16 App (WOW & Win32s) Calling Win32 DLL Code

PSS ID Number: Q114341
Authored 01-May-1994 Last modified 10-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5

SUMMARY

The INTEROP sample demonstrates two general methods for calling routines in a Win32 DLL from a Windows-based application: thunks and SendMessage(). There are two different thunking methods, one for each platform: Generic Thunks on Windows NT and Universal Thunks on Win32s. The message used is WM_COPYDATA, a new message introduced by Windows NT and Win32s. All three methods provide a way to call functions and pass data across the 16-32 boundary, translating any pointers in the process. The advantages of WM_COPYDATA over thunks are that it is fast the exact same code runs on either platform (the thunk used will depend on the platform). The disadvantage of WM_COPYDATA is that a method must be devised to get a function return value (other than true or false) back to the calling application.

NOTE: Universal Thunks were designed to work with a Win32-based application calling a 16-bit DLL. The method described here has limitations. Because the application is 16-bit, no 32-bit context is created, so certain calls will not work from the Win32 DLL.

The sample consists of the following source files:

APP16.C	-	Win16 Application
DLL16.C	-	16-bit side of Universal Thunk/Generic Thunk
STUB32.C	-	32-bit stub that loads the 32-bit DLLs on Win32s
UTDLL32.C	-	32-bit side of the Universal Thunk
DISP32.C	-	Dispatch calls sent through WM_COPYDATA
DLL32.C	-	Win32 DLL

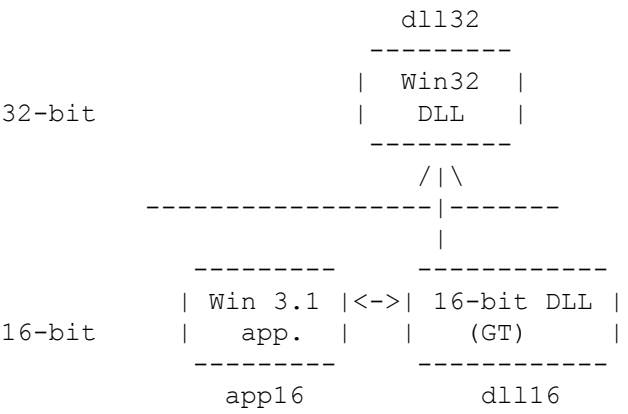
This sample is included with the Microsoft Win32 SDK. It is located in SCT\SAMPLES\INTEROP. NOTE: There is also an RPC sample named INTEROP, but it is in a different directory.

MORE INFORMATION

Generic Thunk

Under Windows NT, it is possible to call routines in a Win32 DLL from a Windows-based application using an interface called Generic Thunks. The SDK file DOC\SDK\MISC\GENTHUNK.TXT describes the interface.

Here is a picture of the way the pieces fit together in INTEROP:



DLL16 is loaded when APP16 is loaded. If it detects that WOW is present, then it loads DLL32.

WOW presents a few new 16-bit application programming interfaces (APIs) that allow you to load the Win32 DLL, get the address of the DLL routine, call the routine (passing it up to thirty-two 32-bit arguments), convert 16:16 (WOW) addresses to 0:32 addresses (useful if you need to build up a 32-bit structure that contains pointers and pass a pointer to it), and free the Win32 DLL. These functions are:

```
DWORD FAR PASCAL LoadLibraryEx32W( LPCSTR, DWORD, DWORD );
DWORD FAR PASCAL GetProcAddress32W( DWORD, LPCSTR );
DWORD FAR PASCAL CallProc32W( DWORD, ..., LPVOID, DWORD, DWORD );
DWORD FAR PASCAL GetVDMPointer32W( LPVOID, UINT );
BOOL FAR PASCAL FreeLibrary32W( DWORD );
```

When linking the Win16 application, you need to put the following statements in the .DEF file, indicating that the functions will be imported from the WOW kernel:

```
IMPORTS
   kernel.LoadLibraryEx32W
   kernel.FreeLibrary32W
   kernel.GetProcAddress32W
   kernel.GetVDMPointer32W
   kernel.CallProc32W
```

Note that although these functions are called in 16-bit code, they need to be provided with 32-bit handles, and they return 32-bit handles.

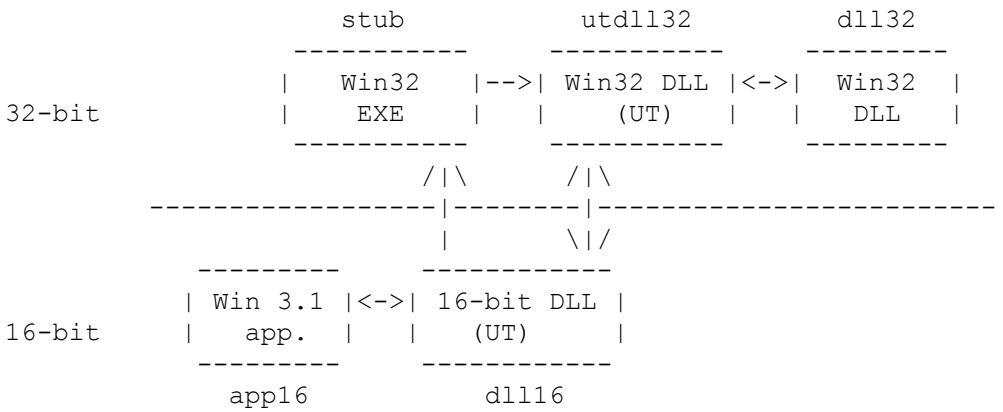
In addition, be sure that your Win32 DLL entry points are declared with the _stdcall convention; otherwise, you will get an access violation.

Universal Thunk

Under Win32s, it is possible to call routines in a Win32 DLL from a Win16 application using an interface called Universal Thunks. The interface is

described in the Win32s Programmer's Reference. The sample UTSAMPLE, shows the opposite (and more typical) case, a Win32 application calling 16-bit routines.

Here is a picture of the way the pieces fit together in INTEROP:



The load order is as follows: The Windows 3.1 application loads the 16-bit DLL. The 16-bit DLL checks to see whether the 32-bit side has been initialized. If it has not been initialized, then the DLL spawns the 32-bit .EXE (stub), which then loads the 32-bit DLL that sets up the Universal Thunks with the 16-bit DLL. Once all of the components are loaded and initialized, when the Windows 3.x application calls an entry point in the 16-bit DLL, the 16-bit DLL uses the 32-bit Universal Thunk callback to pass the data over to the 32-bit side. Once the call has been received on the 32-bit side, the proper Win32 DLL entry point can be called.

WM_COPYDATA

The wParam and lParam for this message are as follows:

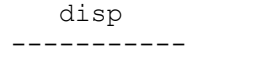
```
wParam = (WPARAM) (HWND) hwndFrom;   /* handle of sending window */
lParam = (LPARAM) (PCOPYDATASTRUCT) pcds;
```

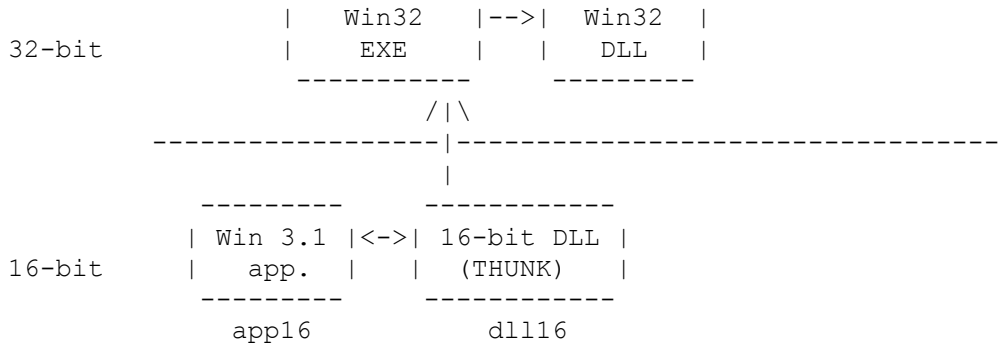
Where hwndFrom is the handle of the sending window and COPYDATASTRUCT is defined as follows:

```
typedef struct tagCOPYDATASTRUCT {
    DWORD dwData;
    DWORD cbData;
    PVOID lpData;
} COPYDATASTRUCT;
```

The INTEROP sample uses dwData as a function code, indicating which Win32 DLL entry point should be calling and lpData to contain a pointer to the data structure to be passed to the function.

Here is a picture of the way the pieces fit together in INTEROP:





DLL16 is loaded when APP16 is loaded. DISP is spawned to handle WM_COPYDATA messages, regardless of platform. DISP dispatches the calls to DLL32, marshalling the arguments.

Additional reference words: 3.50
KBCategory: kbref
KBSubcategory: SubSys

Secure Erasure Under Windows NT

PSS ID Number: Q94239
Authored 30-Dec-1992 Last modified 03-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1, 3.5, and 3.51

SUMMARY

File systems under Windows NT currently have virtual secure erasure (when a file is deleted, the data is no longer accessible through the operating system). Although the bits could still be on disk, Windows NT will not allow access to them.

MORE INFORMATION

The NTFS file system does this by keeping a high-water mark, for each file, of bytes written to the file. Everything below the line is real data, anything above the line is (on disk) random garbage that used to be free space, but any attempt to read past this high-water mark returns all zeros.

Other reusable objects are also protected. For example, all the memory pages in a process's address space are zeroed when they are touched (unlike the file system, a process may directly access its pages, and thus the pages must be actually zeroed rather than virtually zeroed).

Note that file system security assumes physical security; in other words, if a person has physical access to a machine and can boot an alternative operating system and/or add custom device drivers and programs, he/she can always get direct access to the bits on disk.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

Security and Screen Savers

PSS ID Number: Q96780
Authored 25-Mar-1993 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, and 3.51

SUMMARY

Screen savers are user-mode applications that execute in a different desktop (see the note at the end of the article). Therefore, a screen saver cannot enumerate windows of user-mode applications. This design prevents unauthorized users from viewing the contents of applications displayed on the screen. For secure screen savers (those that ask for a password), this adds a further layer of protection.

Screen savers also execute in the security context of the logged-on user. A screen saver may call ExitWindowsEx(), to log off from or shut down the system, or any other application programming interface (API) that the logged-on user has permission to perform.

MORE INFORMATION

A sample screen saver SCRNSAVE is distributed on the Win32 SDK CD.

NOTE: A desktop is a virtual screen. Windows NT 3.5 and earlier have three desktops--the main desktop, the WinLogon desktop, and a desktop for screen savers. Windows NT 3.51 and later support and document creating multiple desktops.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

Security Attributes on Named Pipes

PSS ID Number: Q102798
Authored 10-Aug-1993 Last modified 21-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

The March beta release of the Windows NT (and earlier) did not require security attributes on pipes. It was valid at that time to enter NULL for the last parameter of the Win32-based application programming interface (API) CreateNamedPipe(). This is no longer the case.

MORE INFORMATION

Windows NT 3.1 and later require security attributes for pipes. Please note that setting the security attributes parameter to NULL does not indicate that you want a NULL security descriptor (SD), rather it indicates that you want to inherit the security descriptor of the current access token. This generally means that any client wanting to connect to your pipe server must have the same security attributes as the user that started the server. For example, if the user who started the server was the administrator of the machine, then any client who wants to connect must also be an administrator to that machine.

Below is an code sample that demonstrates creating a named pipe with a NULL security descriptor.

```
HANDLE               hPipe;    // Pipe handle.
SECURITY_ATTRIBUTES  sa;       // Security attributes.
PSECURITY_DESCRIPTOR pSD;      // Pointer to SD.

// Allocate memory for the security descriptor.

pSD = (PSECURITY_DESCRIPTOR) LocalAlloc(LPTR,
                             SECURITY_DESCRIPTOR_MIN_LENGTH);

// Initialize the new security descriptor.

InitializeSecurityDescriptor(pSD, SECURITY_DESCRIPTOR_REVISION);

// Add a NULL descriptor ACL to the security descriptor.

SetSecurityDescriptorDacl(pSD, TRUE, (PACL) NULL, FALSE);

sa.nLength = sizeof(sa);
sa.lpSecurityDescriptor = pSD;
sa.bInheritHandle = TRUE;
```

```
   // Create a local named pipe with a NULL security descriptor.

   hPipe = CreateNamedPipe(
         "\\\\.\\PIPE\\test",    // Pipe name = 'test'.
         PIPE_ACCESS_DUPLEX      // 2-way pipe.
         | FILE_FLAG_OVERLAPPED, // Use overlapped structure.
         PIPE_WAIT               // Wait on messages.
         | PIPE_READMODE_MESSAGE // Specify message mode pipe.
         | PIPE_TYPE_MESSAGE,
         MAX_PIPE_INSTANCES,     // Maximum instance limit.
         OUT_BUF_SIZE,           // Buffer sizes.
         IN_BUF_SIZE,
         TIME_OUT,               // Specify time out.
         &sa);                   // Security attributes.
```

It is important to note that by specifying TRUE for the fDaclPresent parameter and NULL for pAcl parameter of the SetSecurityDescriptorDacl() API, a NULL access control list (ACL) is being explicitly specified.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseIpc

Security Context of Child Processes

PSS ID Number: Q111545
Authored 14-Feb-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

When a process is created with the CreateProcess() API, it operates under the same security context as the parent. The child process inherits the parent's security context, which are defined by the parent's access token. Even if a thread in the parent process impersonates a client and then creates a new process, the new process still runs under the parent's original security context and not the under the impersonation token.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

Semicolons Cannot Separate Macros in .HPJ File

PSS ID Number: Q83020
Authored 02-Apr-1992 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The author of a Help system can combine individual macro commands into a macro string, which the Microsoft Help Compiler processes as a unit. When the macro string is part of an RTF text file, the individual macros in the string are separated from each other with a semicolon character (;). The Help system runs the individual macros of a macro string sequentially.

The author can define a macro string that is run when the user loads a Help file. This macro string is placed into the [CONFIG] section of the Help project (.HPJ) file. However, in the .HPJ file, the individual macros of the string are separated from each other with a colon character (:) because the semicolon character indicates the beginning of a comment.

MORE INFORMATION

In the following sample [CONFIG] section, a macro adds two buttons to the Help window's button bar. The first button is labeled "Other", which when chosen brings up the About Help dialog box. The second button is labeled "Test". When chosen, it disables the "Other" button and jumps to the topic represented by "context_string." To create the "Test" button, two macros are concatenated to form the macro parameter in the CreateButton call.

```
[CONFIG]
; This first button is added so that the demonstration macro is
; complete. This macro just creates a button. Choosing the button
; brings up the About Help dialog box.

CreateButton("other_button","&Other","About()")

; This macro also creates a button. Choosing the button disables
; "other_button", created above, and jumps to the topic represented by
; "topic_string."
;
; Note that the two macros in the CreateButton macro are separated by
; a colon, not a semicolon.
;
; NOTE: The following macro should appear on a single line.
```

```
CreateButton("test_button","&Test","DisableButton(`Other_Button'):
JumpId(`testhelp.hlp',`context_string')")

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsHlp
```

SendMessage() in a Multithreaded Environment

PSS ID Number: Q95000
Authored 28-Jan-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

When thread X calls SendMessage() to send a message to a window created by thread Y, it must wait until thread Y calls PeekMessage(), GetMessage(), or WaitMessage() before the SendMessage() call can continue. This process prevents synchronization problems.

MORE INFORMATION

Because of the multithreaded environment of Windows NT, SendMessage() does not behave in the same manner as it does under Windows 3.1. Under Windows 3.1, SendMessage() simply calls the window procedure given to it. In Windows 3.1, if you use the SendMessage() function to send a message to a window of another task, the system will perform a task switch to the target window (change to the stack of the target Windows-based application), let the target window process the message [when it calls GetMessage() or PeekMessage()], and then switch back to the original message.

Under Windows NT, however, only the thread that created a window may process the window's messages. Therefore, if thread X sends a message [via SendMessage()] to a window that was created by thread Y, thread X must wait for thread Y to be in a receiving state, and handle the message for it.

Thread Y is only in a receiving state when it calls PeekMessage(), GetMessage(), or WaitMessage(), because synchronization problems may occur if a thread is interrupted while processing other messages. While in a receiving state, thread Y may process messages sent to its windows via SendMessage() (in this case, by thread X).

Note that PeekMessage() and GetMessage() look in thread Y's message queue for messages. Because SendMessage() does not post any messages, PeekMessage() and GetMessage() will not see any indication of the SendMessage() call. The two functions merely serve as a point in time at which SendMessage() (thread X) may "interrupt," and have thread Y process its message next. Then PeekMessage() or GetMessage() continues normal operation under thread Y.

Because of this behavior, if thread X sends a message to thread Y, and thread Y is locked in a tight loop, thread X is now locked as well. This may be prevented by using SendNotifyMessage(), which behaves as SendMessage() does above, but returns immediately. This may be an advantage if it is not important that the sent message be completed before thread Y

continues. Note, however, that because SendNotifyMessage() is asynchronous, thread X should not pass pointers to any of its local variables when making the call, because they may be gone by the time thread Y attempts to look at them. This would result in a general protection violation (GP fault) when thread Y accesses the pointer.

Additional reference words: 3.10 3.50 3.51 4.00 95 GP-fault
KBCategory: kbprg
KBSubcategory: UsrMisc

Services and Redirected Drives

PSS ID Number: Q115848
Authored 06-Jun-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

When writing a service, do not use WNetAddConnection() to redirect drives to remote shares, because the redirector does not allow the logged-on user to see the server or enumerate the drive. Instead, use universal naming convention (UNC) names whenever possible to redirect drives to remote shares.

MORE INFORMATION

WNetAddConnection() creates a global symbolic link describing the drive. Therefore, the user can change to the drive at the command prompt. However, WNetOpenEnum(), WNetEnumResource, and the "net use" command fail to list the drive connection that was created by the service. This happens because the logged-on user does not have an active session to the remote connection, so the redirector will not let the user see the remote server.

The File Manager calls GetDriveType() on each drive and puts up an icon for each drive found. The File Manager creates an icon for redirected drives created from a service because there is a global symbolic link to that drive. However, the share name is not available. In addition, the logged-on user cannot use File Manager to disconnect the drive because the drive was created by the service, not the logged-on user.

If the service process is running in the Local System account, then only that process or another process running in the Local System account can call WNetCancelConnection() to disconnect the drive.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseService

SetActiveWindow() and SetForegroundWindow() Clarification

PSS ID Number: Q97925
Authored 25-Apr-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

By default, each thread has an independent input state (its own active window, its own focus window, and so forth). SetActiveWindow() always logically sets a thread's active window state. To force a window to the foreground, however, use SetForegroundWindow(). SetForegroundWindow() activates a window and forces the window into the foreground. SetActiveWindow() always activates, but it brings the active window into the foreground only if the thread is the foreground thread.

NOTE: If the target window was not created by the calling thread, the active window status of the calling thread is set to NULL, and the active window status of the thread that created the target window is set to the target window.

Applications can call AttachThreadInput() to allow a set of threads to share the same input state. By sharing input state, the threads share their concept of the active window. By doing this, one thread can always activate another thread's window. This function is also useful for sharing focus state, mouse capture state, keyboard state, and window Z-order state among windows created by different threads whose input state is shared.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrWndw

SetBkColor() Does Not Support Dithered Colors

PSS ID Number: Q69885
Authored 06-Mar-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The syntax for the SetBkColor function is documented in the Microsoft Windows Software Development Kit (SDK) as follows:

```
DWORD SetBkColor(HDC hDC, COLORREF crColor);
```

SetBkColor sets the current background color of the specified device context (DC) to the color that the crColor parameter references, or to the nearest physical color if the device cannot represent the RGB color value that the crColor parameter specifies. In other words, SetBkColor cannot be used to set the background to a dithered color and defaults to the physical color that is closest to the requested crColor value.

MORE INFORMATION

This behavior can cause unexpected results for an application that changes the background color of a control to a color that cannot be represented by a color provided by the display device.

Specifically, when an application specifies a dithered color for the background of an edit control, and specifies the same color for the text background, Windows paints the control in two distinct colors.

For example, using the standard VGA display driver, the following call, in which COLOR_INACTIVEBORDER is a green/gray specified by RGB(64, 128, 128), sets the background color to gray (RGB(128, 128, 128)) rather than the dithered green/gray that is desired:

```
SetBkColor(wParam, GetSysColor(COLOR_INACTIVEBORDER));
```

To illustrate, if the application uses the function call while processing the WM_CTLCOLOR message to change the color of an edit control, the window background is painted green/gray, and the text background defaults to the nearest physical color, which is gray. This produces a gray rectangle inside a green/gray rectangle rather than the desired green/gray for the entire edit control.

This behavior can also occur with other controls such as option buttons and list boxes. However, an application can avoid this problem

by using the SetBkMode function to set the background mode to TRANSPARENT. This allows the dithered brush pattern to show through beneath the text to achieve the desired results. That solution is not practical with a multiline edit control because if text is inserted, and the background mode has been set to TRANSPARENT, the text that is pushed to the right by the inserted text leaves its image behind. The result is text superimposed on top of other text, which quickly becomes unreadable.

To partially work around this situation for a multiline edit control, use the GetNearestColor function to determine the nearest physical color to the desired color, as in the code fragment below. In this case, the entire edit control is gray:

```
   case WM_CREATE:
      {
      HDC hDC;
      hDC = GetDC(hWnd);
      hGrayBrush = CreateSolidBrush(GetNearestColor(hDC,
            RGB(64, 128, 128)));
      ReleaseDC(hWnd, hDC);
      hWndEdit = CreateWindow( ... ES_MULTILINE ... );
      }
      break;

   case WM_CTLCOLOR:
      if (HIWORD(lParam) == CTLCOLOR_EDIT)
         {
         // The following call creates the nearest physical
         // color; therefore, it will be the same as the
         // hGrayBrush created above.
         SetBkColor(wParam, RGB(64, 128, 128));
         SetTextColor(wParam, RGB(255, 0, 0)); // red text
         return (DWORD)hGrayBrush;
         }
      else
        return DefWindowProc(hWnd, identifier, wParam, lParam);
      break;
```

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiDrw

SetClipboardData() and CF_PRIVATEFIRST

PSS ID Number: Q24252
Authored 16-Dec-1987 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The documentation for SetClipboardData() states that CF_PRIVATEFIRST can be used to put private data formats on the clipboard. It also states that data of this format is not automatically deleted. However, that is apparently not true. That is, the data is removed automatically when the clipboard is emptied (sending a WM_DESTROYCLIPBOARD message) and the new item is set into the clipboard. The old item is shown; however, the handle now is invalid because GlobalFree() is called on it.

MORE INFORMATION

GlobalFree() was not called on this handle. If you try to use the other handle to this memory, you will find that the one you initially received from GlobalAlloc() is still valid. Only the clipboard handle has been invalidated by the call to EmptyClipboard().

The documentation states that "Data handles associated (with CF_PRIVATEFIRST) will not be freed automatically." This statement refers to the memory associated with that data handle. When SetClipboardData() is called under standard data types, it frees the block of memory identified by hMem. This is not the case for CF_PRIVATEFIRST. Applications that post CF_PRIVATEFIRST items on the clipboard are responsible for the memory block containing those items.

This is not intended to imply that items placed on the clipboard will remain on the clipboard if they are CF_PRIVATEFIRST. When a call is made to EmptyClipboard(), all objects will be removed.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrClp

SetErrorMode() Is Inherited

PSS ID Number: Q105304
Authored 17-Oct-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

An application can use SetErrorMode() to control whether the operating system handles serious errors or whether the application itself will handle the errors.

NOTE: The error mode will be inherited by any child process. However, the child process may not be prepared to handle the error return codes. As a result, the application may die during a critical error without the usual error message popups occurring.

This behavior is by design.

One solution is to call SetErrorMode() before and after the call to CreateProcess() in order to control the error mode that is passed to the child. Be aware that this process must be synchronized in a multithreaded application.

There is another solution available in Windows NT 3.5 and later. CreateProcess() has a new flag CREATE_DEFAULT_ERROR_MODE that can be used to control the error mode of the child process.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseExcept

SetParent and Control Notifications

PSS ID Number: Q104069
Authored 05-Sep-1993 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

An edit, list box, or combo box control sends notifications to the original parent window even after SetParent has been used to change the control's parent. A button control sends notifications to the new parent after SetParent has been used to change its parent.

Edit, list box, and combo box controls keep a private copy of the window handle of the parent at the time of creation. This handle is not changed when SetParent is used to change the control's parent. Consequently, the notifications (EN_*, LBN_*, and CBN_* notifications) go to the original parent.

Note that WM_PARENTNOTIFY messages go to the new parent and GetParent() returns the new parent. If it is required that notifications go to the new parent window, code must be added to the old parent's window procedure to pass on the notifications to the new parent.

For example:

```
case WM_COMMAND:
  hwndCtl = LOWORD(lParam);

  // If notification is from a control and the control is no longer this
  //   window's child, pass it on to the new parent.
  if (hwndCtl && !IsChild(hWnd, hwndCtl))
      SendMessage(GetParent(hwndCtl), WM_COMMAND, wParam,  lParam);
  else Do normal processing;
```

Button controls send notifications to the new parent after SetParent has been used to change the parent.

Additional reference words: 3.10 3.50 3.51 4.00 95 listbox combobox
KBCategory: kbprg
KBSubcategory: UsrCtl

SetTimer() Should Not Be Used in Console Applications

PSS ID Number: Q102482
Authored 03-Aug-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

SetTimer() was not designed to be used with a console application because it requires a message loop to dispatch the timer signal to the timer procedure. In a console application, this behavior can be easily emulated with a thread that is set to wait on an event.

MORE INFORMATION

In Windows NT 3.1, SetTimer() can work within a console application, but it requires a thread in a loop calling GetMessage() and DispatchMessage().

For example,

```
while (1)
{
   GetMessage();
   DispatchMessage();
}
```

Because this requires a thread looping, there is no real advantage to adding a timer to a console application over using a thread waiting on an event.

Another option is to use a multimedia timer, which does not require a message loop and has a higher resolution. In Windows NT 3.5, the resolution can be set to 1 msec using timeBeginPeriod(). See the help for timeSetEvent() and the Multimedia overview. Any application using Multimedia calls must include MMSYSTEM.H, and must link with WINMM.LIB.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMisc

Setting Dynamic Breakpoints in WinDbg

PSS ID Number: Q100642
Authored 24-Jun-1993 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The WinDbg breakpoint command contains a metacircular interpreter; that is, you can execute commands dynamically once a breakpoint is hit. This allows you to perform complex operations, including breaking when an automatic variable has changed, as described below.

The command interpreter of WinDbg allows any valid C expression to serve as a break condition. For example, to break whenever a static variable has changed, use the following expression in the Expression field of the breakpoint dialog box:

```
&<variablename>
```

In addition, the length should be specified as 4 (the size of a DWORD) in the length field.

This technique does not work for automatic variables because the address of an automatic variable may change depending on the value that the stack pointer has upon entering the function that defines the automatic variable. This is one case where the breakpoint needs to be redefined dynamically.

For this purpose, a breakpoint can be enabled at function start and disabled at function exit, so that the address of the variable is recomputed.

MORE INFORMATION

Suppose that the name of the function is "subroutine" and the local variable name is "i". The following steps will be used:

1. Start the program and step into the function that defines the automatic variable with the commands:

   ```
   g subroutine
   p
   bp500 ={subroutine}&i /r4 /C"?i"
   ```

   The breakpoint number is chosen to be large so that the breakpoint will be well out of range of other breakpoints. Note that /r4 indicates a length of 4 because i is an integer. Make this number larger for other data types. The command "?i" prints out the value

of i.

2. Next, disable this first breakpoint with the command

   bd500

   because the address of i may change. The breakpoint will be enabled when in the scope of function subroutine.

3. The second breakpoint definition is set at the entry point of the function:

   bp .<FirstLine> /C"be 500;g"

   This is where the breakpoint is enabled. Note that <FirstLine> is the line number of the first statement in the function subroutine.

4. The last breakpoint is set at the end of the function

   bp .<LastLine> /C"bd 500;g"

   and will disable the breakpoint again. Note that <LastLine> is the line number of the last statement in the function subroutine.

   Note that if the function has more than one exit point, multiple breakpoints may have to be defined.

Program execution stops when breakpoint #500 is hit (for example, the value of i changes), but execution will continue after the other two breakpoints because they contain go ("g") commands.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsWindbg

Setting File Permissions

PSS ID Number: Q98952
Authored 18-May-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

In Windows NT, local access controls can be set on just NTFS partitions, not FAT or HPFS partitions or floppies. Read/execute-only permissions should work properly on a CD-ROM.

The exception is that ACLs (access control lists) can be set on shares, regardless of the file system, to control access to all the files within that share. For example, you can give read access to everyone, but give full access just to members of a certain group or to certain individuals.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

Setting the Console Configuration

PSS ID Number: Q105674
Authored 22-Oct-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

To create a command prompt with custom features such as

```
Settings
Fonts
Screen Size and Position
Screen Colors
```

create a new entry in the Program Manager for CMD.EXE (suppose that the description is CUSTOM), choose these items from the CMD system menu, and select Save Configuration in each dialog box. The settings are saved in the registry under

```
HKEY_CURRENT_USER\
          Console\
          custom
```

and are used when starting the CUSTOM command prompt from the Program Manager or when specifying:

```
start "custom"
```

This behavior is really a convenient side effect of

```
start <string>
```

which sets the title in the window title bar. When you create a new console window with the START command, the system looks in the registry and tries to match the title with one of the configurations stored there. If it cannot find it, it defaults to the values stored in:

```
HKEY_CURRENT_USER\
          Console\
      Configuration
```

This functionality can be duplicated in your own applications using the registry application programming interface (API).

For more information, please see the "Registry and Initialization Files" overview and the REGISTRY sample.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseMisc

Setup Toolkit .INF File Format and Disk Labels

PSS ID Number: Q88141
Authored 18-Aug-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, version 3.5

In the Microsoft Setup Toolkit for Windows, the Source Media Descriptions section of the .INF file contains one line for each of the disks you use to install your application. Each of these lines consists of four quoted strings, separated by commas. The second quoted string is the disk label, which you create using the disk-layout utilities. This disk label has nothing to do with the MS-DOS disk label. The disk label in the .INF file comes from the Disk Labels command on the Options menu in DSKLAYT.EXE and is arbitrarily chosen by the developer during the disk-layout process.

The Setup Toolkit only uses this disk label to prompt the user for disks. The Setup Toolkit uses the tag filename in the Source Media Descriptions section to determine if the proper disk has been placed in the drive.

Additional reference words: 3.00 3.10 3.50 MSSetup tool kit
KBCategory: kbtool
KBSubcategory: TlsMss

SHARE.EXE Functionality Built into Windows NT

PSS ID Number: Q101191
Authored 07-Jul-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

The functionality of the MS-DOS SHARE.EXE utility is built into the Windows NT kernel. Any application or application programming interface (API) that relies on the SHARE.EXE functionality is automatically supported. This functionality cannot be disabled.

MORE INFORMATION

If you run the MS-DOS version of SHARE.EXE, you will receive a message stating that SHARE is already installed. The Windows NT MS-DOS emulation hooks Interrupt 2Fh function 10H and always returns a status indicating that SHARE is installed.

If you run an MS-DOS-based application and it complains that SHARE.EXE is not installed, the application may be searching the AUTOEXEC.BAT file for a "share" string rather than using the proper Interrupt 2Fh interface.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: SubSys

Sharing All Data in a DLL

PSS ID Number: Q109619
Authored 05-Jan-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

Win32 dynamic-link libraries (DLLs) use instance data by default. This means that each application that uses a DLL gets its own copy of the DLL's data. However, it is possible to share the DLL data among all applications that use the DLL.

If you only need to share some of the DLL data, we recommend creating a new section and sharing it instead. For additional information, please see the following article(s) in the Microsoft Knowledge Base:

ARTICLE-ID: Q89817
TITLE : How to Specify Shared and Nonshared Data in a DLL

If you want to share *all* of the DLL static data, it is important to do two things:

- First, the DLL must use the DLL version of the C Run-time (for example CRTDLL.LIB or MSVCRT.LIB). Please see your product documentation for more information about using the C Run-time in a DLL.

- Second, you will need to specify that both .data and .bss are shared. Often, this is done in the "SECTIONS" portion of the .DEF file. For example:

```
SECTIONS
   .bss  READ WRITE SHARED
   .data READ WRITE SHARED
```

If you are using Visual C++ 32-bit Edition, you will have to specify this using the -section switch on the linker. For example:

```
link -section:.data,rws -section:.bss,rws
```

Only static data will be shared. Memory allocated dynamically with calls to APIs/functions such as GlobalAlloc() or malloc() will still be specific to the calling process.

The system will try to load the shared memory block at the same address in each process. However, if the block cannot be loaded into the same memory address, the system allocates a new block of memory for that process and the memory is not shared. No run time warnings are given when this happens, therefore, named shared memory is generally a safer option.

MORE INFORMATION

The C Run-time uses global variables. If the CRT is statically linked to the DLL, these variables will be shared among all clients of the DLL and will most likely cause an exception c0000005.

The reason you need to specify both .data and .bss as shared is because they each hold different types of data. The .data section holds initialized data and the .bss section holds the uninitialized data.

One reason for sharing all data in a DLL is to have consistent behavior in the DLL between Win32 (running on Windows NT) and Win32s (running on Windows 3.1). When running on Win32s, a 32-bit DLL shares its data among all of the processes that use the DLL.

Note that it is not necessary to share all data to behave identically between Win32 and Win32s. The DLL can use thread local storage (TLS) on Win32s to store variables as instance data. For additional information, please see the following article(s) in the Microsoft Knowledge Base:

   ARTICLE-ID: Q109620
   TITLE   : Creating Instance Data in a Win32s DLL

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseDll

Sharing Memory Between 32-Bit and 16-Bit Code on Win32s

PSS ID Number: Q105762
Authored 24-Oct-1993 Last modified 23-Feb-1995

The information in this article applies to:

- Microsoft Win32s versions 1.0, 1.1, and 1.2

SUMMARY

This article discusses many of the issues involved in sharing memory across the process boundary under Win32s.

MORE INFORMATION

Memory allocated by a Win32-based application using GlobalAlloc() can be shared with a 16-bit Windows-based application on Win32s. If the memory is allocated with GMEM_MOVEABLE, then GlobalAlloc() returns a handle and not a pointer. The 16-bit Windows-based application can use the low word of this handle. The high word is all zeros. Make sure to lock the handle using GlobalLock() in the 16-bit Windows-based application to get a pointer.

NOTE: GlobalAlloc (GMEM_FIXED...) is not the same as GlobalFix(GlobalAlloc(GMEM_MOVEABLE...)). GMEM_FIXED will allocate locked pages, which is most often not what you want.

Memory allocated by a 16-bit Windows-based application via GlobalAlloc() must be fixed via GlobalFix() and translated before it can be passed to a Win32-based application. Whenever a Windows object is passed to a Win32-based application by its 32-bit address, the memory must be fixed, because the address is computed from the selector base only once. If Windows moves the memory, the linear address used by the Win32-based application will no longer be valid.

If you are using the Universal Thunk, you can also pass a buffer from a Win32-based application to a 16-bit dynamic-link library (DLL) in the UTRegister() call. The address is translated for you. Another alternative is the translation list passed to the callable stubs. Addresses passed in the translation list will be translated during the thunking process. For more information on the Universal Thunk, please see the "Win32 Programmer's Reference."

NOTE: The ability to share global memory handles under Win32s is a result of the implementation of Windows 3.1, in which all applications run in the same address space. This is not true of existing Win32 platforms and will not be true of future Win32 platforms.

Allocating memory with GlobalAlloc() gets you tiled selectors. However, you can only tile 255 selectors at a time and there is an overall limit of 8192 selectors in the system. If you allocate memory using new, malloc(), HeapAlloc(), LocalAlloc() or VirtualAlloc, your allocated memory does not

automatically get you tiled selectors. However, because you don't automatically get tiled selectors, whenever you pass memory to 16-bit code, selectors must be synthesized for you. There's currently a limit of 256 selectors that Win32s maintains for select synthesis. Also note that each block of memory that you pass is limited to 32K in size due to the way that Win32s tiles selectors.

Additional reference words: 1.00 1.10 1.20
KBCategory: kbprg
KBSubcategory: W32s

Sharing Objects with a Service

PSS ID Number: Q106387
Authored 07-Nov-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

To share objects (file mapping, synchronization, and so forth) created by a service, you must place a null DACL (discretionary access-control list) in the security descriptor field when the object is created. This grants everyone access to the object.

MORE INFORMATION

This null DACL is not the same as a NULL, which is used to specify the default security descriptor. For example, the following code can be used to create a mutex with a null DACL:

```
PSECURITY_DESCRIPTOR    pSD;
SECURITY_ATTRIBUTES     sa;

pSD = (PSECURITY_DESCRIPTOR) LocalAlloc( LPTR,
               SECURITY_DESCRIPTOR_MIN_LENGTH);

if (pSD == NULL)
{
   Error(...);
}

if (!InitializeSecurityDescriptor(pSD, SECURITY_DESCRIPTOR_REVISION))
{
   Error(...);
}

// Add a NULL DACL to the security descriptor..

if (!SetSecurityDescriptorDacl(pSD, TRUE, (PACL) NULL, FALSE))
{
   Error(...);
}

sa.nLength = sizeof(sa);
sa.lpSecurityDescriptor = pSD;
sa.bInheritHandle = TRUE;

mutex = CreateMutex( &sa, FALSE, "SOMENAME" );
```

If you are creating one of these objects in an application and the object will be shared with a service, you could also use a null DACL to grant everyone access. As an alternative, you could add an access-control entry (ACE) to the DACL that grants access to the user account that the service is running under. This would restrict access to the object to the service.

For a more detailed example, please see the SERVICES sample.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

Sharing Win32 Services

PSS ID Number: Q91698
Authored 02-Nov-1992 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

Win32 services are discussed in the overview for the Service Control Manager. The documentation says that:

> A Win32 service runs in a Win32 process which it may or may not share with other Win32 services.

Whether or not a service has its own process is determined by which of these service types is specified in the call to CreateService() to add the service to the Service Control Manager Database.

SERVICE_WIN32_OWN_PROCESS

> This service type indicates that only one service can run in the process. This allows an application to spawn multiple copies of a service under different names, each of which gets its own process. This is the most common type of service.

SERVICE_WIN32_SHARE_PROCESS

> This service type indicates that more than one service can be run in a single process. When the second service is started, it is started as a thread in the existing process. A new process is not created. An example of this is the LAN Manage Workstation and the LAN Manager Server. Note that the service must be started in the system account, which is .\System. The name must be NULL.

The service type for each service is stored in the registry. The values are as follows:

> SERVICE_WIN32_OWN_PROCESS 0x10
> SERVICE_WIN32_SHARE_PROCESS 0x20.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseService

Showing the Beginning of an Edit Control after EM_SETSEL

PSS ID Number: Q64758
Authored 09-Aug-1990 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

In a single-line edit control created with the ES_AUTOHSCROLL style, when the EM_SETSEL message is used to select the entire contents of the control, the text in the control is scrolled to the left and the caret is placed at the end of the text. This occurs when the control contains more text than can be displayed at one time. The order of the starting and ending positions specified in the lParam of the EM_SETSEL message makes no difference.

If your application needs to have the entire contents selected and the beginning of the string in view, create the edit control using the ES_MULTILINE style. The order of the starting and ending positions in the EM_SETSEL message is respected by multiline edit controls.

MORE INFORMATION

Consider the following example, which sets and then selects the text in a single-line edit control created with the ES_AUTOHSCROLL style:

```
//hEdit and szText defined elsewhere
SetWindowText(hEdit, szText);
SendMessage(hEdit, EM_SETSEL, 0, MAKELONG(0x7FFF, 0));
```

According to the documentation for the EM_SETSEL message, the low-order word of lParam specifies the starting position of the selection and the high-order word specifies the ending position. However, a single-line edit control ignores this ordering and always selects the text from the lower position to the higher position.

If the content of the edit control is longer than the control can display, the text is scrolled to the end of the selection, and the caret is positioned there. In some situations, it is necessary to show the beginning of the text after the selection is made with EM_SETSEL. In Windows 3.00, there is no documented method to accomplish this positioning using a single-line edit control.

A multiline edit control, sized to display only one line and created without the ES_AUTOVSCROLL style, will appear to the user as a single-line control. However, this control will respect the order of the start and end positions in the EM_SETSEL message.

In the sample code above, a multiline edit control will select the text from the specified starting position to the specified ending position, regardless of which position is higher. In this example, the text is scrolled to the beginning and the caret is placed there. The beginning of the selected text is visible in the control.

NOTE: A multiline edit control uses up slightly more memory in the USER heap than a single-line edit control.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Simulating CreatePatternBrush() on a High-Res Printer

PSS ID Number: Q74793
Authored 30-Jul-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, version 4.0

SUMMARY

When a pattern brush is used to fill an area of the page on the printer, the printer's high resolution will cause a fine pattern to lose definition and appear as a shade of gray.

Brushes that are created with the CreatePatternBrush() function are 8 pixels by 8 pixels (8 x 8 pixels) in size. On a 300 dots-per-inch (dpi) laser printer, the pattern will be 0.027 inches wide.

To create a pattern that gives similar effects on the screen as on the printer, it is necessary to compare the screen resolution to the printer resolution, and to compensate for the differences.

For example, if the video display is 100 dpi (typical of a VGA), and the printer is 300 dpi (a typical laser printer), the bit must be three times larger in each direction. The following compares a screen bitmap and a printer bitmap:

```
   10101010          111000111000111000111000
   01010101          111000111000111000111000
   10101010          111000111000111000111000
   01010101          000111000111000111000111
   10101010          000111000111000111000111     and so forth
   01010101          000111000111000111000111
   10101010          111000111000111000111000
   01010101          111000111000111000111000
                     111000111000111000111000
    Video            000111000111000111000111
                     000111000111000111000111
                     000111000111000111000111
                     111000111000111000111000
                     111000111000111000111000
                     111000111000111000111000
                     000111000111000111000111
                     000111000111000111000111
                     000111000111000111000111
                     111000111000111000111000
                     111000111000111000111000
                     111000111000111000111000

                              Printer
```

However, since the pattern brush is always 8 x 8 pixels, a different approach must be used when printing:

1. Use the StretchBlt() function to create, from the video bitmap, the 24 x 24 pixel bitmap for the printer.

2. Manually "tile" this bitmap into the region to be painted.

Additional reference words: 3.00 3.10 4.00 95
KBCategory: kbprg
KBSubcategory: GdiPrn

Size Comparison of 32-Bit and 16-Bit x86 Applications

PSS ID Number: Q97765
Authored 20-Apr-1993 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

It is expected that a 32-bit version of an x86 application (console or GUI) will be larger than the 16-bit version. Much of this difference is due to the flat memory-model addressing of Windows NT. For each instruction, note that the opcodes have not changed in size, but the addresses have been widened to 32 bits.

In addition, the EXE format under Windows NT (the PE format) is optimized for paging; EXEs are demand-loaded and totally mappable. This leads to some internal fragmentation because protection boundaries must fall on sector boundaries within the EXE file.

The MIPS (or any RISC) version of a Win32-based application typically will be larger and require more memory than its x86 counterpart.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsMisc

SizeofResource() Rounds to Alignment Size

PSS ID Number: Q57808
Authored 17-Jan-1990 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SizeofResource() returns the resource size rounded up to the alignment size. Therefore, if you have your own resource types, you cannot use SizeofResource() to get the actual resource byte count.

It has been suggested that this be changed to reflect the actual number of bytes in the resource so that applications can use SizeofResource() to determine the size of each resource. This suggestion is under review and will be considered for inclusion in a future release of the Windows Software Development Kit (SDK).

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrRsc

SNMP Agent Breaks Up Variable Bindings List

PSS ID Number: Q127870
Authored 21-Mar-1995 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

When the SNMP agent receives a request for mutiple variables in a single packet, then for each entry in the variable bindings list, the agent queries the required sub-agent (in this case the .DLL acting as the agent) and packs up the results in a response variable bindings list and returns it in a single packet.

For example, say the varibles requested are:

```
ip.ipInReceives            (Internet MIB II )
tcp.tcpMaxConn            (Internet MIB II )

.iso.org.dod.internet.private.enterprises.lanmanager.lanmgr-2.common.
comVersionMaj            (LanManager MIB II)

icmp.icmpOutErrors         (Internet MIB II )
```

In this case, the agent queries the INETMIB2.DLL file twice, the LMMIB2.DLL once, and the INETMIB2.DLL once. Then it packs the results in a response packet and sends it to the requesting manager. There is no "snapshot" of the MIB.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: NtwkSnmp

Sockets Applications on Microsoft Windows Platforms

PSS ID Number: Q124876
Authored 15-Jan-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0
- Microsoft Win32s version 1.2

SUMMARY

This article documents the resources necessary to do Winsock development on the different Microsoft Windows platforms. The key components necessary for Winsock programming are:

- TCP/IP networking support
- A Windows Sockets include file
- A Windows Wockets import library
- The Windows Sockets Architecture specification.

NOTE: Some implementations of Windows Sockets may support additional protocols, and TCP/IP will not be strictly necessary. See the documentation from your vendor for more information.

MORE INFORMATION

Where you get the necessary components for Winsock programming depends on what platform you are using.

Microsoft Windows NT Versions 3.1 and 3.5 and Windows 95

Header filename: WINSOCK.H
Import library name: WSOCK32.LIB

If you are using Microsoft Windows NT version 3.1 or 3.5, the TCP/IP protocol is provided as a component of the operating system. Please see your operating system documentation for more information about installing TCP/IP support for Microsoft Windows NT.

The Winsock header file, import library, and specification are all supplied as part of the Win32 SDK. If you do not have the Win32 SDK, it can be purchased as part of the Microsoft Developer Network Level 2 subscription.

The header file and import library are also supplied with the 32-bit editions of Visual C++. Visual C++ does not include the Windows Sockets specification.

Microsoft Windows for Workgroups Version 3.11

Header filename: WINSOCK.H
Import library name: WINSOCK.LIB

Windows for Workgroups does not include support for the TCP/IP protocol. However, Microsoft does provide a TCP/IP protocol for Windows for Workgroups free of charge. To learn how to obtain the TCP/IP package for Windows for Workgroups, please see the following article in the Microsoft Knowledge Base:

   ARTICLE-ID: Q111682
   TITLE   : WFWG 3.11: How to Obtain Microsoft DLC and Microsoft TCP/IP

The Winsock include file, import library, and specification are available from the Microsoft Software library. Download WSA16.EXE, a self-extracting file, from the Microsoft Library (MSL) on the following services:

 - CompuServe
      GO MSL
      Search for WSA16.EXE
      Display results and download

 - Microsoft Download Service (MSDL)
      Dial (206) 936-6735 to connect to MSDL
      Download WSA16.EXE

 - Internet (anonymous FTP)
      ftp ftp.microsoft.com
      Change to the /softlib/mslfiles directory
      get WSA16.EXE

The header file and library for Winsock programming are not included with Visual C++ for Windows.

Microsoft Windows Versions 3.0, 3.1, and 3.11

Header filename: Probably WINSOCK.H (determined by vendor)
Import library name: Probably WINSOCK.LIB (determined by vendor)

TCP/IP support on these versions of Microsoft Windows will have to come from the underlying network. Your network vendor can tell you what TCP/IP support is available.

Depending on your network and the TCP/IP implementation, you may also need to get the Winsock header and library files from your vendor. However, while not guaranteed, the files supplied in WSA16.EXE (see above) may work on your implementation.

Microsoft Win32s Version 1.2

Header filename: WINSOCK.H
Import library name: WSOCK32.LIB

A Winsock application running on Win32s will use the networking support of the underlying system. See the appropriate section above for information about TCP/IP support on the host platform.

The header file, import library, and specification will be part of the Win32 SDK from which the Win32s application was created. See the section on Windows NT above for more information.

Additional reference words: 3.10 3.50 4.00 1.20 tcpip
KBCategory: kbprg kbfile kbnetwork
KBSubcategory: NtwkWinsock W32s

Some Basic Concepts of a Message-Passing Architecture

PSS ID Number: Q74476
Authored 21-Jul-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The following is excerpted from an article in the April 1991 issue of Software Design (Japan).

MORE INFORMATION

Asynchronous message-passing means that Windows will send an application messages to act on and that these messages may come in any order. At present, all messages are sent to specific windows. Every window has a function that Windows calls to send that window a message. This function processes the message and returns to Windows. When the function returns to Windows, Windows may then send messages to other windows in the same program, other programs, or to the same window again.

Because most of these messages are generated by user actions (picking an item in a menu, moving a window, and so forth), the specific messages a window receives will differ each time the program is run. This is what makes the messages asynchronous.

This message passing is what makes Windows programming difficult for many programmers. The programmer is no longer writing a program in which he or she controls the flow from beginning to end. Rather, a Windows program is written as a large number of objects, each one designed to handle a specific message from Windows.

Understanding message passing is critical and because it leads to so much confusion, the concept will be explained in greater detail in this article. If message passing is understood, the remainder of Windows can be learned fairly easily. However, it is very unlikely that a Windows program or other graphical user interface (GUI) program can be successfully developed without a thorough understanding of message passing.

In a message-passing system, the focus changes from being proactive (the programmer controls the program flow) to being reactive (Windows controls the program flow). [Or as it has been put by some Macintosh programmers, "Don't call us, we'll call you."] For example, consider the situation where a user chooses an action from a menu in a program. In a proactive program, the program reads the keyboard, determines that the key(s) pressed are meant to run the action, and calls the function that performs that action. In a reactive system, the program is sent a message indicating that the

user chose that item from a menu. When the program receives the message, it calls the function that performs the action. When this function is done, control returns to the system. Although the reactive approach is substantially different from the proactive approach, it is also simpler.

In Windows, every window (including dialog boxes) has a "response function" registered to it. When Windows sends a message to a window, it calls the response function for that window and passes it the message. All messages from Windows are passed to window response functions; there is no other way for Windows to send a message to a program. Therefore, all messages are for a specific window or group of windows.

However, there are four considerations involved with this method:

1. Messages are sent in two distinct ways. The first method consists of messages that are posted to a first-in, first-out queue (PostMessage). The second method consists of messages that are sent (SendMessage). Posted messages, aside from PAINT messages, are serialized, meaning that messages cannot be posted anywhere except to the end of the queue, and the application is sent messages only from the beginning of the queue. Posted PAINT messages are an exception. They are added together and sent only when there is nothing else in the queue. This is done to reduce the number of times a window has to paint itself.

   Messages that are sent are passed to the application immediately, and the send function does not return until the message is processed. However, when a message is posted, an indeterminate number of messages and amount of time will pass before the message is actually sent to a window and acted on. Also, when a message is sent, an message posted earlier may not yet have been acted on.

2. Sending messages or calling functions (which may, as part of their actions, also send messages), can lead to additional messages being generated. The most dangerous situation is where the action for a message generates the same message again. If, while processing a message, an application sends the same message to itself, the application will run out of stack space quite quickly. If an application POSTS the same message to itself, the application will not run out of stack but it will generate an unending stream of messages.

3. If, while processing a message, an application calls a function that sends a message, the application will process the second message in the middle of processing the first message. Therefore, each window response function MUST be fully re-entrant. It is even possible for a function to be re-entered to process the same message as the message currently being processed. For this reason, using global or static variables in a response function is very dangerous.

   Also, if the application uses properties, scratch files and/or other data storage mechanisms, extreme caution is required. Consider the situation where one message reads in data from a file, then a second message reads in the same data, makes changes to that

data, and writes it back to the file. If the first message makes additional changes to the data and writes its new data back to the file, the changes caused by the second message are completely lost. With files, each application must implement its own sharing mechanism. For properties, allocated memory and other memory storage, there is a simple solution: lock the item and use the pointer returned. Because only one lock is allowed, this prevents contention. Never copy the data into a scratch buffer to copy back later.

4. Because messages come in due to user interaction, the application cannot be written to assume that when a particular message is received that another message has already been processed and performed its functions. While, for a given action, a specific sequence of messages may occur, in the interest of remaining completely compatible with potential changes in future versions of Windows, it is recommended that no message ordering dependencies be introduced unless absolutely necessary. The best example of this is that the first message a window gets when it is being created is not WM_CREATE, rather it is WM_GETMINMAXINFO. When one message must logically follow a second (such as WM_CREATE always preceding WM_DESTROY), then it is fine to depend on the specific ordering of those two messages.

The asynchronous, reactive nature of Windows programming can cause confusion. Because the program has no control over the order that messages arrive, the response to ANY specific message CANNOT depend on other messages having been processed or NOT been processed.

To confuse matters even further, an application may be in the middle of processing one message when it calls a Windows function that sends the application another message. When processing this second message, some dependent processing may be only half finished. If an application will check and only do some processing if another message has not already performed it, the application must be prepared for the case where another message has begun the processing, but has not completed it.

Further, when an application sets up a modal dialog box, the DialogBox() function will not return until after the dialog box is dismissed and processing completed. Therefore, after calling the dialog box function, all combinations of user-generated messages may be received before the function returns.

NOTE: 16-bit Windows is non-preemptively multitasks its Windows tasks. Therefore, an application generally does not need to be designed to process a user-originated message in the middle of processing another message. However, when an application calls a Windows function, the application may then get a set of specific messages sent to it by Windows before the called function returns.

32-bit Windows 95 and Windows NT are preemptive multitasking systems. Each thread has its own input queue. It is a good idea to create a separate thread of execution for the user interface so that it is responsive to user input.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMsg

Some CTRL Accelerator Keys Conflict with Edit Controls

PSS ID Number: Q67293
Authored 29-Nov-1990 Last modified 13-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

Some keys produce the same ASCII values as CTRL+key combinations. These keys conflict with edit controls if one of the CTRL+key combinations is used as a keyboard accelerator.

The following table lists some of the conflicting keys.

ASCII Value	Key Combination	Equivalent	Windows Virtual Key
0x08	CTRL+H	BACKSPACE	VK_BACK
0x09	CTRL+I	TAB	VK_TAB
0x0D	CTRL+M	RETURN	VK_RETURN

For example, consider the following scenario:

1. CTRL+H has been assigned as an accelerator keystroke to invoke Help

2. An edit control has the focus

3. BACKSPACE is pressed to erase the previous character in the edit control

This results in Help being invoked because pressing BACKSPACE is equivalent to pressing CTRL+H. The edit control does not receive the BACKSPACE key press that it requires because TranslateAccelerator() encounters the 0x08 ASCII value and invokes the action assigned to that accelerator. This limitation is caused by the use of the ASCII key code for accelerators instead of the system-dependent virtual key code.

MORE INFORMATION

When messages for the edit control are processed in a message loop that translates accelerators, this translation conflict will occur. Child windows and modeless dialog boxes are the most common situations where this happens.

The affected keystrokes are translated during the processing of the WM_KEYDOWN message for the letter. For example, when the user types CTRL+H, a WM_KEYDOWN is processed for the CTRL key, then another

WM_KEYDOWN is processed for the letter "H". In response to this message, TranslateAccelerator() posts a WM_COMMAND message to the owner of the CTRL+H accelerator. Similarly, when the user presses the BACKSPACE key, a WM_KEYDOWN is generated with VK_BACK as the key code. Because the ASCII value of BACKSPACE is the same as that for CTRL+H, TranslateAccelerator() treats them as the same character. Either sequence will cause a WM_COMMAND message to be sent to the owner of the CTRL+H accelerator, which deprives the child window with the input focus of the BACKSPACE key message.

Because this conflict is inherent to ASCII, the safest way to avoid the difficulty is to avoid using the conflicting sequences as accelerators. Any other ways around the problem may be version dependent rather than a permanent fix.

A second way around the situation is to subclass each edit control that is affected. In the subclass procedure, watch for the desired key sequence(s). The following code sample demonstrates this procedure:

```
/* This code subclasses a child window edit control to allow it to
 * process the RETURN and BACKSPACE keys without interfering with the
 * parent window's reception of WM_COMMAND messages for its CTRL+H
 * and CTRL+M accelerator keys.
 */

/* forward declaration */
long FAR PASCAL NewEditProc(HWND, unsigned, WORD, LONG);

/* required global variables */
FARPROC lpfnOldEditProc;
HWND    hWndOwner;


/* edit control creation in MainWndProc */

TEXTMETRIC tm;
HDC hDC;
HWND hWndEdit;
FARPROC lpProcEdit;

...

 case WM_CREATE:

    hDC = GetDC(hWnd);
    GetTextMetrics(hDC, &tm);
    ReleaseDC(hWnd, hDC);

    hWndEdit = CreateWindow("Edit", NULL,
        WS_CHILD | WS_VISIBLE | ES_LEFT | WS_BORDER,
        50, 50, 50 * tm.tmAveCharWidth, 1.5 * tm.tmHeight,
        hWnd, 1, hInst, NULL);

    lpfnOldEditProc = (FARPROC) GetWindowLong (hWndEdit, GWL_WNDPROC);
    lpProcEdit = MakeProcInstance ((FARPROC) NewEditProc, hInst);
```

```
    SetWindowLong(hWndEdit, GWL_WNDPROC, (LONG) lpProcEdit);
    break;

...

/* subclass procedure */

long FAR PASCAL NewEditProc(HWND hWndEditCtrl, unsigned iMessage,
                            WORD wParam, LONG lParam )
{
  MSG  msg;

  switch (iMessage)
    {
  case WM_KEYDOWN:
    switch (wParam)
        {
    case VK_BACK:
        // This assumes that the next message in the queue will be a
        // WM_COMMAND for the window which owns the accelerators. If
        // this edit control were in a modeless dialog box, hWndOwner
        // should be set to NULL. It may also be NULL in this case.
        PeekMessage(&msg, hWndOwner, 0, 0, PM_REMOVE);

        // Since TranslateAccelerator() processed this message as an
        // accelerator, a WM_CHAR message must be supplied manually to
        // the edit control.
        SendMessage(hWndEditCtrl, WM_CHAR, wParam, MAKELONG(1, 14));
        return 0L;

    case VK_RETURN:
        // Same procedures here.
        PeekMessage(&msg, hWndOwner, 0, 0, PM_REMOVE);
        SendMessage(hWndEditCtrl, WM_CHAR, wParam, MAKELONG(1, 28));
        return 0L;
        }
        break;
    }
    return CallWindowProc(lpfnOldEditProc, hWndEditCtrl, iMessage,
                          wParam, lParam);
}
```

NOTE: Be sure to export the subclass function in the DEF file.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg kbcode
KBSubcategory: UsrCtl

Some Subsystems Persevere Through Logons

PSS ID Number: Q94995
Authored 28-Jan-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

Subsystems such as the Local Security Authority (LSA), the spooler, the Win32 server, and the OS/2 subsystem do not terminate when the user logs off, and then restart at next logon.

Windows on Win32 (WOW) is not a true subsystem; rather, it is a user-mode program. When you run an MS-DOS program, the console you run it from has a virtual DOS machine (VDM) attached to it until the console exits (which it is forced to do at logoff). When you run the first Win16 program, the WOW VDM is started and continues to run until the user logs off.

Additional reference words: 3.10 and 3.50
KBCategory: kbprg
KBSubcategory: Subsys

Source-level Debugging Under NTSD

PSS ID Number: Q99053
Authored 20-May-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

The following are the steps used for source-level debugging under NTSD:

1. Compile using -Zi and -Od.

2. Link using debug:full and debugtype:coff.

3. Load the program into the debugger.

4. Use s+ to change to source mode.

   -or-

   Use s& to change to mixed mode.

For a console application, type "g main" to get to the program start. For a GUI application, type "g WinMain" to get to the program start.

Type "v .<number>" to list source lines starting at <number>. For example, type the following

```
v .20
```

to see all lines starting at line 20.

Additional reference words: 3.10 3.50
KBCategory: kbtool
KBSubcategory: TlsMisc

Specifying Filenames Under the POSIX Subsystem

PSS ID Number: Q99361
Authored 26-May-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

When specifying filenames under POSIX, use the format

```
//C/subdir/executable.exe
```

to specify C:\SUBDIR\EXECUTABLE.EXE. If you fail to use this format, you will receive ENAMETOOLONG as the errno.

NOTE: The filenames are case-sensitive.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: Subsys

Specifying Serial Ports Larger than COM9

PSS ID Number: Q115831
Authored 05-Jun-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

CreateFile() can be used to get a handle to a serial port. The "Win32 Programmer's Reference" entry for "CreateFile()" mentions that the share mode must be 0, the create parameter must be OPEN_EXISTING, and the template must be NULL.

CreateFile() is successful when you use "COM1" through "COM9" for the name of the file; however, the message "INVALID_HANDLE_VALUE" is returned if you use "COM10" or greater.

If the name of the port is \\.\COM10, the correct way to specify the serial port in a call to CreateFile() is as follows:

```
   CreateFile(
      "\\\\.\\COM10",     // address of name of the communications device
      fdwAccess,          // access (read-write) mode
      0,                  // share mode
      NULL,               // address of security descriptor
      OPEN_EXISTING,      // how to create
      0,                  // file attributes
      NULL                // handle of file with attributes to copy
   );
```

NOTES: This syntax also works for ports COM1 through COM9. Certain boards will let you choose the port names yourself. This syntax works for those names as well.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseCommapi

Specifying Time to Display and Remove a Dialog Box

PSS ID Number: Q74888
Authored 01-Aug-1991 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

It is possible to modify the timing of the display of a dialog box. For example, an application has its copyright message in a dialog box that does not have any push buttons. This dialog box is designed to be displayed for five seconds and then to disappear. This article discusses a method to implement this functionality.

MORE INFORMATION

Windows draws the dialog box on the screen during the processing of a WM_PAINT message. Because all other messages (except for WM_TIMER messages) are processed before WM_PAINT messages, there may be some delay before the dialog box is painted. This delay may be avoided by placing the following code in the processing of the WM_INITDIALOG message:

```
ShowWindow(hDlg);
UpdateWindow(hDlg);
```

This code causes Windows to send a WM_PAINT message to the dialog box, bypassing the other messages that may be in the application's queue.

To keep the dialog box on the screen for a particular period of time, a timer should be created during the processing of the WM_INITDIALOG message. When the WM_TIMER message is received, call EndDialog() to close the dialog box.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDlgs

Specifying Windows "Bounding Box" Coordinates

PSS ID Number: Q27585
Authored 07-Mar-1988 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

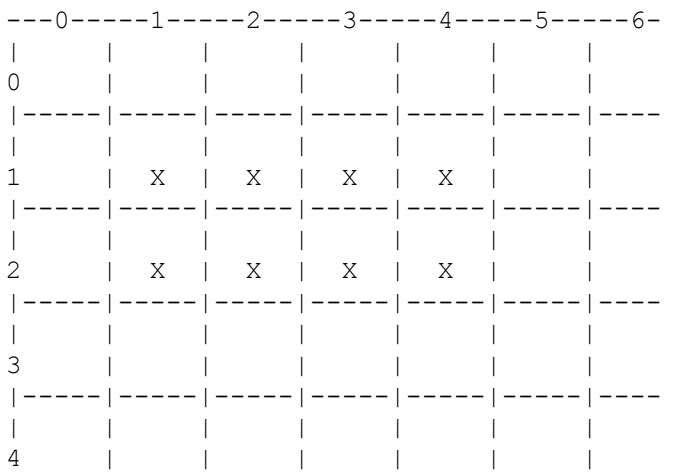
GDI functions, such as Rectangle, Ellipse, RoundRect, Chord, and Pie, have parameters that specify the coordinates of a "bounding box" into which the figure is drawn. Windows draws the figure up to, but not including, the right and bottom coordinates.

MORE INFORMATION

Suppose the following call is made:

```
Rectangle(hDC, 1, 1, 5, 3)
```

Assuming that the device context is using the MM_TEXT mapping mode (in which case logical units map directly to physical pixels), the resulting rectangle will be 4 pixels wide and 2 pixels tall. The following diagram shows which pixels are affected:



It may be helpful to think of the display as a grid, with each pixel contained in a grid cell. The X1, Y1, X2, and Y2 parameters to the Rectangle function specify an imaginary "bounding box" drawn on the grid. The rectangle is drawn within the bounding box.

The height, width, and area of the resulting rectangle have the following useful properties:

```
   Height = X2 - X1
   Width  = Y2 - Y1
   Area   = Height * Width
```

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiDrw

Starting and Terminating Windows-Based Applications

PSS ID Number: Q105676
Authored 22-Oct-1993 Last modified 04-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

A 16-bit Windows-based application running under Windows NT version 3.1 is running as a thread in a single virtual MS-DOS machine (VDM). These threads are nonpreemptively scheduled. A 16-bit Windows-based application shares an address space and an input queue with other 16-bit Windows-based applications. Objects created by a thread (application) are owned by the thread (application). This environment is called WOW (Windows on Win32).

Windows NT version 3.5 introduces support for multiple WOWs, so each 16-bit Windows-based application can be run in its own address space. For additional information, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID: Q115235
TITLE : Running a Windows-Based Application in its Own VDM

When a 16-bit Windows-based application is started by using CreateProcess(), the process handle and the thread handle contained in the PROCESS_INFORMATION structure are pseudo-handles. The only application programming interfaces (APIs) that can use the process handle are WaitForSingleObject(), WaitForMultipleObjects(), and WaitForInputIdle().

NOTE: When you call CreateProcess() with CREATE_SEPARATE_WOW_VDM, the handles returned in the hProcess and hThread fields of the PROCESS_INFORMATION structure are valid handles. The hProcess handle is the handle to the new WOW process, but the hThread handle is the handle to the WOWEXEC thread, and not the thread of the application that you spawned. You can wait on either of these handles and you will be released when the Windows-based application terminates, because the separate VDM goes away when the last application in it terminates. Be aware that if your Windows-based application spawns another Windows-based application and terminates, the VDM stays around, because the application is run in the same VDM as the application that spawned it.

A common question is "How can I terminate a 16-bit process from a 32-bit process?" However, as implied above, PROCESS_INFORMATION.hProcess cannot be used in TerminateProcess() and PROCESS_INFORMATION.dwThreadId cannot be used in PostThreadMessage().

One way to terminate an individual 16-bit Windows-based application is to enumerate the desktop windows using EnumWindows(), determine which is the correct window, obtain the thread ID with GetWindowThreadProcessId(), and post a WM_QUIT message via PostThreadMessage() to terminate the application.

Another way to terminate an individual 16-bit Windows-based application is to use the TOOLHELP APIs. Use TaskFirst() and TaskNext() to enumerate the tasks, determine which is the correct task, and call TerminateApp() to kill the application. The Windows SDK sample THSAMPLE demonstrates how this can be done.

Additional reference words: 3.10 3.50 win16
KBCategory: kbprg
KBSubcategory: SubSys

StartPage/EndPage Resets Printer DC Attributes in Windows 95

PSS ID Number: Q125696
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

When you print under Windows version 3.x, the printer device context attributes, including things like mapping modes, current pen, current brush, and so on, are reset to their defaults for the device when the end of a page is reached. The escapes NEWFRAME and NEXTBAND and the API EndPage all cause the printer device context to be reset.

When you print under Windows NT version 3.x, the printer device context attributes are not reset during a print job.

When you print under Windows 95, the point at which the printer device context is reset to the default attributes depends on what version the executable was marked as. For executables marked as 3.x, the printer device context will be reset when EndPage is called. For executables marked as 4.0, the printer device context will be reset when StartPage is called. This applies to both 16-bit-based and Win32-based executables running under Windows 95.

A 16-bit-based executable's version can be set by using the Resource compiler's /xx switch where xx is 30, 31, or 40. A Win32-based executable's version can be set by using the /SUBSYSTEM:windows,x.x linker switch.

Additional reference words: 4.00
KBCategory: kbprg
KBSubcategory: GdiPrn

Stroke Fonts Marked as OEM Character Set Are ANSI

PSS ID Number: Q72020
Authored 10-May-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

There are three stroke (or vector) fonts packaged with Windows versions 3.0 and 3.1 and Windows NT: Roman, Script, and Modern. These fonts are marked as belonging to the OEM character set when, in fact, they belong to the ANSI character set. NOTE: Windows 95 provides only the Modern vector font. The Roman and Script fonts are included in the True Type fonts shipped with the system.

The OEM character set is the character set used by the hardware device on which Windows is running (for example, the IBM PC). The IBM PC OEM character set and ANSI character set are listed in "Microsoft Windows Software Development Kit Reference Volume 2" for the Windows SDK version 3.0 and in "Programmer's Reference, Volume 3: Messages, Structures, and Macros" for the Windows SDK version 3.1.

The fonts were marked in this manner for two reasons. First, in previous versions of Windows, the stroke fonts did include non-ANSI characters. Second, mismarking the character set ensures proper font mapping. The character-set attribute of a font is assigned a very high penalty weight in the font mapping scheme. If stroke fonts were not marked as using the OEM character set, a stroke font might be chosen by the font mapper [during a SelectObject() call] instead of a raster font when a requested raster font size is not available. This behavior occurs because most raster fonts belong to the ANSI character set, character size has much lower penalty weight than character set, and stroke fonts can be scaled to any desired size. Some raster fonts can be scaled; however, they can be scaled only to specific sizes.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiFnt

Support for Sleep() on Win32s

PSS ID Number: Q100713
Authored 27-Jun-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.0, 1.1, and 1.2

The Win32 application programming interface (API) documentation indicates that Sleep() is supported on Win32s. It is important to note, however, that the behavior of Sleep() on Win32s is not the same as it is under Windows NT.

Under Win32s, Sleep() calls Yield(). The Windows version 3.1 Yield() function yields only if the message queue is empty; therefore, Sleep() cannot be relied on to do anything. Use a PeekMessage() loop to do idle time processing.

Additional reference words: 1.00 1.10 1.20
KBCategory: kbprg
KBSubcategory: W32s

Switching Between Single and Multiple List Boxes

PSS ID Number: Q57959
Authored 23-Jan-1990 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

After creating a list box with the CreateWindow() API, changing the list box from a single selection to a multiple selection can be accomplished in the following way:

Create two hidden list boxes in the .RC file, and during the WM_INITDIALOG routine, display one of the boxes. Change between the two by making one hidden and the other one visible using the ShowWindow function.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Symbolic Information for System DLLs

PSS ID Number: Q103862
Authored 01-Sep-1993 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, and 3.51

SUMMARY

The debugging information for the system dynamic-link libraries (DLLs) is contained separately in files with a .DBG extension. The Win32 SDK setup, SETUPSDK, will install the following .DBG files by default in <SYSTEMROOT>\SYMBOLS\DLL:

ADVAPI32.DBG	OLECLI32.DBG
COMDLG32.DBG	OLESVR32.DBG
CRTDLL.DBG	RASAPI32.DBG
DLCAPI.DBG	RPCNS4.DBG
GDI32.DBG	RPCRT4.DBG
INETMIB1.DBG	SHELL32.DBG
KERNEL32.DBG	USER32.DBG
LMMIB2.DBG	VDMDBG.DBG
LZ32.DBG	VERSION.DBG
MGMTAPI.DBG	WIN32SPL.DBG
MPR.DBG	WINMM.DBG
NDDEAPI.DBG	WINSTRM.DBG
NETAPI32.DBG	WSOCK32.DBG
NTDLL.DBG	

For Windows NT version 3.5, the additional .DBG files are:

GLU32.DBG
MSACM32.DBG
OLEAUT32.DBG
OPENGL32.DBG

For Windows NT versions 3.51, the additional .DBG files are:

COMCTL32.DBG
DHCPMIB.DBG
OLE32.DBG
OLEDLG.DBG
RICHED32.DBG
WINSMIB.DBG

Note that these files are not installed by the alternative Win32 SDK install method, MANUAL.BAT. Therefore, WinDbg will warn that symbol information cannot be found for each of the system DLLs called by the debuggee.

These .DBG files can be manually installed by copying them from the SDK CD. For x86, they are located in \SUPPORT\DEBUG\I386\SYMBOLS\DLL. For MIPS, they are located in \SUPPORT\DEBUG\MIPS\SYMBOLS\DLL. Note that there are more than 200 .DBG files in each of these directories.

MORE INFORMATION

In Windows NT 3.1, there are also debugging versions of the system DLLs that can be installed by using SWITCH.BAT, which is located on the CD in \SUPPORT\DEBUGDLL. Refer to page 11 of the "Getting Started" manual and the batch file itself for more information.

Additional reference words: 3.10 3.50
KBCategory: kbsetup
KBSubcategory: Setins

System GENERIC_MAPPING Structures

PSS ID Number: Q102105
Authored 28-Jul-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1

There is not a Win32 application programming interface (API) to retrieve the GENERIC_MAPPING structures for Windows NT objects. The MapGenericMask() Win32 API is intended to use GENERIC_MAPPING structures associated with private objects created by the application.

Additional reference words: 3.10
KBCategory: kbprg
KBSubcategory: BseSecurity

System Versus User Locale Identifiers

PSS ID Number: Q100488
Authored 22-Jun-1993 Last modified 24-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

In Windows NT version 3.1, the following pairs of application programming interfaces (APIs) have the same functionality:

```
GetSystemDefaultLCID()   and GetUserDefaultLCID()
GetSystemDefaultLangID() and GetUserDefaultLangID()
```

The user LangID and LCID are always set to the system value. In future versions of Windows NT, it will be possible to set the LangID and the LCID on a per-user basis.

Note that it is possible to set the LCID on a per-thread basis [that is, SetThreadLocale()] in Windows NT 3.1.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: UsrLoc WintlDev

Taking Ownership of Registry Keys

PSS ID Number: Q111546
Authored 14-Feb-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

To take ownership of a registry key it is necessary to have a handle to the key. A handle to the key can be obtained by opening the key with a registry API (application programming interface) such as RegOpenKeyEx(). If the user does not have access to the registry key, the open operation will fail and this will in turn prevent ownership being taken (because a handle to the key is required to change the key's security).

The solution to this problem is to first enable the TakeOwnership privilege and then to open the registry key with WRITE_OWNER access as shown below:

```
RegOpenKeyEx(HKEY_CLASSES_ROOT,"Testkey",0,WRITE_OWNER,&hKey);
```

This function call will provide a handle to the registry, which can be used in the following call to take ownership:

```
RegSetKeySecurity(hKey,OWNER_SECURITY_INFORMATION, &SecurityDescriptor);
```

Please note that you will need to initialize the security descriptor being passed to RegSetKeySecurity() and set the owner field to the new owner SID.

Taking ownership of a registry key is not a common operation. It is typically an operation that an administrator would use as a last resort to gain access to a registry key.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

Text Alignment in Single Line Edit Controls

PSS ID Number: Q108940
Authored 20-Dec-1993 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

Text in single line edit controls cannot be centered or right-aligned (right-justified). Text in single line edit controls is left-aligned (left-justified) by default. This is by product design, and therefore specifying the ES_RIGHT or ES_CENTER style while creating the single line edit control does not have any effect on the text alignment.

Windows does not allow text to be centered or right-aligned in a single line edit control. However, an easy way to work around this problem is to create a multiline edit control that is the same size as a single line edit control.

Text in a multiline edit control can be centered or right/left aligned. Note that the multiline edit control should be created without the WS_VSCROLL or ES_AUTOVSCROLL style.

While single line edit controls may support text alignment in a future release of Windows, Windows 3.x applications must use multiline edit controls to achieve the same effect.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

The Clipboard and the WM_RENDERFORMAT Message

PSS ID Number: Q31668
Authored 15-Jun-1988 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

The clipboard sends a WM_RENDERFORMAT message to an application to request that application format the data last copied to the clipboard in the specified format, and then pass a handle to the formatted data to the clipboard.

If an application cannot supply the requested data, it should return a NULL handle. Because most applications provide access to the actual data (not rendered) through the CF_TEXT format, applications that use the clipboard can get the applicable data when rendering fails.

If the application cannot render the data because the system is out of memory, the application can call GlobalCompact(-1) to discard and compress memory, then try the memory allocation request again.

If this fails to provide enough memory, the application can render the data into a file. However, applications that use this technique must cooperate in order to know that the information is in a file, the name of the file, and the format of the data.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrClp

The Parts of a Windows Combo Box and How They Relate

PSS ID Number: Q65881
Authored 26-Sep-1990 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

A Windows combo box is a compound structure composed of individual windows. Three types of windows can be created as part of a combo box:

- A combo box itself, of window class "ComboBox"
- An edit control, of window class "Edit"
- A list box, of window class "ComboLBox"

The relationship among these three windows varies depending upon the different combo box styles.

MORE INFORMATION

For combo boxes created with the CBS_SIMPLE styles, the ComboBox window is the parent of the edit control and the list box that is always displayed on the screen. When GetWindowRect() is called for a combo box of this style, the rectangle returned contains the edit control and the list box.

Combo boxes created with the CBS_DROPDOWNLIST style have no edit control. The region of the combo box that displays the current selection is in the ComboBox window itself. When GetWindowRect() is called for a combo box of this style, the rectangle returned does not include the list box.

For combo boxes created with the CBS_DROPDOWN style, three windows are created. The combo box edit control is a child of the ComboBox window. When GetWindowRect() is called for a combo box of this style, the rectangle returned does not include the list box.

However, the ComboLBox (list box) window for combo boxes that have the CBS_DROPDOWN or CBS_DROPDOWNLIST style is not a child of the ComboBox window. Instead, each ComboLBox window is a child of the desktop window. This is required so that, when the drop-down list box is dropped, it can extend outside the application window or dialog box. Otherwise, the list box would be clipped at the window or dialog box border.

Because the ComboLBox window is not a child of the ComboBox window, there is no simple method to get the handle of one window, given the other. For example, given a handle to the ComboBox, the handle to any associated drop-down list box is not readily available. The ComboLBox is a private class registered by USER that is a list box with the class style CS_SAVEBITS.

Additional reference words: control focus release 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

The SBS_SIZEBOX Style

PSS ID Number: Q102485
Authored 03-Aug-1993 Last modified 09-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

A size box is a small rectangle the user can expand to change the size of the window. When you want a size box, you create a SCROLLBAR window with the SBS_SIZEBOX flag. This action creates a size box with the height, width, and position that you specified in the call to CreateWindow. If you specify SBS_SIZEBOXBOTTOMRIGHTALIGN, the box will be aligned in the lower right of the rectangle you specified when creating the window. If you specify SBS_SIZEBOXTOPLEFTALIGN, the box will be aligned in the upper left of the rectangle you specified in your call to CreateWindow().

MORE INFORMATION

The user moves the mouse pointer over to the box, presses and holds the left mouse button, and drags the mouse pointer to resize the window. When the user does this, the borders on the window (the frame) move. When the user releases the mouse button, the window is resized.

You create a size box by creating a child window of type WS_CHILD | WS_VISIBLE | SBS_SIZEBOX | SBS_SIZEBOXTOPLEFTALIGN. You don't have to do any of the processing for this; the system will take care of it. You will notice in your window procedure that you will get the scroll bar messages plus the WM_MINMAXINFO message. Size boxes work similar to the way the WS_THICKFRAME/WS_SIZEBOX style does on a window.

Under Windows NT, applications that create a size box either using WS_SIZEBOX or WS_THICKFRAME or by created the SBS_SIZEBOX control have no way of showing the user that such a feature exists. Under Windows 95, the size box appears as a jagged edge, usually at the bottom right corner.

NOTE: Make sure that the main window is created with the WS_VSCROLL and WS_HSCROLL styles.

Additional reference words: 3.10 3.50 4.00 sizebox
KBCategory: kbprg
KBSubcategory: UsrCtl

The Use of PAGE_WRITECOPY

PSS ID Number: Q105532
Authored 20-Oct-1993 Last modified 21-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

The documentation to CreateFileMapping(), VirtualAlloc(), and VirtualProtect() indicates that the PAGE_WRITECOPY protection gives copy-on-write access to the committed region of pages. As it is, PAGE_WRITECOPY makes sense only in the context of file mapping, where you want to map something from the disk into your view and then modify the view without causing the data to go on the disk.

The only case where VirtualAlloc() should succeed with PAGE_WRITECOPY is the case where CreateFileMapping() is called with -1 and allocates memory with the SEC_RESERVE flag and later on, VirtualAlloc() is used to change this into MEM_COMMIT with a PAGE_WRITECOPY protection.

There is a bug in Windows NT 3.1 such that the following call to VirtualAlloc() will succeed:

```
lpCommit = VirtualAlloc(lpvAddr, cbSize, MEM_COMMIT, PAGE_WRITECOPY);
```

This call will fail under Windows NT 3.5.

NOTE: lpvAddr is a pointer to memory that was allocated with MEM_RESERVE and PAGE_NOACCESS.

One case where this might be useful is when emulating the UNIX fork command. Emulating fork behavior would involve creating instance data and using threads or multiple processes.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseMm

The Use of the SetLastErrorEx() API

PSS ID Number: Q97926
Authored 25-Apr-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SetLastErrorEx() is intended for better debugging support, not for passing additional error information.

The SetLastErrorEx() application programming interface (API) differs from the SetLastError() API only in that it raises a debug "RIP" event. The RIP event is intended to give text to the debugger so that the user can retry, ignore, and so forth, these errors. SetLastErrorEx() raises an exception only if SetDebugErrorLevel() has been called by the debugger to allow the errors to be passed on.

The error type can be determined from the debugger by examining the debug event structure that is passed with the event. The debug event structure contains a RIP_INFO substructure.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseMisc

Thread Handles and Thread IDs

PSS ID Number: Q127992
Authored 22-Mar-1995 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.5, 3.51, and 4.0

The CreateThread() API is used to create threads. The API returns both a thread handle and a thread identifier (ID). The thread handle has full access rights to the thread object created. The thread ID uniquely identifies the thread on the system level while the thread is running. The ID can be recycled after the thread has been terminated. This relationship is similar to that of the process handle and the process ID (PID).

There is no way to get the thread handle from the thread ID. While there is an OpenProcess() API that takes a PID and returns the handle to the process, there is no corresponding OpenThread() that takes a thread ID and returns a thread handle.

The reason that the Win32 API does not make thread handles available this way is that it can cause damage to an application. The APIs that take a thread handle allow suspending/resuming threads, adjusting priority of a thread relative to its process, reading/writing registers, limiting a thread to a set of processors, terminating a thread, and so forth. Performing any one of these operations on a thread without the knowledge of the owning process is dangerous, and may cause the process to fail.

If you will need a thread handle, then you need to request it from the thread creator or the thread itself. Both the creator or the thread will have a handle to the thread and can give it to you using DuplicateHandle(). This requirement allows both applications to coordinate their actions.

NOTE: You can also take full control of the application by calling DebugActiveProcess(). Debuggers receive the thread handles for a process when the threads are created. These handles have THREAD_GET_CONTEXT, THREAD_SET_CONTEXT, and THREAD_SUSPEND_RESUME access to the thread.

Additional reference words: 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseProcThrd

Thread Local Storage Overview

PSS ID Number: Q94804
Authored 18-Jan-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

Thread local storage (TLS) is a method by which each thread in a given process is given a location(s) in which to store thread-specific data.

Four functions exist for TLS: TlsAlloc(), TlsGetValue(), TlsSetValue(), and TlsFree(). These functions manipulate TLS indexes, which refer to storage areas for each thread in a process. A given index is valid only within the process that allocated it.

Note that the Visual C++ compiler supports an alternate syntax:

```
_declspec( thread )
```

which can be used in place of directly calling these APIs. Please see the description of _declspec in the VC++ "Language and Run-time Help" helpfile for more information.

MORE INFORMATION

A call to TlsAlloc() returns a global TLS index. This one TLS index is valid for every thread within the process that allocated it, and should therefore be saved in a global or static variable.

Thread local storage works as follows: when TlsAlloc() is called, every thread within the process has its own private DWORD-sized space reserved for it (in its stack space, but this is implementation-specific). However, only one TLS index is returned. This single TLS index may be used by each and every thread in the process to refer to the unique space that TlsAlloc() reserved for it.

For this reason, TlsAlloc() is often called only once. This is convenient for DLLs, which can distinguish between DLL_PROCESS_ATTACH (where the first process's thread is connecting to the DLL) and DLL_THREAD_ATTACH (subsequent threads of that process are attaching). For example, the first thread calls TlsAlloc() and stores the TLS index in a global or static variable, and every other thread that attaches to the DLL refers to the global variable to access their local storage space.

Although one TLS index is usually sufficient, a process may have up to TLS_MINIMUM_AVAILABLE indexes (guaranteed to be greater than or equal to 64).

Once a TLS index has been allocated (and stored), the threads within the process may use it to set and retrieve values in their storage spaces. A thread may store any DWORD-sized value in its local storage (for example, a DWORD value, a pointer to some dynamically allocated memory, and so forth). The TlsSetValue() and TlsGetValue() APIs are used for this purpose.

A process should free TLS indexes with TlsFree() when it has finished using them. However, if any threads in the process have stored a pointer to dynamically allocated memory within their local storage spaces, it is important to free the memory or retrieve the pointer to it before freeing the TLS index, or it will be lost.

For more information, please see "Using Thread Local Storage" in the "Processes and Threads" overview in the "Win32 Programmer's Reference".

Example

Thread A within a process calls TlsAlloc(), and stores the index returned in the global variable TlsIndex:

```
   TlsIndex = TlsAlloc();
```

Thread A then allocates 100 bytes of dynamic memory, and stores it in its local storage:

```
   TlsSetValue( TlsIndex, malloc(100) );
```

Thread A creates thread B, which stores a handle to a window in its local storage space referred to by TlsIndex.

```
   TlsSetValue( TlsIndex, (LPVOID)hSomeWindow );
```

Note that TlsIndex refers to a different location when thread B uses it, than when thread A uses it. Each thread has its own location referred to by the same value in TlsIndex.

Thread B may terminate safely because it does not need to specifically free the value in its local storage.

Before thread A terminates, however, it must first free the dynamically allocated memory in its local storage

```
   free( TlsGetValue( TlsIndex ) );
```

and then free the TLS index:

```
   if ( !TlsFree( TlsIndex ) )
      // TlsFree() failed.  Handle error.
```

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseProcThrd

Time Stamps Under the FAT File System

PSS ID Number: Q101186
Authored 07-Jul-1993 Last modified 15-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

Windows NT considers the time stamp on a file stored on a FAT (file allocation table) partition to be standard time if the current time is standard time, and daylight time if the current time is daylight time, regardless of what time of year the file was originally time stamped.

This is not an issue under NTFS, which consistently implements Universal Coordinated Time.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseFileio

Timer Resolution in Windows NT

PSS ID Number: Q115232
Authored 22-May-1994 Last modified 01-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

In Win32-based applications, the GetTickCount() timer resolution on Windows NT, version 3.1, is 15 milliseconds (ms) for x86 and 10 ms for MIPS and Alpha. On Windows NT, version 3.5, the GetTickCount() timer resolution is 10 ms on the 486 or greater, but the resolution is still 15 ms on a 386.

NOTE: The measurements in milliseconds indicate the period of the interrupt, not the units of the returned value.

The Win32 API QueryPerformanceCounter() returns the resolution of a high-resolution performance counter if the hardware supports one. For x86, the resolution is about 0.8 microseconds (0.0008 ms). For MIPS, the resolution is about twice the clock speed of the processor. You need to call QueryPerformanceFrequency() to get the frequency of the high-resolution performance counter.

NOTE: These numbers are likely to change in future versions.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseMisc

Tips for Installing TAPI Service Providers

PSS ID Number: Q131356
Authored 08-Jun-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0
- Microsoft Windows Telephony Software Development Kit (TAPI SDK) version 1.0

SUMMARY

When installing TAPI Service Providers (TSPs), there is more involved than just copying the TSP files to the system directory. The TAPI system files may need to be installed, and the TSP must be installed into TAPI. However, this process can vary greatly from one Windows version to another.

MORE INFORMATION

First and foremost, be sure to do version checking when installing all files.

Second, use a setup application rather than relying on an .INF file. While the ATSP sample uses a .INF file to install, there are two major problems with using a .INF file:

- TAPI needs to already be installed (which isn't guarenteed under Windows version 3.1 or Windows for Workgroups).

- You can't do operating system version checking if you use an .INF file.

Files Necessary for Windows Version 3.1 or Windows for Workgroups

When a TSP is installed under Windows version 3.1 or Windows for Workgroups, it must check and make sure all the necessary system files are present. The files to be installed can be found in the TAPI 1.0 SDK \REDIST directory and must be distributed with the TSP. Some of these are optional and some are not. Here is a list of files and where they should be placed:

Mandidory TAPI files:

```
TELEPHON.CPL [system]
TAPI.DLL     [system]
TAPIADDR.DLL [system]
TAPIEXE.EXE  [system]
TELEPHON.HLP [system]
```

Optional TAPI files:

```
ATSPEXE.EXE  [system]
```

```
    ATSP.TSP     [system]
    ATSP.HLP     [system]
    DIALER.EXE   [windows]
    DIALER.HLP   [windows]
```

Files Necessary for Windows 95

Because the Windows 95 system files are not redistributable and because TAPI is installed automatically under Windows 95, the TAPI system files must not be distributed with the TSP. The TAPI version 1.0 files are not compatible with Windows 95, so it is important that no TAPI system files are installed under Windows 95.

One complication for Windows 95 is the TELEPHON.CPL file. This file is installed to the system directory by default along with all the rest of the Windows 95 TAPI files. However, because this file is not needed by most people using Windows 95, it is installed in the system directory as TELEPHON.CP$ to reduce control panel clutter. TSPs that need this applet should first look in the system directory for TELEPHON.CPL; if that isn't found, the TSP should locate TELEPHON.CP$ (also looking in the system directory), and rename it to TELEPHON.CPL.

Installing the TSP

Once all the files are installed, the TSP must now be installed into TAPI. Under Windows version 3.1, this is done through the Telephony Control Panel applet (TELEPHON.CPL). There are no APIs to automate the installation, so TSP installation programs must either tell the user how to install their TSP into TAPI or use an application such as MSTEST that can simulate keystrokes to automate this part of the installation. Note that TSPs must have version information (as demonstrated by the ATSP sample) to show in the Add Driver dialog.

Adding TSPs through the Telephony Control Panel is also available under Windows 95, but the exact sequence of keystrokes is slightly different. However, several TAPI APIs (lineAddProvider, lineConfigProvider, and lineRemoveProvider) have been added that make the TELEPHON.CPL unnecessary. It's a good idea to have each TSP have its own control panel applet that calls these APIs, rather than rely on TELEPHON.CPL. The setup program should install the TSP directly by using these APIs instead of asking the user to run a control panel applet.

Additional reference words: 1.00 1.30 1.40 4.00 95
KBCategory: kbprg
KBSubcategory: TAPI

Tips for Writing Multiple-Language Scripts

PSS ID Number: Q89865
Authored 01-Oct-1992 Last modified 24-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

To aid you in writing multiple-language resources, the Win32 development system supports language scripts.

MORE INFORMATION

To create a multiple language script, first create a single-language script file (American English, for example), and duplicate the translations in your script file. You need a complete translation only once for each major language. Only those resources that have differences between the major language and the sublanguage need be included in the sublanguage areas of the script. The system will use the main language resource if it doesn't find a resource for the sublanguage.

Sample Code

```
LANGUAGE LANG_ENGLISH,SUBLANG_ENGLISH_US
<original script file>
LANGUAGE LANG_ENGLISH,SUBLANG_ENGLISH_UK
<portions of script file that are different for UK>
LANGUAGE LANG_ENGLISH,SUBLANG_ENGLISH_AUS
<portions of script file that are different for Australia>
LANGUAGE LANG_ENGLISH,SUBLANG_ENGLISH_CAN
<portions of script file that are different for Canada>
LANGUAGE LANG_FRENCH,SUBLANG_FRENCH
<entire script file translated to French>
LANGUAGE LANG_FRENCH,SUBLANG_FRENCH_CAN
<portions of script file that are different for Canada>
LANGUAGE LANG_FRENCH,SUBLANG_FRENCH_SWISS
<portions of script file that are different for Switzerland>
LANGUAGE LANG_FRENCH,SUBLANG_FRENCH_BELGIAN
<portions of script file that are different for Belgium>
LANGUAGE LANG_SPANISH,SUBLANG_SPANISH
<entire script file translated to Spanish>
```

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: WIntlDev

Tips for Writing Windows Sockets Apps That Use AF_NETBIOS

PSS ID Number: Q129316
Authored 24-Apr-1995 Last modified 04-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5

SUMMARY

The six issues listed in this article must be addressed when writing Windows Sockets applications that use the AF_NETBIOS protocol family.

NOTE: Windows 95 does not support AF_NETBIOS.

MORE INFORMATION

Issues to Address When Writing Windows Sockets Applications
---

1. When creating an AF_NETBIOS socket, specify -1 times lana for the protocol option.

   The Windows NT WinSock library allows the programmer to create a socket that allows communication over a particular lana. The lana number is specified via the protocol option of the socket() function. To create an AF_NETBIOS socket, specify -1 times the lana for the protocol option of the socket() function. For example:

```
SOCKET hSock0 = socket( AF_NETBIOS, SOCK_DGRAM, 0 );  // lana 0;
SOCKET hSock1 = socket( AF_NETBIOS, SOCK_DGRAM, -1 ); // lana 1;
```

   The lana numbers are basically indices to the list of transports supported by the NetBIOS implementation. A given host has one unique lana number for every installed transport that supports NetBIOS. For example, listed below are some possible lana numbers for a typical Windows NT configuration:

Lana	Transport
0	TCP/IP // lana 0 is the default NetBIOS transport
1	IPX/SPX w/NetBIOS
3	NetBEUI

   In the case of a multihomed host (a machine with multiple network adapters), the number of unique lana numbers equals the number of network transports that support NetBIOS times the number of network adapters. For example, if the machine depicted above contained two network adapters, it would have a total of 3 * 2 = 6 lana numbers.

Also, please note the WSNETBS.H header included with the Windows NT version 3.5 SDK erroneously defines the NBPROTO_NETBEUI symbol. This symbol cannot be used as a protocol option and should be ignored.

2. Use the snb_type values provided by the WSNETBS.H header for NetBIOS name registration and deregistration.

   When filling out a sockaddr_nb structure, you must specify the appropriate value for the snb_type field. This field is used during the bind() operation to handle NetBIOS name registration. The WSNETBS.H header defines several values for this field; however only the following two values are currently implemented:

   - NETBIOS_UNIQUE_NAME

     Registers a unique NetBIOS name. This action is usually performed by a client or a server to register an endpoint.

   - NETBIOS_GROUP_NAME

     Registers a group name. This action is typically performed in preparation for sending or receiving NetBIOS multicasts.

   Names are registered during the bind() operation.

3. Use the supported socket types of SOCK_DGRAM and SOCK_SEQPACKET.

   Due to the nature of NetBIOS connection services, SOCK_STREAM is not supported.

4. Choose a NetBIOS port that does not conflict with your network client software.

   The NetBIOS port is an eight-bit value stored in the last position of the snb_name that is used by various network services to differentiate various type of NetBIOS names. When you register NetBIOS names, choose port values that do not cause conflicts with existing network services. This is of particular importance if you are registering a NetBIOS name that duplicates a user name or a machine name on the network. The following lists the reserved port values:

   0x00, 0x03, 0x06, 0x1f, 0x20, 0x21, 0xbe, 0xbf, 0x1b, 0x1c, 0x1d, 0x1e

5. Applications should use all available lana numbers when initiating communication.

   Because the NetBIOS interface can take advantage of multiple transport protocols, it is important to use all lanas when initiating communication. Server applications should accept connections on sockets for each lana number, and client applications should attempt to connect on every available lana. In a similar fashion, data gram broadcasts should be sent from sockets created from each lana.

   The following diagram depicts the lana mappings for two machines.

In order for a client application running on Machine A to communicate with a server application on Host B, the client application must create a socket on lana 3, and the server must create a socket on lana 1. Because the client and the server cannot know in advance which single lana to use, they must create sockets for all lanas.

```
   Host A (Client)            Host B (Server)
   ---------------            ---------------
   lana  Transport            lana  Transport
   ----  ---------            ----  ---------

   0     NetBEUI              0     TCP/IP
   3     IPX/SPX <==========> 1     IPX/SPX
                              3     NetBEUI
```

The above diagram illustrates several other important points about lanas. First, a transport that has a certain lana number on one host does not necessarily have the same lana number on other machines. Second, lana numbers do not have to be sequential.

The EnumProtocols() function can be used to enumerate valid lana numbers. Listed below is a code fragment that demonstrates this type of functionality:

```
#include <nspapi.h>

      DWORD          cb = 0;
      PROTOCOL_INFO *pPI;
      BOOL           pfLanas[100];

      int            iRes,
                     nLanas = sizeof(pfLanas) / sizeof(BOOL);

      // Specify NULL for lpiProtocols to enumerate all protocols.

      // First, determine the output buffer size.
      iRes = EnumProtocols( NULL, NULL, &cb );

      // Verify the expected error was received.
      // The following code must appear on one line.
      assert( iRes == -1 && GetLastError() ==
                  ERROR_INSUFFICIENT_BUFFER );

      if (!cb)
      {
         fprintf( stderr, "No available NetBIOS transports.\n");
         break;
      }

      // Allocate a buffer of the specified size.
      pPI = (PROTOCOL_INFO*) malloc( cb );

      // Enumerate all protocols.
      iRes = EnumProtocols( NULL, pPI, &cb );
```

```
         // EnumProtocols() lists each lana number twice, once for
         // SOCK_DGRAM and once for SOCK_SEQPACKET. Set a flag in pfLanas
         // so unique lanas can be identified.

         memset( pfLanas, 0, sizeof( pfLanas ));

         while (iRes > 0)
            // Scan protocols looking for AF_NETBIOS
            if ( pPI[--iRes].iAddressFamily == AF_NETBIOS )
               // found one.
               pfLanas[ pPI[iRes].iProtocol ] = TRUE;

         fprintf( stderr, "Available NetBIOS lana numbers: " );
         while( nLanas-- )
            if ( pfLanas[nLanas] )
               fprintf( stderr, "%d ", nLanas );

         free( pPI );
      }
```

6. Performance: Use of AF_NETBIOS is recommended for communication with down-level clients.

   On Windows NT, NetBIOS is a high level emulated interface. Consequently, applications that use the WinSock() function over NetBIOS obtain lower throughput than applications that use WinSock() over a native transport such as IPX/SPX or TCP/IP. However, due to the simplicity of the WinSock interface, it is a desirable interface for writing new 32-bit applications that communicate with NetBIOS applications running on down-level clients like Windows for Workgroups or Novell Netware.

Additional reference words: 3.50
KBCategory: kbnetwork kbcode
KBSubcategory: NtwkWinsock

Top-Level Menu Items in Owner-Draw Menus

PSS ID Number: Q69969
Authored 07-Mar-1991 Last modified 15-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

Microsoft Windows version 3.0 allows an application to specify owner-draw menus. This provides the application with complete control over the appearance of items in the menu. However, Windows 3.0 only supports owner-draw items in a pop-up menu. Top-level menu items with the MF_OWNERDRAW style do not work properly.

In Windows 3.1, Windows NT, and Windows 95, top-level menu items with the MF_OWNERDRAW style work properly.

MORE INFORMATION

An application may append an item with the MF_OWNERDRAW style to a top-level menu. At this point, the parent application should receive a WM_MEASUREITEM message and a WM_DRAWITEM message.

However, a WM_MEASUREITEM message is never sent to the parent window for the menu item. In addition, the WM_DRAWITEM message is sent only when the selection state of the item changes (the action field in the DRAWITEMSTRUCT is equal to ODA_SELECTED). The WM_DRAWITEM message is not sent with the action field in the DRAWITEMSTRUCT equal to ODA_DRAWENTIRE.

Additional reference words: 3.00 3.10 3.50 4.00
KBCategory: kbprg
KBSubcategory: UsrMen

Tracking Brush Origins in a Win32-based Application

PSS ID Number: Q102353
Authored 03-Aug-1993 Last modified 23-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

When writing to the Win32 API on Windows NT, it is no longer necessary to keep track of brush origins yourself. GDI32 keeps track of the brush origins by automatically recognizing when the origin has been changed. In Windows, you have to explicitly tell GDI to recognize the change by calling UnrealizeObject() with a handle to the brush. When a handle to a brush is passed to UnrealizeObject() in Windows NT, the function does nothing, therefore, it is still safe to call this API.

Win32s and Windows 95 require that you track the brush origins yourself, just as Windows 3.x does.

In Windows 3.1, the default brush origin is the screen origin. In Windows NT, the default origin is the client origin.

Additional reference words: 3.10 3.50 4.00
KBCategory: kbprg
KBSubcategory: GdiPnbr

Translating Client Coordinates to Screen Coordinates

PSS ID Number: Q11570
Authored 23-Oct-1987 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

The GetClientRect function always returns the coordinates (0, 0) for the origin of a window. This behavior is documented in the "Microsoft Windows Software Development Kit (SDK) Programmer's Reference" manual.

MORE INFORMATION

To determine the screen coordinates for the client area of a window, call the ClientToScreen function to translate the client coordinates returned by GetClientRect into screen coordinates. The following code demonstrates how to use the two functions together:

```
   RECT rMyRect;

   GetClientRect(hwnd, (LPRECT)&rMyRect);
   ClientToScreen(hwnd, (LPPOINT)&rMyRect.left);
   ClientToScreen(hwnd, (LPPOINT)&rMyRect.right);
```

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrWndw

Translating Dialog-Box Size Units to Screen Units

PSS ID Number: Q74280
Authored 15-Jul-1991 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

In the Microsoft Windows graphical environment, the MapDialogRect function converts dialog-box units to screen units easily.

MORE INFORMATION

When a application dynamically adds a child window to a dialog box, it may be necessary to align the new control with other controls that were defined in the dialog box's resource template in the RC file. Because the dialog box template defines the size and position of a controls in dialog-box units rather than in screen units (pixels), the application must translate dialog-box units to screen units to align the new child window.

An application can use the following two methods to translate dialog-box units to screen units:

1. The MapDialogRect function provides the easier method. This function converts dialog-box units to screen units automatically.

   For more details on this method, please see the documentation for the MapDialogRect function in the Microsoft Windows Software Development Kit (SDK).

2. Use the GetDialogBaseUnits function to retrieve the size of the dialog base units in pixels. A dialog unit in the x direction is one-fourth of the width that GetDialogBaseUnits returns. A dialog unit in the y direction is one-eighth of the height that the function returns.

   For more details on this method, see the documentation for the GetDialogBaseUnits function in the Windows SDK.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDlgs

Transparent Blts in Windows NT

PSS ID Number: Q89375
Authored 21-Sep-1992 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, and 3.51

SUMMARY

In order to perform a transparent blt in Microsoft Windows versions 3.0 and 3.1, the BitBlt() function must be called two or more times. This process involves nine steps. (For more information on this process, see article Q79212 in the Microsoft Knowledge Base.)

Windows NT introduces a new method of achieving transparent blts. This method involves the use of the MaskBlt() function. The MaskBlt() function lets you use any two arbitrary ROP3 codes (say, SRCCOPY and BLACKNESS) and apply them on a pel-by-pel basis using a mask.

MORE INFORMATION

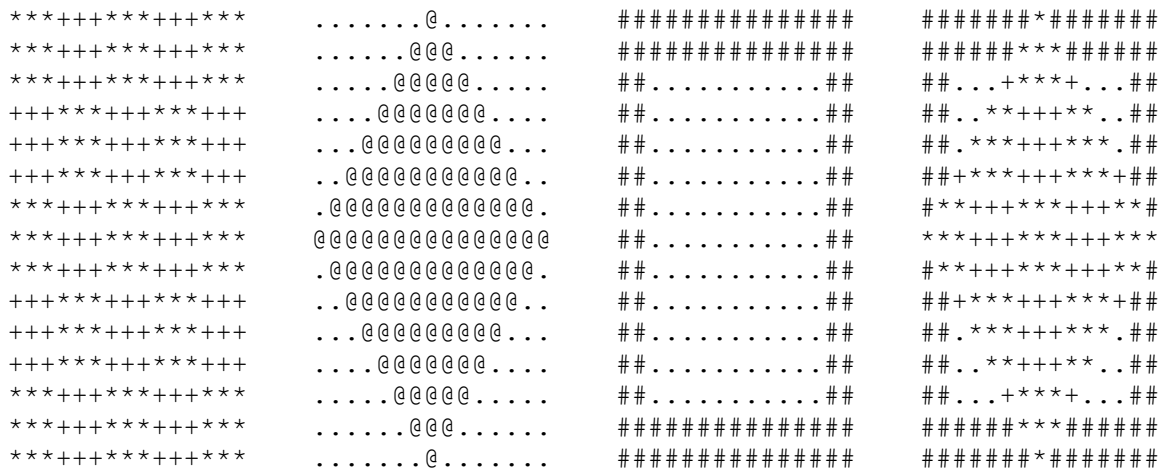
For this example, the source and target bitmaps contain 4 BPP. The call to the MaskBlt() function is as follows:

```
   MaskBlt(hdcTrg,  // handle of target DC
                0,  // x coord, upper-left corner of target rectangle
                0,  // y coord, upper-left corner of target rectangle
               15,  // width of source and target rectangles
               15,  // height of source and target rectangles
           hdcSrc,  // handle of source DC
                0,  // x coord, upper-left corner of source rectangle
                0,  // y coord, upper-left corner of source rectangle
          hbmMask,  // handle of monochrome bit-mask
                0,  // x coord, upper-left corner of mask rectangle
                0,  // y coord, upper-left corner of mask rectangle
       0xAACC0020   // raster-operation (ROP) code
                );
```

The legend is as follows

```
   '.' = 0,
   '@' = 1,
   '+' = 2,
   '*' = 3,
   '#' = 15
```

```
Source Bitmap      Mask Bitmap        Target Bitmap      Result
-------------------------------------------------------------------------
```



Note that the ROP "AA" is applied where 0 bits are in the mask and the ROP "CC" is applied where 1 bit is in the mask. This a transparency.

When creating a ROP4, you can use the following macro:

```
#define ROP4(fore,back) ((((back) << 8) & 0xFF000000) | (fore))
```

This macro can be used to call the MaskBlt() function as follows:

```
MaskBlt(hdcDest, xTrgt, yTrgt,
                        cx, cy,
            hdcSrc, xSrc, ySrc,
          hbmMask, xMask, yMask,
       ROP4(PATCOPY, NOTSRCCOPY)
                             );
```

This call would draw the selected brush where 1 bit appears in the mask and bitwise negation of the source bitmap where 0 bits appear in the mask.

Additional reference words: 3.10 3.50 pixel
KBCategory: kbprg
KBSubcategory: GdiBmp

Transparent Windows

PSS ID Number: Q92526
Authored 09-Nov-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

Microsoft Windows version 3.1 does not support fully functional transparent windows.

MORE INFORMATION

If a window is created using CreateWindowEx() with the WS_EX_TRANSPARENT style, windows below it at the position where the original window was initially placed are not obscured and show through. Moving the WS_EX_TRANSPARENT window, however, results in the old window background moving to the new position, because Windows does not support fully functional transparent windows.

WS_EX_TRANSPARENT was designed to be used in very modal situations and the lifetime of a window with this style must be very short. A good use of this style is for drawing tracking points on the top of another window. For example, a dialog editor would use it to draw tracking points around the control that is being selected or moved.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrWndw

Trapping Floating-Point Exceptions in a Win32-based App

PSS ID Number: Q94998
Authored 28-Jan-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The _controlfp() function is the portable equivalent to the _control87() function.

To trap floating-point (FP) exceptions via try-except (such as EXCEPTION_FLT_OVERFLOW), insert the following before doing FP operations:

```
// Get the default control word.
int cw = _controlfp( 0,0 );

// Set the exception bits ON.
cw &=~(EM_OVERFLOW|EM_UNDERFLOW|EM_INEXACT|EM_ZERODIVIDE|EM_DENORMAL);

// Set the control word.
_controlfp( cw, MCW_EM );
```

This turns on all possible FP exceptions. To trap only particular exceptions, choose only the flags that pertain to the exceptions desired.

Note that any handler for FP errors should have _clearfp() as its first FP instruction.

MORE INFORMATION

By default, Windows NT has all the FP exceptions turned off, and thus computations result in NAN or INFINITY rather than an exception. Note, however, that if an exception occurs and an explicit handler does not exist for it, the default exception handler will terminate the process.

If you want to determine which mask bits are set and which are not during exception handling, you need to use _clearfp() to clear the floating-point exception. This routine returns the existing FP status word, giving the necessary information about the exception. After this, it is safe to query the chip for the state of its control word with _controlfp(). However, as long as an unmasked FP exception is active, most FP instructions will fault, including the fstcw in _controlfp().

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseExcept

Treeviews Share Image Lists by Default

PSS ID Number: Q131287
Authored 07-Jun-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 3.51, 4.0
- Microsoft Win32s version 1.3

SUMMARY

The LVS_SHAREIMAGELISTS style is available for listview controls to enable the same image lists to be used with multiple listview controls. There is no equivalent TVS_SHAREIMAGELISTS, however, for the treeview control because treeviews share image lists by default.

MORE INFORMATION

By default, a listview control takes ownership of the image lists associated with it. For listviews, this could be any of three image lists (those with large icons, small icons, or state images). These three image lists are set by calling ListView_SetImageList() with the flags LVSIL_NORMAL, LVSIL_SMALL, or LVSIL_STATE respectively. The listview then takes responsibility for destroying these image lists when it is destroyed.

Setting the LVS_SHAREIMAGELISTS style, however, moves ownership of the image lists from the listview control to the application. Because this style assumes that the image lists are shared by multiple listviews, specifying this style requires that the application destroy the image lists when the last listview using them is destroyed. Failure to do so causes a memory leak in the system.

Similarly, because treeviews share image lists by default, an application that associates an image list with a treeview should ensure that the image list is destroyed after the last treeview using it is destroyed. Another way to do this is to first set the treeview's image list to NULL by using TreeView_SetImageList(); then destroy the handle to the previous image list returned by the function.

Additional reference words: 1.30 4.00 95 Common Controls
KBCategory: kbprg
KBSubcategory: UsrCtl

Troubleshooting Win32s Installation Problems

PSS ID Number: Q106715
Authored 14-Nov-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.1, 1.15, and 1.2

SUMMARY

The installation guide for Win32s that is included in the Win32 SDK recommends running Freecell to verify that the installation was successful. This article discusses some of the errors that may occur when trying to run Freecell on an unsuccessful install. The article also contains a list of corrective actions to help you reinstall Win32s.

MORE INFORMATION

Possible symptoms

One of the following may occur when running Freecell if the installation is not successful:

- File Error: Cannot find OLECLI.DLL

  -or-

- Win32s - Error:
  Improper installation. Win32s requires WIN32S.EXE and WIN32S16.DLL to run. Reinstall Win32s.

  -or-

- Win32s - Error:
  Improper installation. Windows requires w32s.386 in order to run. Reinstall Win32s.

  -or-

- Error: Cannot find file freecell.exe (or one of its components)...

  -or-

- The display is corrupted as soon as you run FreeCell.

Possible solutions

Check the following when Win32s does not install correctly:

- If you are having video problems, check to see if you have an S3 video card. Certain S3 drivers do not work with Win32s. Either use the generic drivers shipped with Windows or contact your video card manufacturer for an updated driver. For additional information on the S3 driver and Win32s please see the following article(s) in the Microsoft Knowledge Base:

  ARTICLE-ID: Q117153
  TITLE : PRB: Display Problems with Win32s and the S3 Driver

- Make sure that the following line is in your SYSTEM.INI file

  device=*vmcpd

- If you have a printer driver by LaserMaster, delete it or comment it out because it interferes with installing Win32s. Then reboot the computer so that the changes you made will take effect. After you successfully reinstall Win32s, reinstall the driver or remove the comment characters.

  The driver interferes with installing Win32s because the LaserMaster drivers create a WINSPOOL device . The extension is ignored when the filename portion of a path matches a device name. As a result, when Setup tries to write to WINSPOOL.DRV, it fails, because it attempts to write to WINSPOOL. In fact, any Win32-based application that tries to link to WINSPOOL.DRV also fails; however, most Win32-based applications that print under Win32s do not use the WINSPOOL application programming interfaces(APIs) because they are not supported in Win32s. As a result, you can usually just disable this driver while installing Win32s and then reenable it afterwards.

- Delete the \WIN32S directory, the \WIN32APP directory, W32SYS.DLL, W32S16.DLL, and WIN32S.EXE from your hard drive before installing. Although it is possible to install Win32s on top of an old installation of Win32s, it is better to remove the old files before installing the new ones.

  Edit the WIN32S.INI file on your hard drive. Change the line(s) SETUP = 1 to read SETUP = 0. Reboot your computer and reinstall Win32s.

- Make sure that paging is enabled. From the Control Panel, select the 386 Enhanced icon, choose Virtual Memory, and choose Change. Verify that the drive type is not set to none. The type can be set to either temporary or permanent.

- If you are using SHARE, not VSHARE.386 (which WFW machines use), make sure that SHARE is enabled. Edit AUTOEXEC.BAT and add the following line if it is not already there:

  C:\DOS\SHARE.EXE

If you still receive errors when running Freecell, compare the binaries on your hard drive with those on the CD. Use the MS-DOS program FILECOMP.EXE, COMP.EXE, or FC.EXE and do a binary compare. For example, if you have the Win32 SDK CD, type the following:

```
   fc /b <system>\win32s\w32s.386 <cd>:\mstools\win32s\nodebug\w32s.386
```

If you have the 32-bit Visual C++ CD, type the following:

```
   fc /b <system>\win32s\w32s.386 <cd>:\msvc32s\win32s\retail\w32s.386
```

The results of the compare will be, "FC: no differences encountered", if the binaries were correctly installed. If the binaries are not the same, you might have a bad copy of the files or a bad CD.

As a last resort, you can try reinstalling Win32s that shipped in the Visual C++ 32-bit edition. There seems to be a slight difference between the SETUP.EXE program on the Win32 SDK CD and the SETUP.EXE on the Visual C++ CD. If you try installing from the 32-bit Visual C++ CD, you should NOT remove the \WIN32APP or \FREECELL directory or any of the Freecell files. The Visual C++ CD does not contain Freecell.

NOTE: Using Freecell Help will generate errors, including "Routine Not Found" or "Help Topic Does Not Exist." The generation of these errors has nothing to do with whether or not Win32s is installed correctly.

FREECELL.HLP was meant to be used with WINHLP32.EXE. FREECELL.HLP uses the advanced features of WINHLP32 for full-text searching. WINHELP.EXE, which runs on Windows 3.1, does not support this. As a result, each time FREECELL.HLP tries to bind the Find button to the full-text searching APIs, it fails, and Windows Help displays the message box. You can still read the information in the help file. You can use the Search button to do keyword searches.

Additional reference words: 1.10
KBCategory: kbsetup
KBSubcategory: W32s

TrueType Font Converters and Editors

PSS ID Number: Q87817
Authored 06-Aug-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The text below lists a number of commercial software tools that convert existing fonts to TrueType fonts or help build new fonts. The tools are listed in alphabetical order by name. None of these tools is recommended over any other, nor over any that are absent from the list. Each of the tools has its own strengths and weaknesses. Before making any purchase, examine the tool and its documentation to see if it meets your needs. This list will be updated as more tools become available.

Some of the following tools run on an Apple Macintosh while others run on an IBM PC/AT or compatible computer. The same TrueType font can run on a Macintosh or with Microsoft Windows operating system version 3.1.

To convert a font from the Macintosh to Windows 3.1, save the data fork from a Macintosh font to a file, copy the file to an MS-DOS-formatted disk, give the file a .TTF file extension, and use the Windows 3.1 Control Panel to install the font.

The products included here are manufactured by vendors independent of Microsoft; we make no warranty, implied or otherwise, regarding these products' performance or reliability.

MORE INFORMATION

Last Update: 24 July 1992

FONT CONVERSION UTILITIES

AllType for MS-DOS

Author: Atech Software
5964 La Place Court, Suite 125
Carlsbad, CA 92008
(619) 438-6883
Description: Character-based application running under MS-DOS. Converts almost any font format to any other. Supported formats include TrueType, Type-1, Type-3, Nimbus-Q, and

Intellifont.

Evolution 2.0 for Macintosh

Author: Image Club Graphics, Inc.
1902 11th St. SE, Suite 5
Calgary, Alberta, Canada T2G 3G2
(403) 262-8008
Description: Converts almost any font format to any other. Supported formats include TrueType, Type-1, and Type-3.

FontMonger for Windows
FontMonger for Macintosh

Author: Ares Software
561 Pilgrim Drive, Suite D
Foster City, CA 94404
(415) 578-9090
Description: Converts almost any font format to any other. Supported formats include TrueType, Type-1, Type-3, and Intellifont. Also provides minor font editing by creating composite characters or rearranging the characters in a font.

Incubator for Windows

Author: Type Solutions, Inc.
91 Plaistow Rd
Plaistow, NH 03865
(603) 382-6400
Description: Supports adding effects to TrueType fonts. Contact Type Solutions for more information.

Metamorphosis Professional for Macintosh

Author: Altsys Corp.
269 W. Renner Rd
Richardson, TX 75080
(214) 680-2060
Description: Converts almost any font format to any other. Supported formats include TrueType, Type-1, Type-3, and PICT format. One interesting feature allows the user to read a Type-1 font from the ROM of an Apple LaserWriter printer and convert the font to another format.

FONT EDITORS

Fontographer 3.5 for Windows
Fontographer 4.1 for Macintosh

Author: Altsys Corp.
269 W. Renner Rd
Richardson, TX 75080
(214) 680-2060
Description: A complete font editing tool. Supports creating a font from scratch and modifying existing fonts. Supports a variety of formats including TrueType and Type-1. Includes an autohinter.

FontStudio 2.0 for Macintosh

Author: Letraset Graphic Design Software
40 Eisenhower Dr
Paramus, NJ 07653
(800) 343-TYPE or (201) 845-6100
Description: A complete font editing tool. Supports creating fonts from scratch and modifying existing fonts. Supports a variety of formats including TrueType and Type-1. Includes an autohinter.

TypeMan 1.0 for Macintosh

Author: Type Solutions, Inc.
91 Plaistow Rd
Plaistow, NH 03865
(603) 382-6400
Description: Designed for font foundries, this tool provides precise control over the hints in a font and includes an autohinter. Supports specifying a font in a high-level programming language, which the tool compiles to the binary TrueType format.

Additional reference words: 3.10 3.50 4.00 95 true type
KBCategory: kbprg
KBSubcategory: GdiTt

Trusted DDE Shares

PSS ID Number: Q128125
Authored 27-Mar-1995 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), version 3.5

SUMMARY

To allow someone else to connect to DDE shares when you are logged in, you have to trust your existing DDE share. The reason is that when the other person connects to the share remotely, the application he will connect to is running in your security context, not the remote user's, because you are the logged-on user. You need to give permission for the other person to access the share. Even another person who is an administrator cannot trust a share for your account.

In your code, you would use NDdeShareAdd() to create the share and NDdeSetTrustedShare() to trust the share.

Alternatively, you can use DDESHARE to create the share. For more information on DDESHARE, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID: Q114089
TITLE : Using the Windows NT NetDDE Share Manager

If you have a DDE share that always needs to be available, you can write a program that trusts the specific shares and sets up a logon script to trust the share for every user. The logon script should be a .BAT file that calls the .EXE file, so that you can add other things to the logon script as necessary.

MORE INFORMATION

The online documentation for NDdeSetTrustedShare says:

> The NDdeSetTrustedShare function is called to promote the referenced DDE share to trusted status within the current user's context.

DDE shares are a machine resource, not an account resource, just as shared drives are. However, NetDDE runs an application that must run in the context of the current user. This is the reason that the share must be trusted, so that the application can run in the user's context.

The prototype for the function is:

```
UINT NDdeSetTrustedShare(lpszServer, lpszShareName, dwTrustOptions)
```

The parameter lpszServer is the address of the server name on which the DDE share resides. The DDE Share Database (DSDM) will be modified. This service manages the shared DDE conversations and is used by the NetDDE service. This parameter will generally be the current machine, because you can't trust a share for someone else.

The dwTrustOptions are NDDE_TRUST_SHARE_START and NDDE_TRUST_SHARE_INIT. NDDE_TRUST_SHARE_START allows the DDE server, such as Excel, to be started in the user's context. This allows a DDE client to make a NetDDE connection without the DDE server already running. When the NetDDE agent on the server machine detects the attempted connection, it launches the associated DDE server application if it is not already running.

NDDE_TRUST_SHARE_INIT allows a client to initiate to the DDE server if it is already executing in the user's context. This allows a DDE client to make a NetDDE connection to a DDE server already running on the server machine. If the DDE server is not already running, the connection will fail.

Additional reference words: 3.50
KBCategory: kbprg
KBSubcategory: UsrNetDde

Types of File I/O Under Win32

PSS ID Number: Q99173
Authored 24-May-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

There are multiple types of file handles that can be opened using the Win32 API and the C Run-time:

Returned Type	File Creation API	API Set
HANDLE	CreateFile()	Win32
HFILE	OpenFile()/_lcreat()	Win32
int	_creat()/_open()	C Run-time
FILE *	fopen()	C Run-time

In general, these file I/O "families" are incompatible with each other. On some implementations of the Win32 application programming interfaces (APIs), the OpenFile()/_lcreat() family of file I/O APIs are implemented as "wrappers" around the CreateFile() family of file I/O APIs, meaning that OpenFile(), _lcreat(), and _lopen() end up calling CreateFile(), returning the handle returned by CreateFile(), and do not maintain any state information about the file themselves. However, this is an implementation detail only and is NOT a design feature.

NOTE: You cannot count on this being true on other implementations of the Win32 APIs. Win32 file I/O APIs may be written using different methods on other platforms, so reliance on this implementation detail may cause your application to fail.

The rule to follow is to use one family of file I/O APIs and stick with them--do not open a file with _lopen() and read from it with ReadFile(), for example. This kind of incorrect use of the file I/O APIs can easily be caught by the compiler, because the file types (HFILE and HANDLE respectively) are incompatible with each other and the compiler will warn you (at warning level /w3 or higher) when you have incorrectly passed one type of file handle to a file I/O API that is expecting another, such as passing an HFILE type to ReadFile(HANDLE, ...) in the above example.

MORE INFORMATION

Compatibility

The OpenFile() family of file I/O functions is provided only for compatibility with earlier versions of Windows. New Win32-based

applications should use the CreateFile() family of file I/O APIs, which provide added functionality that the earlier file I/O APIs do not provide.

Each of the two families of C Run-time file I/O APIs are incompatible with any of the other file I/O families. It is incorrect to open a file handle with one of the C Run-time file I/O APIs and operate on that file handle with any other family of file I/O APIs, nor can a C Run-time file I/O family operate on file handles opened by any other file I/O family.

_get_osfhandle()

For the C Run-time unbuffered I/O family of APIs [_open(), and so forth], it is possible to extract the operating system handle that is associated with that C run-time handle via the _get_osfhandle() C Run-time API. The operating system handle is the handle stored in a C Run-time internal structure associated with that C Run-time file handle. This operating system handle is the handle that is returned from an operating system call made by the C Run-time to open a file [CreateFile() in this case] when you call one of the C Run-time unbuffered I/O APIs [_open(), _creat(), _sopen(), and so forth].

The _get_osfhandle() C Run-time call is provided for informational purposes only. Problems may occur if you read or write to the file using the operating system handle returned from _get_osfhandle(); for these reasons we recommend that you do not use the returned handle to read or write to the file.

_open_osfhandle()

It is also possible to construct a C Run-time unbuffered file I/O handle from an operating system handle [a CreateFile() handle] with the _open_osfhandle() C Run-time API. In this case, the C Run-time uses the existing operating system handle that you pass in rather than opening the file itself. It is possible to use the original operating system handle to read or write to the file, but it is very important that you use only the original handle or the returned C Run-time handle to access the file, but not both, because the C Run-time maintains state information that will not be updated if you use the operating system handle to read or write to the file.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseFileio

Types of Thunking Available in Win32 Platforms

PSS ID Number: Q125710
Authored 02-Feb-1995 Last modified 14-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)
- Microsoft Win32s, version 1.3

SUMMARY

Thunks allow code on one side of the 16-32 process boundary to call into code on the other side of the boundary. Each Win32 platform employs one or more thunking mechanisms. This table summarizes the thunking mechanisms provided by the different Win32 platforms.

	Win32s	Windows 95	Windows NT
Generic Thunk		X	X
Universal Thunk	X		
Flat Thunk		X	

Generic Thunks allow a 16-bit Windows-based application to load and call a Win32-based DLL on Windows NT and Windows 95.

Windows 95 also supports a thunk compiler, so a Win32-based application can load and call a 16-bit DLL

Win32s Universal Thunks allow a Win32-based application running under Win32s to load and call a 16-bit DLL. You can also use UT to allow a 16-bit Windows-based application to call a 32-bit DLL under Win32s, but this isn't officially supported. Certain things do not work on the 32-bit side because the app was loaded with the context of a 16-bit Windows-based application.

This article describes the types of thunking mechanisms available on each Win32 platform.

MORE INFORMATION

Windows NT

Windows NT supports Generic Thunks, which allow 16-bit code to call into 32-bit code. Generic Thunks do not allow 32-bit code to call into 16-bit code. The generic thunk is implemented by using a set of API functions that are exported by the WOW KERNEL and WOW32.DLL.

In Windows NT, 16-bit Windows-based applications are executed in a

subsystem (or environment) called WOW (Windows On Win32). Each application runs as a thread in a VDM (virtual DOS machine).

Using generic thunks is like explicitly loading a DLL. The four major APIs used in generic thunking are: LoadLibraryEx32W(), FreeLibrary32W(), GetProcAddress32W(), and CallProc32W(). Their functionality is very similar to LoadLibraryEx(), FreeLibrary(), GetProcAddress(), and calling the function through a function pointer. The Win32-based DLL called by the thunk is loaded into the VDM address space. The following is a example of thunking a call to GetVersionEx():

```
   void FAR PASCAL __export MyGetVersionEx(OSVERSIONINFO *lpVersionInfo)
   {
      HINSTANCE32 hKernel32;
      FARPROC lpGetVersionEx;

      // Load KERNEL32.DLL
      if (!(hKernel32 = LoadLibraryEx32W("KERNEL32.DLL", NULL, NULL)))
      {
         MessageBox(NULL, "LoadLibraryEx32W Failed", "DLL16", MB_OK);
         return;
      }

      // Get the address of GetVersionExA in KERNEL32.DLL
      if (!(lpGetVersionEx =
            GetProcAddress32W( hKernel32, "GetVersionExA")))
      {
         MessageBox(NULL, "GetProcAddress32W Failed", "DLL16", MB_OK);
         return;
      }
      lpVersionInfo->dwOSVersionInfoSize = sizeof(OSVERSIONINFO);

      // Call GetVersionExA
      CallProc32W(lpVersionInfo, lpGetVersionEx, 1, 1);

      // Free KERNEL32.DLL
      if (!FreeLibrary32W(hKernel32))
      {
         MessageBox(NULL, "FreeLibrary32W Failed", "DLL16", MB_OK);
         return;
      }
      return;
   }
```

Win32s

All 16-bit Windows-based and Win32-based applications run in a single address space in Win32s. The mechanism that is provided for accessing 16-bit code from 32-bit code is called the Universal Thunk. The Universal Thunks consists of 4 APIs. The major APIs, UTRegister() and UTUnRegister(), are exported by KERNEL32. The prototype for UTRegister() is:

```
   BOOL UTRegister(HANDLE hModule,        // Win32-based DLL handle
                   LPCTSTR lpsz16BITDLL,  // 16-bit DLL to call
```

```
                    LPCTSTR lpszInitName,  // thunk initialization procedure
                    LPCTSTR lpszProcName,    // thunk procedure
                    UT32PROC *ppfn32Thunk,   // pointer to thunk procedure
                    FARPROC pfnUT32CallBack, // optional callback
                    LPVOID lpBuff);          // shared memory buffer
```

NOTES: lpszInitName, pfnUT32CallBack, and lpBuff are optional parameters. The value for ppfn32Thunk is the returned value of the 32-bit function pointer to the thunk procedure. The buffer lpBuff is a globally allocated shared memory buffer that is available to the 16-bit initialization routine via a 16-bit selelctor:offset pointer.

The function pointer returned in ppfn32Thunk has the following syntax:

```
   WORD (*ppfn32Thunk)(lpBuff, dwUserDefined, *lpTranslationList);
```

where lpBuff is the pointer to the shared data area, dwUserDefined is available for application use (it is most commonly used as a switch for multiple thunked functions), and lpTranslationList is an array of flat pointers within lpBuff that are to be translated into selector:offset pointers.

This method is not portable to other platforms.

Windows 95

Thunking in Windows 95 allows 16-bit code to call 32-bit code and vice-versa. The mechanism used is a thunk compiler. To use the thunk compiler you need to create a thunk script, which is the function prototype with additional information about input and output variables.

The thunk compiler produces a single assembly language file. This single assembly language file should be assembled using two different flags - DIS_32 and -DIS_16 to produce a 16-bit and 32-bit object files. These object modules should be liked to their respective 16-bit and 32-bit DLL's. There are no special APIs used, all you have to do is call the function.

In addition to Flat Thunks, Windows 95 supports the Windows NT Generic Thunk mechanism. Generic thunks are recommended for portability between Windows 95 and Windows NT.

REFERENCES

For more information on Generic Thunks, see GENTHUNK.TXT on the Win32 SDK CD.

For more information on the Universal Thunk, see the "Win32s Programmer's Reference" and the UTSAMP sample on the Win32 SDK CD.

Additional reference words: 1.30 3.50 4.00
KBCategory: kbprg
KBSubcategory: SubSys BseMisc W32s

Understanding Win16Mutex

PSS ID Number: Q125867
Authored 07-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

SUMMARY

Windows 95 offers preemptive multithreaded scheduling for Win32 processes, yet also provides the familiar non-preemptive task switching found in Windows 3.x for Win16 applications. Some of the Win32 system DLLs, such as USER32.DLL and GDI32.DLL, thunk to their 16-bit counterparts for compatibility and size. Unlike Windows 3.x, which did not preemptively schedule processes, Windows 95 must be able to handle the possibility that it might be reentered by two or more processes each calling the same API functions. Windows 95 must do so in a way that uses little memory and is compatible with all existing 16-bit applications and DLLs.

Win16Mutex is a global semaphore that is used to protect the 16-bit Windows 95 components from being reentered by preventing Win32 threads from thunking to 16-bit components while other 16-bit code is running. Win16Mutex is internal to Windows 95 and is not accessible from applications or DLLs. This article explains how Win16Mutex works and offers design tips for minimizing the effects Win16Mutex may have on Win32 applications.

MORE INFORMATION

Because Windows 3.x is a non-preemptive system, Windows 3.x did not need to be designed to prevent the system from being reentered. Only one task (application instance) at a time can call system services (API functions) because other tasks cannot run until the active task voluntarily yields control of the CPU. Since only one task can execute at a time, it is not possible to have two different tasks calling the same API function, and thus Windows does not need to protect itself from reentrancy.

Windows 95 differs from Windows 3.x because Windows 95 provides support for both Win32 and Win16 applications. In Windows 95, every instance of every Win16 application is a process with exactly one thread, and every Win32 process has at least one thread. Win32 threads are preemptively scheduled and may even preempt Win16 processes. Because many Win32 API functions are thunked to 16-bit Windows API functions, there is now a possibility for the 16-bit Windows components to be reentered. Since the 16-bit Windows components are largely the same as in Windows 3.x, they need to be protected from being reentered.

The Win16Mutex provides this protection by allowing only one thread (not process) at a time to access the 16-bit APIs. Whenever Win16Mutex is owned

by a thread, any other thread that tries to claim Win16Mutex will block until Win16Mutex is released. Now the question remains: "When does Win16Mutex get claimed and released?"

Whenever a Win16 process is running, it owns Win16Mutex. That is, when a Win16 process first gets a message via GetMessage or PeekMessage, the Win16 process claims Win16Mutex. The Win16 process releases Win16Mutex whenever the process yields, such as when the process calls GetMessage or PeekMessage and doesn't return. The only way a Win16 process can keep Win16Mutex indefinitely is to never yield; since the message-processing mechanism provides the scheduler in 16-bit Windows, the only way to never yield is to stop processing messages (which makes the application unresponsive to user input).

The only time a thread in a Win32 process claims Win16Mutex is when the thread makes a call to an API function which thunks to one of the 16-bit Windows components or when the thread thunks directly to a Win16 DLL. Immediately after the call returns, the process releases Win16Mutex. Not all API functions thunk to 16-bit components; most 32-bit USER and GDI functions thunk to 16-bit USER and GDI, but none of the 32-bit KERNEL functions thunk to 16-bit KRNL386. One exception is when a Win32 process spawns a Win16 process, KERNEL32 thunks to KRNL386 to call the Win16 loader.

Putting 16-bit and 32-bit behaviors together, you can see that when a Win16 process is running, and a thread of a Win32 process preempts the Win16 process and calls a function which thunks to a 16-bit component, the Win32 thread is put to sleep until the Win16 process yields, which releases Win16Mutex. Likewise, when one Win32 process's thread claims Win16Mutex and then loses its timeslice, and another thread from either the same or a different process tries to claim Win16Mutex, the second thread blocks until Win16Mutex is released.

Win16Mutex is internal to Windows 95 and may not be manipulated or even checked by applications and DLLs. Win16Mutex is implemented mainly in the thunk layer so that every Win32 API which thunks to a Win16 component will automatically claim Win16Mutex before entering 16-bit code. Additionally, thunks created by the thunk compiler to allow Win32 applications and DLLs to call 16-bit DLLs claim Win16Mutex automatically, so programmers do not have to do so explicitly.

One way to lessen the impact that Win16 applications have on the responsiveness of Win32 applications is to create multiple threads where the primary thread (the initial thread of the process) controls the entire user interface for the process and each additional thread performs some useful task, such as reading or writing to a data file, but does not make user interface or graphics calls. This way, if the Win32 process's primary thread blocks waiting for a Win16 process to yield, its other threads are still performing useful work.

Additional reference words: 4.00
KBCategory: kbprg
KBSubcategory: BseProcThread

UNICODE and _UNICODE Needed to Compile for Unicode

PSS ID Number: Q99359
Authored 26-May-1993 Last modified 18-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

To compile code for Unicode, you need to #define UNICODE for the Win32 header files and #define _UNICODE for the C Run-time header files. These #defines must appear before the

```
#include <windows.h>
```

and any included C Run-time headers. The leading underscore indicates deviance from the ANSI C standard. Because the Windows header files are not part of this standard, it is allowable to use UNICODE without the leading underscore.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: UsrNls

Unicode Conversion to Integers

PSS ID Number: Q89295
Authored 16-Sep-1992 Last modified 24-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

Under Windows NT, strings may be either Unicode or ANSI. There is no function for reliably converting a string that might be either Unicode or ANSI to an integer. This is because, given a random set of bytes, it is difficult to determine whether the string is in Unicode or ANSI. The calling program has to know which format the string uses in order to convert it.

If the string uses Unicode, the functions wcstol(), wcstoul(), and wcstod() can be used to perform the conversion.

Note that when you are using the Win32 application programming interface (API), you can choose what kind of characters you get from the console or window manager. The names of the API functions that are called to use Unicode and ANSI characters are different. For more details, see Chapter 93 in the overview, "Unicode."

To mark a string as Unicode, insert the byte-ordering-mark (BOM) 0xFEFF in the string and/or file.

MORE INFORMATION

You can assume that the first 128 bytes in each character set are in the same codepoint. For portability, you should code character conversions in this range as:

```
{
   TCHAR     c;
     ...
     i = c - TEXT('0');
}
```

The TEXT macro places an "L" before the constant if Unicode is defined.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: WIntlDev

Unicode Functions Supported by Windows 95

PSS ID Number: Q125671
Authored 01-Feb-1995 Last modified 15-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

Unlike Windows NT, Windows 95 does not implement the Unicode (or wide character) version of most Win32 functions that take string parameters. With some exceptions, these functions are implemented as stubs that simply return an error value.

In general, Windows 95 implements the ANSI version of these functions. See the Win32 API documentation for information on particular functions and differences between the various Win32 platforms.

One major exception to this rule is OLE. All native 32-bit OLE APIs and interface methods use Unicode exclusively. For more information on this, please see the OLE documentation.

Excluding OLE, Windows 95 supports the following wide character functions:

```
EnumResourceLanguages
EnumResourceNames
EnumResourceTypes
ExtTextOut
FindResource
FindResourceEx
GetCharWidth
GetCommandLine
GetTextExtentExPoint
GetTextExtentPoint32
GetTextExtentPoint
lstrlen
MessageBoxEx
MessageBox
TextOut
```

In addition, Windows 95 implements the following two functions for converting strings to or from Unicode:

```
MultiByteToWideChar
WideCharToMultiByte
```

Additional reference words: 4.00
KBCategory: kbprg
KBSubcategory: WIntlDev

Unicode Implementation in Windows NT 3.1 and 3.5

PSS ID Number: Q103977
Authored 02-Sep-1993 Last modified 27-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.1 and 3.5

SUMMARY

Windows NT is the first widely available operating system to be built upon the Unicode character encoding. Almost all of the strings used in the system have 16-bits reserved for each character. However, Windows NT does not yet realize the Unicode ideal of offering an editor capable of handling one document containing all of the languages of the world.

MORE INFORMATION

Unicode support in Windows NT:

- All Windows USER objects support Unicode strings.
- The Win32 console is Unicode enabled.
- NTFS supports Unicode filenames.
- All of the information strings in the registry are Unicode.
- The L_10646.TTF (Lucida Sans Unicode) font covers over 1300 Unicode characters.
- Most of the TrueType fonts include a Unicode encoding table.

Unicode features missing from Windows NT:

- There is no font support for all of the Unicode characters.
- Although the Win32 console is Unicode enabled, it is not possible to use Unicode fonts in the console. Most Unicode characters will be represented by the "default character" of the System font.
- Winhlp32 is not Unicode enabled.
- There is no general Unicode input method in Windows NT version 3.1. The shell applets and File Manager fully support Unicode. You can use the new Notepad and Character Mapper applets to create files with Unicode text. (Choose the Lucida Sans Unicode font in the Character Mapper, then choose the desired Unicode characters in the Character Mapper and copy them to the clipboard. Paste the clipboard contents into Notepad, making sure Notepad has the Lucida Sans Unicode font selected, and save the file as a "Unicode Files". Note, this same process can be used to give files Unicode filenames.)
- The FAT and HPFS file systems do not support Unicode filenames. (Nor will they in the future; to accomplish this, use NTFS.)

Additional reference words: 3.10 3.50
KBCategory: kbprg

KBSubcategory: UsrNls

Uniqueness Values in User and GDI Handles

PSS ID Number: Q94917
Authored 25-Jan-1993 Last modified 21-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

User and GDI handles, are divided into two parts:

- The lower 16 bits is an index into a system table of handle structures, which includes information such as the type of handle (window, menu, cursor, and so forth), as well as a value called the uniqueness.

- The upper 16 bits contain the same uniqueness value.

The first time a handle is issued by the system, the uniqueness value is 0 (zero). It is incremented each time the handle is re-used. In Windows NT 3.1, if you pass in a value of 0xFFFF for the uniqueness, the client side (that is, USER32.DLL) will look up the correct uniqueness value in shared memory and use the correct handle. In Windows NT 3.5, use 0x0000 for the uniqueness value.

This is important because it alleviates potential conflicts with re-used handles. For example, when a window is destroyed, its handle is reused by the system. The uniqueness value prevents an old handle to a destroyed window from being misinterpreted by the system as the handle to a new object, which was given the same handle value.

MORE INFORMATION

In Win32s, use 0x0000 in the upper 16 bits for the uniqueness.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: UsrMisc

Use 16-Bit .FON Files for Cross-Platform Compatibility

PSS ID Number: Q100487
Authored 22-Jun-1993 Last modified 17-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

The AddFontResource() function installs a font resource in the GDI font table. Under Windows NT and Windows 95, the module can be a .FON file or a .FNT file. Under Windows 3.1, the module must be a .FON file. When using Win32s, AddFontResource() passes its argument to the Win16 AddFontResource, and therefore .FON files should be used for portability.

In addition, when running under Windows NT or Windows 95, the module can be either a 32-bit "portable executable" or a 16-bit .FON file. However, if the same Win32 executable is run under Win32s, the call to AddFontResource() fails if the .FON is not in 16-bit format. Therefore, for compatibility across platforms, use 16-bit .FON files. These can be created using the Windows 3.1 Software Development Kit (SDK).

Additional reference words: 3.10 3.50 4.00
KBCategory: kbprg
KBSubcategory: GdiFnt

Use LoadLibrary() on .EXE Files Only for Resources

PSS ID Number: Q108448
Authored 12-Dec-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

The LoadLibrary() application programming interface (API) can be used to load an executable module. A common use of this function is to load a dynamic-link library (DLL), perform a subsequent call to GetProcAddress() to get the address of an exported DLL routine, and call this DLL routine through the address that is returned. Another use of LoadLibrary() is to load an executable module and retrieve its resources.

In Windows NT, a LoadLibrary() of an .EXE file is supported only for the purposes of retrieving resources. It was decided that it was rather uncommon to load an .EXE for any other purpose, so this limitation was imposed on LoadLibrary() to improve the performance in loading resources. Calling a routine in an .EXE through an address obtained with GetProcAddress() can cause an access violation.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseMisc

Use MoveWindow to Move an Iconic MDI Child and Its Title

PSS ID Number: Q70079
Authored 09-Mar-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, version 3.5

Although Windows does not have application programming interface (API) calls to support dragging iconic windows (windows represented by an icon), calls to MoveWindow() can be used to implement a dragging functionality for iconic multiple-document interface (MDI) child windows.

However, the icon's title is not in the same window as the icon, so when the icon is moved, the title will not automatically move with it. The title is in a small window of its own, so a separate call to MoveWindow() must be made to place the title window correctly below the icon.

The information below describes how to get the window handle for the small title window, so that the appropriate call to MoveWindow() can be made.

The icon's title window is a window of class #32772. This window is a child of the MDI client window, and its owner is the icon window.

To get the window handle to the appropriate icon title window for a given icon, enumerate all the children of the MDI client window, looking for a window whose owner is the icon.

You can use EnumChildWindows() to loop through all the children of the MDI client window.

You can use GetWindow(..., GW_OWNER) to check the parent of each window.

Additional reference words: 3.00 3.10 3.50
KBCategory: kbprg
KBSubcategory: UsrMdi

Use of Allocations w/ cbClsExtra & cbWndExtra in Windows

PSS ID Number: Q11606
Authored 23-Oct-1987 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

The following is an explanation of the use of the allocations with cbClsExtra and cbWndExtra?:

1. cbClsExtra -- extra bytes to allocate to CLASS data structure in USER.EXE local heap when RegisterClass() is called. Accessed by Get/Set CLASS Word/Long ();.

2. cbWndExtra -- extra bytes to allocate to WND data structure in USER.EXE local heap when CreateWindow() is called. Accessed by Get/Set WND Word/Long ();.

You can use these structures at your discretion and for any purpose you desire.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCls

Use of DLGINCLUDE in Resource Files

PSS ID Number: Q91697
Authored 02-Nov-1992 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The dialog editor needs a way to know what include file is associated with a resource file that it opens. Rather than prompt the user for the name of the include file, the name of the include file is embedded in the resource file in most cases.

MORE INFORMATION

Embedding the name of the include file is done with a resource of type RCDATA with the special name DLGINCLUDE. This resource is placed into the .RES file and contains the name of the include file. The dialog editor looks for this resource when it loads a .RES file. If this resource is found, then the include file is opened also; if not, the editor prompts the user for the name of the include file.

In some Windows 3.1 build environments, the dialog editor was used to create dialogs that were placed in more than one .DLG file. These different .DLG files were then included in one .RC file, which was compiled with the resource compiler. Therefore, the resource file gets multiple copies of a RCDATA type resource with the same name, DLGINCLUDE, but the resource compiler and dialog editor don't complain.

In the Win32 SDK, changes were made so that this resource has its own resource type; it was changed from an RCDATA-type resource with the special name, DLGINCLUDE, to a DLGINCLUDE resource type whose name can be specified. The dialog editor would look for resources of the type DLGINCLUDE.

We are being more strict about the need for resources to be unique in the Win32 SDK than the Windows 3.1 SDK. This is good because there was never any guarantee at run time as to which of the two or more resources would be returned by LoadResource().

This means that some applications being ported to Windows NT give an error when their resources are compiled because they have duplicate RCDATA type resources with the same name (DLGINCLUDE). This error is by design. The workaround is straightforward: delete all the DLGINCLUDE RCDATA type resource statements from all the .DLG files.

Finally, because it does not make sense to have the DLGINCLUDE type resources in the executable, the linker will strip them out so that they

don't get linked into the EXE.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsDlg

Use of DocumentProperties() vs. ExtDeviceMode()

PSS ID Number: Q92514
Authored 08-Nov-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

Windows-based applications have used ExtDeviceMode() to retrieve or modify device initialization information for printer drivers. The Win32 API introduces a new function DocumentProperties() that applications can use to configure the settings of the printer.

Note that ExtDeviceMode() calls DocumentProperties(); therefore, it is faster for applications to use DocumentProperties() directly.

Specifying the DM_UPDATE mask allows an application to change printer settings when using DocumentProperties(). Applications should be aware that the GetProcAddress() function is now case sensitive.

Windows-based applications running on Windows NT (WOW) can call ExtDeviceMode(). The spooler's ExtDeviceMode() entry is intended for WOW use.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiPrn

Use of NULL_PEN, NULL_BRUSH, and HOLLOW_BRUSH

PSS ID Number: Q66532
Authored 30-Oct-1990 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

GDI contains several "NULL" stock objects: NULL_BRUSH, HOLLOW_BRUSH, and NULL_PEN. These objects are defined in WINDOWS.H (16-bit SDK) or in WINGDI.H (32-bit SDK). These header files define HOLLOW_BRUSH as NULL_BRUSH, so they are the same objects.

Note that NULL_BRUSH and NULL_PEN are NOT identical to the value NULL. The value NULL is defined as 0 (zero) in WINDOWS.H and is not a valid stock object.

Many GDI functions use the current brush to fill interiors and the current pen to draw lines. In some cases, an application may not want to modify the areas normally affected by the pen or brush. Selecting a NULL_PEN or NULL_BRUSH into the device context tells GDI not to modify the normally affected areas. In short, "NULL_" objects do not draw anything.

For example, the Rectangle() function uses the current brush to fill the interior of the rectangle and the current pen to draw the border. If NULL_PEN is selected into the device context, no border is drawn. If NULL_BRUSH or HOLLOW_BRUSH is selected, the interior of the rectangle is not painted. If both NULL_PEN and NULL_BRUSH are selected, the rectangle will not be drawn.

Additional reference words: 3.00 3.10 3.50 4.00 95 hollow
KBCategory: kbprg
KBSubcategory: GdiPnbr

Use of Polygon() Versus PolyPolygon()

PSS ID Number: Q119164
Authored 09-Aug-1994 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

Polygon() draws a polygon, while PolyPolygon() draws a series of polygons. Using multiple calls to Polygon() can offer better performance than using a single call to PolyPolygon(); this is because PolyPolygon() does not consider the polygons to be independent, but considers them to be one polygon with multiple disjointed edges. However, there are times when PolyPolygon() is preferable, particularly if the number of polygons is small.

MORE INFORMATION

PolyPolygon() batches polygons in a single call, so there is less call overhead than there is for multiple calls to Polygon(). However, to perform one combined fill, PolyPolygon() has to work with all the edges in all of the polygons simultaneously, resulting in sorting overhead. The overhead involved in sorting becomes quite expensive when there are a lot of polygons, causing a net loss of performance in comparison to Polygon().

GDI batches multiple Polygon() calls to be more efficient. Setting the batch limit higher than the default of 10 with GdiSetBatchLimit() improves performance even further. GDI and some drivers optimize convex polygons, but will only optimize a single polygon drawn with either Polygon() or PolyPolygon().

Because PolyPolygon() treats all edges as part of one big polygon, it also draws every pixel to be filled exactly once; this may be a performance advantage if a lot of overlapping polygons are drawn, because Polygon() draws every pixel in each polygon only once, even where there is an overlap.

PolyPolygon() considers all the polygons when applying the current fill mode, as set by calling SetPolyFillMode(). Consequently, if any polygons overlap, the result of one PolyPolygon() call may be different than the result of the equivalent multiple Polygon() calls. If the polygons overlap and the raster operation takes the destination pixel values into account, or if you want the fill rule to be applied to overlapping areas, then it is preferable to use PolyPolygon().

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: GdiDrw

Use Uppercase "K" for Keywords in Windows Help Files

PSS ID Number: Q64050
Authored 20-Jul-1990 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The standard keyword list must be defined by using capital "K" footnotes. Lowercase "k" footnotes may not be used for defining either standard or alternate keyword lists.

MORE INFORMATION

Using lowercase "k" footnotes can result in problems such as the following:

If your application calls WinHelp() using the HELP_KEY option for doing a keyword search and you pass a LPSTR to a keyword defined in a footnote attached to your topic, the Help system displays an "Invalid key word" error message box. For example

```
WinHelp(hWnd,cFileDir,HELP_KEY,(DWORD)(LPSTR)"help");
```

where

```
hWnd     is the handle of the calling window.
cFileDir is the directory path and filename of the .HLP file.
"help"   is the keyword defined in the footnote section of the topic.
```

and the footnote section of the topic is as follows:

```
k sample;help
```

Modifying the footnote for the topic to use an uppercase "K" solves the problem.

```
K sample;help
```

Additional reference words: 3.00 3.10 3.50 4.00 95 key word
KBCategory: kbtool
KBSubcategory: TlsHlp

Using #include Directive with Windows Resource Compiler

PSS ID Number: Q80945
Authored 20-Feb-1992 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The Windows Resource Compiler supports many standard C language preprocessor directives such as "#define" to define symbolic constants, and "#include" to include header and other resource files.

If an application developed for the Windows environment has more than one resource, each resource can be maintained in a separate file. Then, use the #include directive to direct the Resource Compiler to build all the resources into one output file. Using this technique prevents one resource file from becoming unmanageably large with an overwhelming number of resources.

It is important to note that the Resource Compiler treats files with the .C and .H extensions in a special manner. It assumes that a file with one of these two extensions does not contain resources. When a file has the .C or .H file extension, the Resource Compiler ignores all lines in the file except for preprocessor directives (#define, #include, and so forth). Therefore, a file that contains resources that is included in another resource file should not have the .C or .H file extension.

MORE INFORMATION

The following example demonstrates the implications of this situation. The MSG.H file has the following contents:

```
/*
 * This header file defines message IDs for strings in the
 * stringtable resource. All source files can use this header file
 * to reference specific strings.
 */
#define STRING1   1
#define STRING2   2
#define STRING3   3
```

The STRTABLE.RC file has the following contents:

```
/*
 * This file defines the stringtable resource contents for this
 * application. It should be included in the application's resource
```

```
    * file. It requires definitions from the MSG.H header file.
    */
   STRINGTABLE
      {
      STRING1, "This is string 1."
      STRING2, "This is string 2."
      STRING3, "This is string 3."
      }
```

The APP.RC file has the following contents:

```
   /*
    * This is the "main" resource definition file for this
    * application. Among other things, it includes the stringtable
    * resource definition from other header files.
    */

   RESOURCE 1 (MENU)
   RESOURCE 2 (RAW DATA)
   ...

   #include "MSG.H"
   #include "STRTABLE.RC"
```

The Resource Compiler treats both MSG.H and STRTABLE.RC as header files. MSG.H does not include any resources; therefore, it can use the standard .H file extension. However, because STRTABLE.RC includes a resource (a STRINGTABLE), it cannot be named with a .C or .H file extension.

Files that contain resources can have any legal MS-DOS file extension other than .C and .H.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsRc

Using a Dialog Box as the Main Window of an Application

PSS ID Number: Q108936
Authored 20-Dec-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

Dialog boxes might be used as the main window of an application for several reasons. When an application uses a dialog box as the main window, the following should be taken into consideration while designing such an application:

- The dialog box that acts as the main window of an application can be created without an owner.

- If a modal dialog box is created as a main window, TranslateAccelerator() cannot be used.

- The icon for the dialog box should be drawn manually when the dialog box is minimized.

MORE INFORMATION

You can create a modal or modeless dialog box as the main window of an application. In doing so, there is no need to have an overlapped window that acts as the owner of the dialog box. Memory for edit controls created with the DS_LOCALEDIT flag set, and static controls, will come from the heap represented by the hInstance passed to the CreateDialog() or DialogBox() call.

When modal dialog boxes are chosen for this purpose, and accelerator keys are defined in an application, Windows enters a modal message loop that does not process accelerators, unless a WH_MSGFILTER hook is installed and TranslateAccelerator() is called from the hook callback function.

One easy way to avoid this limitation is to create a modeless dialog box and call TranslateAccelerator() from the main message loop.

The icon for a window is stored in its class information. Because a dialog box is a window of global class that all applications in the system are using, changing the icon for this application will change the icon for all dialog boxes in the system. Below is one workaround for drawing the icon manually when the dialog box is minimized:

Sample Code

```
BOOL WINAPI GenericDlgProc (HWND hwnd, UINT msg,
                            WPARAM wParam,  LPARAM lParam)
 {

   RECT rect ;
   switch (msg) {

       case WM_INITDIALOG:
        hIcon = LoadIcon(); // Load the icon that is to be displayed
                            // when minimized.
        return TRUE ;

       case WM_ERASEBKGND:
            if (IsIconic(hwnd) && hIcon) {
              SendMessage( hwnd, WM_ICONERASEBKGND, wParam, 0L );
              return TRUE;
             }
        break;

       case WM_QUERYDRAGICON:
            return (hIcon);

        case WM_PAINT: {
            PAINTSTRUCT ps;

            BeginPaint( hwnd, &ps );

            if (IsIconic(hwnd))   //*** If iconic, paint the icon.
            {
              if (hIcon) {

                //center the icon correctly..
                GetClientRect(hwnd, &rect) ;
                rect.left = (rect.right - GetSystemMetrics(SM_CXICON)) >> 1
;
               rect.top = (rect.bottom - GetSystemMetrics(SM_CYICON)) >> 1
;
                DrawIcon( ps.hdc, rect.left, rect.top, hIcon );
            }
            EndPaint( hwnd, &ps );
        }
        break;
    }
    return FALSE;

} //*** GenericDlgProc
```

The workaround described above assumes that the dialog box belongs to the predefined dialog class. For private dialog box classes, there is no need to manually draw the icon for the dialog box when it is minimized. You can specify the icon while registering the dialog box class. Remember to set the cbWndExtra field to DLGWINDOWEXTRA. When the dialog box is minimized, the icon will be painted automatically.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95 DLGMAIN

KBCategory: kbprg
KBSubcategory: UsrDlgs

Using a Fixed-Pitch Font in a Dialog Box

PSS ID Number: Q77991
Authored 31-Oct-1991 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

To use a fixed-pitch font in a dialog box, during the processing of the dialog box initialization message, send the WM_SETFONT message to each control that will use the fixed font. The following code demonstrates this process:

```
   case WM_INITDIALOG:
     SendDlgItemMessage(hDlg, ID_CONTROL, WM_SETFONT,
           GetStockObject(ANSI_FIXED_FONT), FALSE);
     /*
      * NOTE: This code will specify the fixed font only for the
      * control ID_CONTROL. To specify the fixed font for other
      * controls in the dialog box, additional calls to
      * SendDlgItemMessage() are required.
      */
     break;
```

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDlgs

Using a Modeless Dialog Box with No Dialog Function

PSS ID Number: Q72136
Authored 15-May-1991 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

When creating a modeless dialog box with the default dialog class, an application normally passes a procedure-instance address of the dialog function to the CreateDialog() function. This dialog function processes messages such as WM_INITDIALOG and WM_COMMAND, returning TRUE or FALSE.

It is acceptable to create a modeless dialog box that uses NULL for the lpDialogFunc parameter of CreateDialog(). This type of dialog box is useful when the no controls or other input facilities are required. In this case, using NULL simplifies the programming.

However, the dialog box must be closed through some means other than a push button (for example, via a timer event).

NOTE: A modal dialog box that does not provide a means of closing itself will hang its parent application because control will never return from the DialogBox() function call.

If lpDialogFunc is NULL, no WM_INITDIALOG message will be sent, and DefDlgProc() does not attempt to call a dialog function. Instead, DefDlgProc() handles all messages for the dialog. The application that created the modeless dialog must explicitly call DestroyWindow() to free its system resources.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDlgs

Using a Mouse with MEP Under Windows NT

PSS ID Number: Q83300
Authored 08-Apr-1992 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

The Microsoft Editor (MEP) included with the Win32 Software Development Kit (SDK) can be used with a mouse to position the cursor.

By default, the mouse is not enabled for MEP. It is necessary to add the switch "usemouse:yes" (without the quotation marks) to the TOOLS.INI file under the [m mep] section. The TOOLS.INI file is a text file editable by MEP or Notepad.

To change the position of the cursor, first position the mouse pointer on the new location and then click the left mouse button.

Additional reference words: 3.10 3.50
KBCategory: kbtool
KBSubcategory: TlsMep

Using Built-In Printing Features from a Rich Edit Control

PSS ID Number: Q129860
Authored 08-May-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.51 and 4.0
- Microsoft Win32s, version 1.3

SUMMARY

The Rich Edit control contains built-in printing features that can be used to send formatted text to the printer with minimal effort from the programmer.

MORE INFORMATION

Printing from a Rich Edit control involves the use of the standard printing APIs and two Rich Edit control messages, EM_FORMATRANGE and EM_DISPLAYBAND. The EM_FORMATRANGE message can be used by itself or used in combination with the EM_DISPLAYBAND message. Included below at the end of this article is a code sample which demonstrates the usage of these messages.

EM_FORMATRANGE

This message is used to format the text for the printer DC and can optionally send the output to the printer.

The wParam parameter for this message is a Boolean value that indicates whether or not the text should be rendered (printed) to the printer. A zero value only formats the text, while a nonzero value formats the text and renders it to the printer.

The lParam parameter for this message is a pointer to the FORMATRANGE structure. This structure needs to be filled out before sending the message to the control.

FORMATRANGE Members

HDC hdc - Contains the device context (DC) to render to if the wParam parameter is nonzero. The output is actually sent to this DC.

HDC hdcTarget - Contains the device context to format for, which is usually the same as the hdc member but can be different. For example, if you create a print preview module, the hdc member is the DC of the window in which the output is viewed, and the hdcTarget member is the DC for the printer.

RECT rc - Contains the area to render to. This member contains the

rectangle that the text is formatted to fit in, and subsequently printed in. It also contains the margins, room for headers and footers, and so forth. The rc.bottom member may be changed after the message is sent. If it is changed, it must indicate the largest rectangle that can fit within the bounds of the original rectangle and still contain the specified text without printing partial lines. It may be necessary to reset this value after each page is printed. These dimensions are given in TWIPS.

RECT rcPage - Contains the entire area of the rendering device. This area can be obtained using the GetDeviceCaps() function. These dimensions are given in TWIPS.

CHARRANGE   chrg - Contains the range of characters to be printed. Set chrg.cpMin to 0 and chrg.cpMax to -1 to print all characters.

The return value from EM_FORMATRANGE is the index of the first character on the next page. If you are printing multiple pages, you should set chrg.cpMin to this value before the next EM_FORMATRANGE message is sent.

When printing is complete, this message must be sent to the control with wParam = 0 and lParam = NULL to free the information cache by the control.

EM_DISPLAYBAND

If you use 0 for the wParam parameter in the EM_FORMATRANGE message, then you can use the EM_DISPLAYBAND message to send the output to the printer.

The wParam parameter for this message is not used and should be 0.

The lParam parameter for this message is a pointer to a RECT structure. This RECT structure is the area to display to and is usually the same as the rc member of the FORMATRANGE structure used in the EM_FORMATRANGE message but can be different. For example, the rectangles are not the same if you are printing on a certain portion of a page or built-in margins are being used.

This message should only be used after a previous EM_FORMATRANGE message.

Sample Code

```
void Print(HDC hPrinterDC, HWND hRTFWnd)
   {
   FORMATRANGE fr;
   int         nHorizRes = GetDeviceCaps(hPrinterDC, HORZRES),
               nVertRes = GetDeviceCaps(hPrinterDC, VERTRES),
               nLogPixelsX = GetDeviceCaps(hPrinterDC, LOGPIXELSX),
               nLogPixelsY = GetDeviceCaps(hPrinterDC, LOGPIXELSY);
   LONG        lTextLength;   // Length of document.
   LONG        lTextPrinted;  // Amount of document printed.

   // Ensure the printer DC is in MM_TEXT mode.
   SetMapMode ( hPrinterDC, MM_TEXT );
```

```
   // Rendering to the same DC we are measuring.
   ZeroMemory(&fr, sizeof(fr));
   fr.hdc = fr.hdcTarget = hPrinterDC;

   // Set up the page.
   fr.rcPage.left     = fr.rcPage.top = 0;
   fr.rcPage.right    = (nHorizRes/nLogPixelsX) * 1440;
   fr.rcPage.bottom   = (nVertRes/nLogPixelsY) * 1440;

   // Set up 1" margins all around.
   fr.rc.left   = fr.rcPage.left + 1440;  // 1440 TWIPS = 1 inch.
   fr.rc.top    = fr.rcPage.top + 1440;
   fr.rc.right  = fr.rcPage.right - 1440;
   fr.rc.bottom = fr.rcPage.bottom - 1440;

   // Default the range of text to print as the entire document.
   fr.chrg.cpMin = 0;
   fr.chrg.cpMax = -1;

   // Set up the print job (standard printing stuff here).
   ZeroMemory(&di, sizeof(di));
   di.cbSize = sizeof(DOCINFO);
   if (*szFileName)
      di.lpszDocName = szFileName;
   else
      {
      di.lpszDocName = "(Untitled)";

      // Do not print to file.
      di.lpszOutput = NULL;
      }

   // Start the document.
   StartDoc(hPrinterDC, &di);

   // Find out real size of document in characters.
   lTextLength = SendMessage ( hRTFWnd, WM_GETTEXTLENGTH, 0, 0 );

   do
      {
      // Start the page.
      StartPage(hPrinterDC);

      // Print as much text as can fit on a page. The return value is the
      // index of the first character on the next page. Using TRUE for the
      // wParam parameter causes the text to be printed.

#ifdef USE_BANDING

      lTextPrinted = SendMessage(hRTFWnd,
                                 EM_FORMATRANGE,
                                 FALSE,
                                 (LPARAM)&fr);
      SendMessage(hRTFWnd, EM_DISPLAYBAND, 0, (LPARAM)&fr.rc);
```

```
#else

      lTextPrinted = SendMessage(hRTFWnd,
                                 EM_FORMATRANGE,
                                 TRUE,
                                 (LPARAM)&fr);

#endif

      // Print last page.
      EndPage(hPrinterDC);

      // If there is more text to print, adjust the range of characters to
      // start printing at the first character of the next page.
      if (lTextPrinted < lTextLength)
         {
         fr.chrg.cpMin = lTextPrinted;
         fr.chrg.cpMax = -1;
         }
      }
   while (lTextPrinted < lTextLength);

   // Tell the control to release cached information.
   SendMessage(hRTFWnd, EM_FORMATRANGE, 0, (LPARAM)NULL);

   EndDoc (hPrinterDC);
   }
```

Additional reference words: 1.30 4.00 95
KBCategory: kbprg kbui
KBSubcategory: UsrCtl W32s

Using cChildren Member of TV_ITEM to Add Speed & Use Less RAM

PSS ID Number: Q131278
Authored 07-Jun-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.51, 4.0
- Microsoft Win32s version 1.3

SUMMARY

The cChildren member of the TV_ITEM structure is used to denote the number of child items associated with a treeview item. When used correctly, the cChildren member helps an application go faster and use less memory.

Applications may specify I_CHILDRENCALLBACK for this member, instead of passing the actual number of child items. To retrieve the actual number of child items associated with a treeview item, I_CHILDRENCALLBACK causes the treeview to send a TVN_GETDISPINFO notification when the item needs to be redrawn.

MORE INFORMATION

In a treeview with the TVS_HASBUTTONS style, specify a nonzero value for the cChildren member to add the appropriate plus or minus (+/-) button (to denote expand or collapse) to the left of the treeview item, without having to insert each of the child items to the treeview. Specifying a zero value indicates that the particular item does not have any child items associated with it.

An application that does not use the cChildren member of the TV_ITEM structure when inserting items to the treeview has to insert each of the child items associated with that treeview item in order for the +/- button to show up.

Using the cChildren member, therefore, helps to speed up an application and reduce memory requirements by allowing the application to fill the tree on demand.

Applications such as the Explorer, WinHelp, and RegEdit, which display huge hierarchical structures, take advantage of this feature by initially inserting only the visible items of the tree. The child items associated with a particular item are not inserted until the user clicks the parent item to expand it. At that point, the treeview's parent window receives a TVN_ITEMEXPANDING notification message, and the application inserts the child items for that parent item:

```
   // WM_NOTIFY message handler
   LRESULT MsgNotifyTreeView(HWND hwnd,
                             UINT uMessage,
                             WPARAM wparam,
```

```
                            LPARAM lparam)
{

    LPNMHDR  lpnmhdr = (LPNMHDR)lparam;

    // Just before the parent item gets EXPANDED,
    // add the children.

    if (lpnmhdr->code == TVN_ITEMEXPANDING)
    {
        LPNM_TREEVIEW lpNMTreeView;
        TV_ITEM   tvi;

        lpNMTreeView = (LPNM_TREEVIEW)lparam;
        tvi = lpNMTreeView->itemNew;

        if ((tvi.lParam == PARENT_NODE) &&
            (lpNMTreeView->action == TVE_EXPAND))
        {

            // Fill in the TV_ITEM struct
            // and call TreeView_InsertItem() for each child item.

        }
     }

     return DefWindowProc(hwnd, uMessage, wparam, lparam);
}
```

When the user clicks the same parent item to collapse it, the application removes all the child items from the tree by using TreeView_Expand (,,TVE_COLLAPSE | TVE_COLLAPSERESET). The TVN_ITEMEXPANDED notification message is a good place to do this:

```
// WM_NOTIFY message handler
LRESULT MsgNotifyTreeView(HWND hwnd,
                          UINT uMessage,
                          WPARAM wparam,
                          LPARAM lparam)
{

    LPNMHDR  lpnmhdr = (LPNMHDR)lparam;

    // Just before the parent item is COLLAPSED,
    // remove the children.

    if (lpnmhdr->code == TVN_ITEMEXPANDED)
    {

        LPNM_TREEVIEW lpNMTreeView;
        TV_ITEM       tvi2;


        lpNMTreeView = (LPNM_TREEVIEW)lparam;
        tvi2 = lpNMTreeView->itemNew;
```

```
            // Do a TVE_COLLAPSERESET on the parent to minimize memory use.

            if ((lpNMTreeView->action == TVE_COLLAPSE) &&
                (tvi2.lParam == PARENT_NODE))
            {
                 TreeView_Expand (ghWndTreeView,
                                  tvi2.hItem,
                                  TVE_COLLAPSE | TVE_COLLAPSERESET);
            }
         }
         return DefWindowProc(hwnd, uMessage, wparam, lparam);
      }
```

Applications that dynamically change the number of child items associated with a particular treeview item may specify I_CHILDRENCALLBACK for the cChildren member of its TV_ITEM structure when it is inserted into the tree. Thereafter, when that treeview item needs to be redrawn, the treeview sends a TVN_GETDISPINFO to its parent window to retrieve the actual number of child items and display the +/- button to the left of the treeview item, as appropriate.

An application that uses I_CHILDRENCALLBACK may process the TVN_GETDISPINFO notification as follows:

```
   // WM_NOTIFY message handler
   LRESULT MsgNotifyTreeView(HWND hwnd,
                             UINT uMessage,
                             WPARAM wparam,
                             LPARAM lparam)
   {

       LPNMHDR  lpnmhdr = (LPNMHDR)lparam;

       if (lpnmhdr->code == TVN_GETDISPINFO)
       {
          TV_DISPINFO FAR *lptvdi;

          lptvdi = (TV_DISPINFO FAR *)lparam;

       if ((lptvdi->item.mask & TVIF_CHILDREN) &&
(lptvdi->item.lParam == PARENT_NODE))
          lptvdi->item.cChildren = 1;

       }

      return DefWindowProc(hwnd, uMessage, wparam, lparam);
   }
```

Additional reference words: 4.00 95 Common Control performance speed up
KBCategory: kbprg kbcode
KBSubcategory: UsrCtl

Using Device Contexts Across Threads

PSS ID Number: Q94236
Authored 30-Dec-1992 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, and 3.51

A window created with the CS_OWNDC style retains its device context (DC) attributes across GetDC() calls.

However, the DC attributes are not retained if the GetDC() calls are called from different threads. This is by design because DCs are thread-based. In the Win32 user interface, if the calling thread is not the owner of the window, then GetDC() returns a cache DC instead of the owned DC handle.

To save attributes across threads, one must create a routine to initialize DC attributes, which is then called from threads not owning the given window.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: GdiDc

Using Device-Independent Bitmaps and Palettes

PSS ID Number: Q72041
Authored 11-May-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The method Windows version 3.x uses to transfer colors from the color table of a device-independent bitmap (DIB) to a device-dependent bitmap (DDB) on a machine that supports palette operations depends on the value of the wUsage parameter specified in calls to the CreateDIBitmap or SetDIBits functions.

Specifying DIB_RGB_COLORS matches the colors in the DIB color table to the logical palette associated with the device context (DC) listed in the function call.

Specifying DIB_PAL_COLORS causes the entries in the DIB color table to not be treated as RGB values; instead, they are treated as word indexes into the logical palette associated with the DC listed in the function call.

To create a device-dependent (displayable) bitmap from a DIB that retains the same colors, follow these five steps:

1. Extract the colors from the DIB header.

2. Use the CreatePalette function to make a logical palette with those colors.

3. Use the SelectPalette function to select the logical palette into a device context.

4. Use the RealizePalette function to map the logical palette into the device context.

5. Call the CreateDIBitmap function using the device context that has the logical palette selected.

Because the CreateDIBitmap function creates a bitmap compatible with the device, to create a device-dependent monochrome bitmap from a DIB, call the CreateBitmap function with the desired width and height, specifying one color plane and one bit per pixel. Then call the SetDIBits function to render the image in the newly created monochrome bitmap.

Additional reference words: 3.00 3.10 3.50 4.00 95

KBCategory: kbprg
KBSubcategory: GdiPal

Using Drag-Drop in an Edit Control or a Combo Box

PSS ID Number: Q86724
Authored 15-Jul-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

In the Microsoft Windows environment, an application can register an edit control or a combo box as a drag-drop client through the DragAcceptFiles function. The application must also subclass the control to process the WM_DROPFILES message that Windows sends when the user drops a file.

MORE INFORMATION

The following seven steps demonstrate how to implement drag-drop in an edit control. The procedure to implement drag-drop in a combo box is identical.

1. Add SHELL.LIB to the list of libraries required to build the file.

2. Add the name of the subclass procedure (MyDragDropProc) to the EXPORTS section of the module definition (DEF) file.

3. Include the SHELLAPI.H file in the application's source code.

4. Declare the following procedure and variables:

```
BOOL FAR PASCAL MyDragDropProc(HWND, unsigned, WORD, LONG);

FARPROC lpfnDragDropProc, lpfnOldEditProc;
char    szTemp64[64];
```

5. Add the following code to the initialization of the dialog box:

```
case WM_INITDIALOG:
   // ... other code

   // ------- edit control section --------
   hWndTemp = GetDlgItem(hDlg, IDD_EDITCONTROL);
   DragAcceptFiles(hWndTemp, TRUE);

   // subclass the drag-drop edit control
   lpfnDragDropProc = MakeProcInstance(MyDragDropProc, hInst);

   if (lpfnDragDropProc)
      lpfnOldEditProc = SetWindowLong(hWndTemp, GWL_WNDPROC,
```

```
                (DWORD)(FARPROC)lpfnDragDropProc);
         break;
```

6. Write a subclass window procedure for the edit control.

```
      BOOL FAR PASCAL MyDragDropProc(HWND hWnd, unsigned message,
                                     WORD wParam, LONG lParam)
      {
         int wFilesDropped;

         switch (message)
            {
         case WM_DROPFILES:
            // Retrieve number of files dropped
            // To retrieve all files, set iFile parameter
            // to -1 instead of 0
            wFilesDropped = DragQueryFile((HDROP)wParam, 0,
                  (LPSTR)szTemp64, 63);

            if (wFilesDropped)
               {
               // Parse the file path here, if desired
               SendMessage(hWnd, WM_SETTEXT, 0, (LPSTR)szTemp64);
               }
            else
               MessageBeep(0);

            DragFinish((HDROP)wParam);
            break;

         default:
            return CallWindowProc(lpfnOldEditProc, hWnd, message,
                  wParam, lParam);
            break;
         }
         return TRUE;
      }
```

7. After the completion of the dialog box procedure, free the edit control subclass procedure.

```
      if (lpfnDragDropProc)
         FreeProcInstance(lpfnDragDropProc);
```

Additional reference words: 3.10 3.50 3.51 4.00 95 combobox
KBCategory: kbprg
KBSubcategory: UsrDnd

Using DWL_USER to Access Extra Bytes in a Dialog Box

PSS ID Number: Q88358
Authored 24-Aug-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

Windows extra bytes are used to store private information specific to an instance of a window. For dialog boxes, these extra bytes are already allocated by the dialog manager. The offset to the extra byte location is called DWL_USER.

DWL_USER is the 8th byte offset of the dialog extra bytes. The programmer has 4 bytes (a long) available from this offset for personal use.

CAUTION: DO NOT use more than 4 bytes of these extra bytes, as the rest of them are used by the dialog manager.

Example

```
   DWORD dwNumber = 10;
             .
             .
             .
             .
   case WM_INITDIALOG:
            SetWindowLong(hWnd,DWL_USER,dwNumber); // Store value 10 at
                                                   // byte offset 8
            dwNumber = GetWindowLong(hWnd,DWL_USER); // Retrieve the value
```

NOTE: GetWindowWord and SetWindowWord could be used instead.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDlgs

Using ENTER Key from Edit Controls in a Dialog Box

PSS ID Number: Q102589
Authored 04-Aug-1993 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

Windows-based applications often display data-entry dialog boxes to request information from users. These dialog boxes may contain several edit controls and two command (push) buttons, labeled OK and CANCEL. An example of a data-entry dialog box is one that requests personal information, such as social security number, address, identification (ID) number, date/time, and so on, from users. Each of these items is entered into an edit control.

By default, the TAB key is used in a dialog box to shift focus between edit controls. As a common user-interface, however, one could also use the ENTER (RETURN) key to move between the edit controls (for example, after the user enters a piece of information, pressing ENTER moves the focus to the next field).

There are a few ways to enable the use of the ENTER key to move between edit controls. One method is to make use of WM_COMMAND and the notification messages that come with it in the dialog box for edit controls and buttons. Another method involves subclassing the edit controls. A third involves using App Studio and Class Wizard and creating a new dialog box member function.

MORE INFORMATION

Method I: (WM_COMMAND)

This method is based on the following behavior of dialog boxes (Dialog Manager) and focus handling in Windows.

If a dialog box or one of its controls currently has the input focus, then pressing the ENTER key causes Windows to send a WM_COMMAND message with the idItem (wParam) parameter set to the ID of the default command button. If the dialog box does not have a default command button, then the idItem parameter is set to IDOK by default.

When an application receives the WM_COMMAND message with idItem set to the ID of the default command button, the focus remains with the control that had the focus before the ENTER key was pressed. Calling GetFocus() at this point returns the handle of the control that had the focus before the ENTER key was pressed. The application can check this control handle and determine whether it belongs to any of the

edit controls in the dialog box. If it does, then the user was entering data into one of the edit controls and after doing so, pressed ENTER. At this point, the application can send the WM_NEXTDLGCTL message to the dialog box to move the focus to the next control.

However, if the focus was with one of the command buttons (CANCEL or OK), then GetFocus() returns a button control handle, at which point one can dismiss the dialog box. The pseudo code for this logic resembles the following in the application's dialog box procedure:

```
     case WM_COMMAND:

     if(wParam=IDOFDEFBUTTON || IDOK) {
          // User has hit the ENTER key.

          hwndTest = GetFocus() ;
          retVal = TesthWnd(hWndTest) ;

          //Where retVal is a boolean variable that indicates whether
        //the hwndTest is the handle of one of the edit controls.

          if(hwndTest) {
         //Focus is with an edit control, so do not close the dialog.
            //Move focus to the next control in the dialog.

         PostMessage(hDlg, WM_NEXTDLGCTL, 0, 0L) ;
         return TRUE ;
          }
          else {
            //Focus is with the default button, so close the dialog.
            EndDialog(hDlg, TRUE) ;
            return FALSE ;
          }
        }
     break ;
```

Method II

This method involves subclassing/superclassing the edit control in the dialog box. Once the edit controls are subclassed or superclassed, all keyboard input is sent the subclass/superclass procedure of the edit control that currently has input focus, regardless of whether or not the dialog box has a default command button. The application can trap the key down (or char) messages, look for the ENTER key, and do the processing accordingly. The following is a sample subclass procedure that looks for the ENTER key:

```
//*---------------------------------------------------------------------
//| Title:
//|     SubClassProc
//|
//| Parameters:
//|     hWnd            - Handle to the message's destination window
```

```
//|     wMessage        - Message number of the current message
//|     wParam          - Additional info associated with the message
//|     lParam          - Additional info associated with the message
//|
//| Purpose:
//|     This is the window procedure used to subclass the edit control.
//*---------------------------------------------------------------------

long FAR PASCAL SubProc(HWND hWnd, WORD wMessage,WORD wParam,LONG lParam)
{

    switch (wMessage)
     {

          case WM_GETDLGCODE:
            return (DLGC_WANTALLKEYS |
                    CallWindowProc(lpOldProc, hWnd, wMessage,
                                   wParam, lParam));

          case WM_CHAR:
          //Process this message to avoid message beeps.
         if ((wParam == VK_RETURN) || (wParam == VK_TAB))
           return 0;
            else
           return (CallWindowProc(lpOldProc, hWnd,
                                  wMessage, wParam, lParam));

       case WM_KEYDOWN:
            if ((wParam == VK_RETURN) || (wParam == VK_TAB)) {
              PostMessage (ghDlg, WM_NEXTDLGCTL, 0, 0L);
              return FALSE;
            }

         return (CallWindowProc(lpOldProc, hWnd, wMessage,
                                wParam, lParam));
         break ;

       default:
            break;

     } /* end switch */
```

Method 3

This method involves using App Studio and ClassWizard and creating a new dialog box member function.

This method will allow a user to press the ENTER key and have the focus advance to the next edit control. If the focus is currently on the last edit control in the dialog box, the focus will advance to the first edit control.

First, use App Studio to change the ID of the OK button of the dialog box. The default behavior of App Studio is to give the OK button the ID IDOK.

The OK button's ID should be changed to another value, such as IDC_OK. Also, change the properties of the OK button so that it is not a default pushbutton.

Next, use ClassWizard to create a new dialog box member funciton. Name the new member function something like OnClickedOK. This function should be tied to the BN_CLICKED message from the IDC_OK control.

Once this is done, write the body of the OnClickedOK function. You should put the code that you would normally put in the OnOK function into the new OnClickedOK function, including a class's OnOK function.

Add the following prototype to the header file for the dialog box:

```
   protected:
      virtual void OnOK();
```

Add an OnOK function to the dialog box and code is as demonstrated below:

```
void CMyDialog::OnOK()
{
     CWnd* pwndCtrl = GetFocus();
     CWnd* pwndCtrlNext = pwndCtrl;
     int ctrl_ID = pwndCtrl->GetDlgCtrlID();

     switch (ctrl_ID) {
         case IDC_EDIT1:
             pwndCtrlNext = GetDlgItem(IDC_EDIT2);
             break;
         case IDC_EDIT2:
             pwndCtrlNext = GetDlgItem(IDC_EDIT3);
             break;
         case IDC_EDIT3:
             pwndCtrlNext = GetDlgItem(IDC_EDIT4);
             break;
         case IDC_EDIT4:
             pwndCtrlNext = GetDlgItem(IDC_EDIT1);
             break;
         case IDOK:
             CDialog::OnOK();
             break;
         default:
             break;
     }
     pwndCtrlNext->SetFoucs();
}
```

Additional reference words: 3.10 3.50 3.51 4.00 95 push RETURN keydown
KBCategory: kbprg
KBSubcategory: UsrDlgs

Using Extra Fields in Window Class Structure

PSS ID Number: Q10841
Authored 01-Dec-1987 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

In order to generate several child windows of the same class, each having its own set of static variables and independent of the sets of the variables in the sibling windows, you need to use the cbWndExtra field in WNDCLASS, the window-class data structure, when registering a window; then, use SetWindowWord() (or Long) and GetWindowWord() (or Long). These functions will either get or set additional information about the window identified by hWnd.

Use positive offsets as indexes to access any additional bytes that were allocated when the window class structure was created, starting at zero for the first byte of the extra space. Similarly, if you want to refer to bytes already defined by Windows within the structure, use offsets defined with the GWW and GWL prefixes.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCls

Using FileOpen Common Dialog w/ OFN_ALLOWMULTIPLESELECT Style

PSS ID Number: Q130761
Authored 25-May-1995 Last modified 22-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SUMMARY

This article covers the format of file names returned by the FileOpen common dialog with the OFN_ALLOWMULTIPLESELECT style.

The FileOpen common dialog can be used to specify the location (drive and directory) and name of a file or a set of files. One of the flags needed to provide mulitple-file selection from the FileOpen common dialog is the OFN_ALLOWMULTIPLESELECT flag. When this flag is used, and the user makes a valid selection, the file or files chosen by the user are returned in the lpstrFile member of the OPENFILENAME structure. The format of the string returned in the lpstrFile member depends on how many files (single or multiple) the user selected. This article assumes that the OFN_EXPLORER Style is set for the file open dialog.

MORE INFORMATION

If multiple files were selected, the string is of this form:

```
Drive: \Directory Name\0FileName 1\0FileName 2\0FileName n\0\0.
```

The Directory Name is listed first. Then each file that was selected is listed with a terminating NULL Character, except for the last filename, which is terminated with two NULL characters. The two NULL characters signal the end of the string.

If a single file was selected, the string is of this form:

```
Drive: \Directory Name\FileName\0\0.
```

The Directory Name in this case is not terminated by a NULL character, and the file name is terminated with two NULL characters.

Applications must parse the string returned in the lsptrFile member. In doing so, they should make provisions in the parsing code to have a case where the user can make a single selection (even though the OFN_ALLOWMULTIPLESELECT flag is set) or multiple selections.

NOTE: When using OFN_ALLOWMULTIPLESELECT under Windows 95, you need to use the OFN_EXPLORER flag to get the explorer style dialog and NULL terminated strings. If you don't use OFN_EXPLORER with OFN_ALLOWMULTIPLESELECT, you get the old style dialog and space-delimited strings.

Additional reference words: 4.00
KBCategory: kbprg
KBSubcategory: UsrCmnDlg

Using GDI-Synthesized Italic Fonts

PSS ID Number: Q74467
Authored 21-Jul-1991 Last modified 25-May-1995

The information in this article applies to:

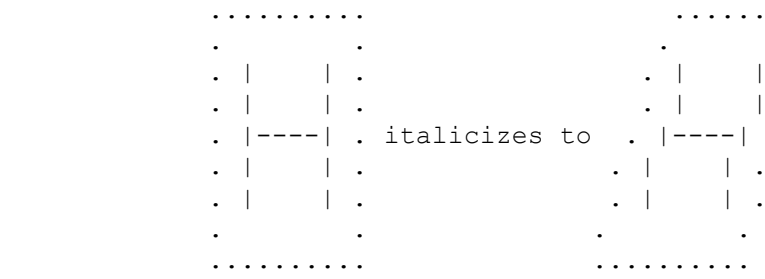
- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

In the Microsoft Windows graphical environment, when an application uses an italic font synthesized by the graphics device interface (GDI), each character and its whole character cell are "sheared," or slanted, to the right, which can cause some unexpected results.

MORE INFORMATION

The capital H in the example below illustrates how GDI synthesizes an italic font:



Note two items in this case:

1. If the text background color is changed so that it does not match the window background color, the text background color occupies the sheared character cell (in other words, it is also slanted). Gaps occur in the background where normal text is adjacent to italic text.

2. The italic character is farther to the right in relation to the lower-left corner of the character cell than is the normal character. Therefore, if normal and italic text start at the same x coordinate on different lines, the italic text appears farther to the right.

To determine the number of units by which the character cell is sheared, call the GetTextMetrics function to fill a TEXTMETRIC data structure with information about the font. The tmOverhang member describes the amount of shear.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiFnt

Using GetDIBits() for Retrieving Bitmap Information

PSS ID Number: Q85846
Authored 22-Jun-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

When saving a bitmap in .DIB file format, the GDI function is used to retrieve the bitmap information. The general use of this function and the techniques for saving a bitmap in .DIB format are largely unchanged; however, this article provides more details on the use of the Win32 API version of the GetDIBits() function.

MORE INFORMATION

The function can be used to retrieve the following information:

- Data in the BitmapInfoHeader (no color table and no bits)

- Data in the BitmapInfoHeader and the color table (no bits)

- All the data (BitmapInfoHeader, color table, and the bits)

The fifth and the sixth parameters of the function are used to tell the graphics engine exactly what the application wants it to return. If the fifth parameter is NULL, then no bits will be returned. If the biBitCount is 0 (zero) in the sixth parameter, then no color table will be returned. In addition, the biSize field of the BitmapInfoHeader must be set to either the size of BitmapInfoHeader or BitmapCoreHeader for the function to work properly.

Refer to the SAVEBMP.C file in the MANDEL sample for details. This sample is included with the Win32 SDK.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiBmp

Using GetForegroundWindow() When Desktop Is Not Active

PSS ID Number: Q118624
Authored 25-Jul-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

GetForegroundWindow() is documented to return the handle of the foreground window, that is, the window that the user is currently working with. The proper handle is returned when the desktop that the application is running on is active; however, when another desktop is active, GetForegroundWindow() returns NULL.

This is expected behavior. There is no way to get the active window in your own desktop while another desktop is active.

The application desktop is one desktop. Other desktops include the logon and screen saver desktops. If GetForegroundWindow() returned a handle to the logon dialog box, it would be possible to create an application that could get user passwords. This would violate Windows NT security.

For this reason, it is not possible to create screen savers that melt or drop out.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: UsrWndw

Using GetUpdateRgn()

PSS ID Number: Q99047
Authored 20-May-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

The documentation provided with the Microsoft Windows Software Development Kit (SDK) for Microsoft Windows version 3.1 does not address all the issues surrounding the use of the GetUpdateRgn() function. This article complements the documentation of GetUpdateRgn().

MORE INFORMATION

The handle to the region (hRgn) does not have to be selected into a device context (DC) to be able to use it with GetUpdateRgn(). Even if the hRgn is selected as the clipping region for a DC, Windows only makes a copy of the hRgn for the hDC instead of using the hRgn directly.

When hRgn is used with GetUpdateRgn(), Windows will ignore any existing region in hRgn. It will replace any existing region with the current update region of the window.

Calling GetUpdateRgn() soon after BeginPaint() always yields an empty region because BeginPaint() validates the update region.

A typical Windows query function (functions that typically begin with Get and Is) does not initiate any action. GetUpdateRgn() is not a typical function in this respect. The third parameter, fErase, works as advertised to initiate a repaint on request.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrPnt

Using GMEM_DDESHARE in Win32 Programming

PSS ID Number: Q99114
Authored 23-May-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

The GMEM_DDESHARE flag remains a legitimate value for GlobalAlloc(). This flag can be used to indicate that the memory will be used for one of the following so that the system can optimize the allocation for these special needs:

DDE
OLE 1.0
Clipboard operations

However, GlobalAlloc(GMEM_DDESHARE, ...) cannot be used to allocate a block of memory that can be shared between processes. This flag was never intended for this purpose, even under Windows versions 3.0 and 3.1 (3.x). GlobalAlloc(GMEM_DDESHARE, ...) works in this case because all Windows-based applications share the same address space; this is not the case under Windows NT.

All allocations of global shared memory can be used within the process that they are allocated in, but another mechanism is required to share memory between processes.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseMm

Using Graphics Within a Help File

PSS ID Number: Q90291
Authored 12-Oct-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, versions 3.51 and 4.0

SUMMARY

This article explains how to use graphics in a Help file with both the help compiler 3.1 (HC30.EXE, HC31.EXE, and HCP.EXE) and the help compiler 4.0 (HCW.EXE).

MORE INFORMATION

TYPES OF GRAPHICS

With the Help Compiler, four types of graphics can be displayed within help topics: bitmaps, metafiles, segmented hypergraphics, and multiple resolution bitmaps. With HC30 and HC31, these graphics are limited to 16 colors but it is possible to use embedded windows to create a 256-color bitmap. HCW is capable of displaying 16 million colors.

The following section discusses the details of the graphics formats listed above, and provides details on their advantages and disadvantages.

Bitmaps

A bitmap is an image that is described by a matrix of memory bits that, when copied to a device, define the color and pattern of a corresponding matrix of pixels on the display surface of the device. The advantage to using a bitmap is that drawing a bitmap is very fast. The disadvantage is that the size of a bitmap is very large. Bitmaps can be created with a graphics editor, such as Paintbrush.

Metafiles

A metafile is a collection of GDI commands that creates desired text or images. There are two advantages to using metafiles: the size of the metafile is small, and metafiles are less device-dependent than bitmaps. The disadvantage of using a metafile is that it takes a long time to draw one.

Segmented Hypergraphics

A segmented hypergraphic is a graphic that has hot spots defined in various regions of the graphic. Clicking hot spots either executes a macro or jumps to a context string. To make a segmented hypergraphic, use the segmented hypergraphic (hot spot) editor (SHED.EXE) included with the Windows 3.1 SDK.

Multiple Resolution Bitmaps

A multiple resolution bitmap is a single bitmap file that contains one or more bitmaps that have been marked for use with specific displays. The advantages of multiple resolution bitmaps are:

1. Bitmaps are prevented from appearing too big or too small on different resolutions.

2. Bitmaps are prevented from looking stretched or compressed from display to display.

3. Colors are mapped correctly on different displays.

The disadvantage of multiple resolution bitmaps is that the files are large. multiple resolution bitmaps can be created from bitmap files with the multiple resolution bitmap compiler, MRBC.

PLACING GRAPHICS

Direct Pasting

Bitmaps and metafiles can be pasted directly from the clipboard into an RTF source file. This allows the help author to see what the topic will look like while it is being edited. There are several disadvantages to this method. The first disadvantage only applies to HC30 and HC31. It is that any graphic pasted directly into a topic is limited to 64K. This is the result of the help compiler's 64K per paragraph limit when processing RTF source files. The second disadvantage applies to all the help compilers. It is size. If the same graphic is used multiple times within the same source file, then a copy of the graphic is made each time it is placed within the source.

By Reference

All of the graphics can be placed "by reference." To insert a graphic by reference, the help author must type {bm? graphic.ext} where bm? is bml, bmr, or bmc and graphic.ext is the name of the graphic file that the author wants to have placed in the help topic. The bml command is used for a left-justified graphic, the bmr command is used for a right justified graphic, and the bmc command is used for a character justified graphic (that is, the graphic is inserted into the paragraph as if it were a character).

One of the advantages of placing a graphic by reference is that it lifts the 64K limit on a graphic. Also, a graphic placed by reference is actually just a "pointer" to the real graphic. Therefore, if the same graphic is used multiple times, it is only "stored" once within the .HLP file.

The disadvantage of placing graphics by reference is that the help author does not see how the topic will appear while in the RTF editor. The bitmap files inserted by reference must be found in the directory specified by either the ROOT or BMROOT settings in the [OPTIONS] section of the help project file. If the bitmap is not located in one of these directories, then the file must be listed in the [BITMAPS] section of the project file, so the help compiler can locate the bitmap.

Hot Spots and Pop Ups

When placing a graphic into an RTF source file, it can be turned into a hot spot or pop up, similar to other text. Just select the graphic and turn on the double or single underline attribute followed immediately by the hidden text for the context string or macro.

Additional reference words: 3.10 4.00 95 WinHelp
KBCategory: kbtool
KBSubcategory: TlsHlp

Using Network DDE Under Win32s

PSS ID Number: Q125475
Authored 29-Jan-1995 Last modified 17-Mar-1995

The information in this article applies to:

- Microsoft Win32s, version 1.2

Network Dynamic Data Exchange (NetDDE) has limited support in Win32s. You can use DDE across the network, however the NDde APIs are not supported under Win32s and will not be supported in the future.

The NDde APIs, such as NDdeShareAdd(), are used to create the NetDDE shares, not for the actual communication. Therefore, to use NetDDE with Win32s, manually create the shares with the DDESHARE utility included in the Windows for Workgroups Resource Kit or by thunking to the 16-bit NDde APIs. Then communicate through DDE or DDEML.

Additional reference words: 1.20
KBCategory: kbprg kbnetwork
KBSubcategory: W32s

Using NTFS Alternate Data Streams

PSS ID Number: Q105763
Authored 24-Oct-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

The documentation for the NTFS file system states that NTFS supports multiple streams of data; however, the documentation does not address the syntax for the streams themselves.

The Windows NT Resource Kit documents the stream syntax as follows:

```
filename:stream
```

Alternate data streams are strictly a feature of the NTFS file system and may not be supported in future file systems. However, NTFS will be supported in future versions of Windows NT.

Future file systems will support a model based on OLE 2.0 structured storage (IStream and IStorage). By using OLE 2.0, an application can support multiple streams on any file system and all supported operating systems (Windows, Macintosh, Windows NT, and Win32s), not just Windows NT.

MORE INFORMATION

The following sample code demonstrates NTFS streams:

```
#include <windows.h>
#include <stdio.h>

void main( )
{
   HANDLE hFile, hStream;
   DWORD dwRet;

   hFile = CreateFile( "testfile",
                    GENERIC_WRITE,
                FILE_SHARE_WRITE,
                            NULL,
                      OPEN_ALWAYS,
                                0,
                            NULL );
   if( hFile == INVALID_HANDLE_VALUE )
      printf( "Cannot open testfile\n" );
   else
      WriteFile( hFile, "This is testfile", 16, &dwRet, NULL );
```

```
   hStream = CreateFile( "testfile:stream",
                            GENERIC_WRITE,
                         FILE_SHARE_WRITE,
                                     NULL,
                              OPEN_ALWAYS,
                                       0,
                                    NULL );
   if( hStream == INVALID_HANDLE_VALUE )
      printf( "Cannot open testfile:stream\n" );
   else
      WriteFile( hStream, "This is testfile:stream", 23, &dwRet, NULL );
}
```

The file size obtained in a directory listing is 16, because you are looking only at "testfile", and therefore

```
   type testfile
```

produces the following:

```
   This is testfile
```

However

```
   type testfile:stream
```

produces the following:

```
   The filename syntax is incorrect
```

In order to view what is in testfile:stream, use:

```
   more < testfile:stream
```

-or-

```
   mep testfile:stream
```

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseFileio

Using One IsDialogMessage() Call for Many Modeless Dialogs

PSS ID Number: Q71450
Authored 18-Apr-1991 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

In the Windows environment, an application can implement more than one modeless dialog box with a single call to the IsDialogMessage() function. This can be done by using the following three-step method:

1. Maintain the window handle to the currently active modeless dialog box in a global variable.

2. Pass the global variable as the hDlg parameter to the IsDialogMessage() function, which is normally called from the application's main message loop.

3. Update the global variable whenever a modeless dialog box's window procedure receives a WM_ACTIVATE message, as follows:

   - If the dialog is losing activation (wParam is 0), set the global variable to NULL.

   - If the dialog is becoming active (wParam is 1 or 2), set the global variable to the dialog's window handle.

MORE INFORMATION

The information below demonstrates how to implement this technique.

1. Declare a global variable for the modeless dialog box's window handle.

```
HWND hDlgCurrent = NULL;
```

2. In the application's main message loop, add a call to the IsDialogMessage() function.

```
while (GetMessage(&msg, NULL, 0, 0))
   {
   if (NULL == hDlgCurrent || !IsDialogMessage(hDlgCurrent, &msg))
      {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
      }
   }
```

3. In the modeless dialog box's window procedure, process the WM_ACTIVATE message.

```
switch (message)
   {
   case WM_ACTIVATE:
      if (0 == wParam)             // becoming inactive
         hDlgCurrent = NULL;
      else                         // becoming active
         hDlgCurrent = hDlg;

      return FALSE;
   }
```

For more information on the WM_ACTIVATE message, see page 6-47 in "Microsoft Windows Software Development Kit Reference Volume 1" for the Windows SDK version 3.0 and page 87 of "Programmer's Reference, Volume 3: Messages, Structures, and Macros" for the Windows SDK version 3.1.

For details on the IsDialogMessage() function, see page 4-266 in "Windows Software Development Kit Reference Volume 1" for the Windows SDK version 3.0 and page 553 of "Programmer's Reference, Volume 2: Functions" for the Windows SDK version 3.1.

For details on using a modeless dialog box in an application for the Windows environment, see Chapter 10 of "Programming Windows," second edition, (Microsoft Press) written by Charles Petzold.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDlgs

Using Printer Escapes w/PS Printers on Windows NT & Win32s

PSS ID Number: Q124135
Authored 19-Dec-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.1, 1.15, 1.15a, and 1.2
- Microsoft Win32 Software Development Kit (SDK) versions 3.1 and 3.5

SUMMARY

To identify and print to PostScript printers from a Win32-based application under Windows NT and under Win32s, you need to special-case your code. This is because the printer drivers respond to different Printer Escapes under Windows NT and Windows/Win32s.

This article discusses how to identify and print to PostScript printers on both Windows NT and Win32s.

MORE INFORMATION

Identification

To identify the printer as a PostScript printer, use this code:

```
int gPrCode = 0;  // Set according to platform.

if( Win32s ) // Using the Win16 driver.
{
   gPrCode = PASSTHROUGH;
   if((Escape(printerIC, GETTECHNOLOGY, NULL, NULL, (LPSTR)szTech) &&
      !lstrcmp(szTech, "PostScript")) &&
      Escape(printerIC, QUERYESCSUPPORT, sizeof(int),
         (LPSTR)gPrCode, NULL )
   {
   // The printer is PostScript.
      ...
   }
}
else // Using Win32 driver under Windows NT.
{
   gPrCode = POSTSCRIPT_PASSTHROUGH; // Fails with Win16 driver
   if( Escape(printerIC, QUERYESCSUPPORT, sizeof(int), (LPSTR)gPrCode,
       NULL))
   {
   // The printer is PostScript.
      ...
   }
}
```

Printing

To send PostScript data to the printer on either platform, use this code:

```
// Assuming a buffer, szPSBuf, of max size MAX_PSBUF containing
// nPSData bytes of PostScript data.

   char szBuf[MAX_PSBUF+sizeof(short)];

// Store length in buffer.
   *((short *)szBuf) = nPSData;

// Store data in buffer.
   memcpy( (char *)szBuf + sizeof(short), szPSBuf, nPSData );

// Note that gPrCode (set when identifying the printer) depends on
// the platform.
   Escape( printerDC, gPrCode, (int) nPSData, szBuf, NULL );
```

However, your output may appear scaled or translated incorrectly or data may be transformed off the page under Win32s.

The origin and scale for Windows printer drivers is not the PostScript default (bottom left/72 dpi) but is instead at the upper left and at the device scale(300 dpi). Therefore, before sending data to the printer, you may need to send a couple of PostScript commands to scale or translate the matrix. For example, for scaling, send the following escape to scale the PostScript transform to 72 dpi:

```
   xres = GetDeviceCaps(printerDC, LOGPIXELSX);
   yres = GetDeviceCaps(printerDC, LOGPIXELSY);

// Two leading spaces for the following operation.
   wsprintf(szBuf, "  %d 72 div %d 72 div scale\n", xres, yres);

// Put actual size into buffer
   *((short *)szBuf) = strlen(szBuf)-2;
   Escape( printerDC, gPrCode, strlen(szBuf)-2, szBuf, NULL );
```

Additional reference words: 1.10 1.20 3.10 3.50
KBCategory: kbprg kbprint kbcode
KBSubcategory: GdiPrn W32s

Using Private Templates with Common Dialogs

PSS ID Number: Q74609
Authored 24-Jul-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, version 3.5

SUMMARY

An application which uses the common dialog library (COMMDLG.DLL) can provide its own dialog resource template to be used instead of the standard template. In this way, the application can include private dialog items specific to its needs without losing the benefits of using the COMMDLG's dialog handling.

MORE INFORMATION

Each common dialog data structure contains an lpTemplateName element. (Note that the Print Dialog structure contains two such elements, each with a distinct name -- see specifics of PrintDlg for details.) This element points to a null-terminated string that names the dialog box template resource to be substituted for the standard dialog template. If the dialog resource is numbered, the application can use the MAKEINTRESOURCE macro to convert the number into a pointer to a string. Alternatively, the application can choose to pass a handle to a preloaded dialog template. The Flags element of the dialog data structure must be set to indicate which method is being used.

After loading the application's dialog template, the common dialog DLL initializes the dialog items as it would for the standard template. This leads to an important point: all dialog items in the standard template must also exist in the application's private template. Note that the items do not have to be enabled or visible -- they just have to exist.

Once the DLL has finished handling the WM_INITDIALOG message, it passes that message on to the application's dialog hook function. The hook function handles WM_INITDIALOG by initializing the application's private dialog items. It can also disable and hide any items from the standard template that the application does not want to use.

The hook function should process messages and notifications concerning the private dialog items.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: UsrCmnDlg

Using Quoted Strings with Profile String Functions

PSS ID Number: Q69752
Authored 28-Feb-1991 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

Microsoft Windows provides profile files which are a mechanism for an application to store configuration about itself. The WIN.INI file is the system profile file in which Windows stores configuration information about itself. In versions of Windows prior to version 3.0, applications also stored configuration information in the WIN.INI file. Windows 3.0 introduced private profile files, which can store application-specific information.

An application can retrieve information from a profile file by calling the GetProfileString or GetPrivateProfileString function. If the profile file associates the specified lpKeyName value with a string that is delimited by quotation marks, Windows discards the quotation marks when it copies the associated string into the application-provided buffer.

For example, if the following entry appears in the profile file:

```
[application name]           [application name]
keyname = 'string'     or     keyname = "string"
```

The GetPrivateProfileString and GetProfileString functions read the string value and discard the quotation marks.

MORE INFORMATION

This behavior allows spaces to be put into a string. For example, the profile entry

```
keyname = string
```

returns the string without a leading space, whereas

```
keyname = ' string'     or     keyname = " string"
```

returns the string with a leading space.

Doubling quotation marks includes quotation marks in the string. For example:

```
keyname = ''string''    or     keyname = ""string""
```

returns the string with its quotation marks -- 'string' or "string".

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrIni

Using ReadFile() and WriteFile() on Socket Descriptors

PSS ID Number: Q104536
Authored 22-Sep-1993 Last modified 21-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

Socket handles for Windows NT sockets are object handles. For example, you can pass a socket handle in the appropriate state to the ReadFile(), ReadFileEx(), WriteFile(), or WriteFileEx() application programming interface (API) to receive and send data. The socket descriptor passed to the file APIs must be a connected, TCP descriptor.

NOTE: There is no way to specify send and receive out-of-band data.

To use a Windows Sockets handle, the ReadFile() and WriteFile() APIs must use asynchronous access. That is, you must specify the overlapped parameter in the call to ReadFile() and WriteFile(). This will allow you to be notified when the I/O has completed.

This functionality is based upon the implementation of Windows Sockets and may not be available to all implementations. For example, although this works in the Win32 subsystem, it is not supported under Win32s.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: NtwkWinsock

Using RLE Bitmaps for Animation Applications In Windows

PSS ID Number: Q75214
Authored 14-Aug-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

Animation that uses a "brute-force" animation scheme, displaying successive complete bitmaps, can be slow and choppy. The alternative approach, displaying an initial frame and then modifying the elements that change, can save a great deal of time, memory, and disk space.

MORE INFORMATION

Animation applications typically use a very simple algorithm to display a sequence of bitmaps, or frames, one right over another, on the same display surface. When the application stores an entire frame in the device-independent bitmap (DIB) format, it suffers from three particular bottlenecks:

1. With large frames or long sequences, the storage file becomes quite large.

2. Loading an entire sequence requires a large amount of memory.

3. Available Windows functions, such as BitBlt or SetDIBitsToDevice, may be too slow to animate the sequence smoothly.

For an animation sequence of bitmaps with a consistent set of colors, preprocessing the animation sequence and storing it in the run-length-encoded (RLE) format enables an application to run faster with a smaller storage file and memory requirement. Note that this RLE file is different from an RLE compressed DIB. This RLE format consists of several frames that are compressed according to the differences between frames. This process is straightforward and consists of five steps:

1. Select the first frame in the sequence and store it in the DIB format.

2. Compare the DIB bitmaps of consecutive frames. Due to the nature of animation sequences, the number of differences is often quite small compared to the size of the entire frame.

3. Encode the set of changed pixels into RLE format. The encoded frame will contain information only on the pixels that change, the delta

records will skip the unchanged pixels. The RLE bitmap is stored instead of the latter frame. For example, the RLE-encoded difference between the first and second frames is stored as the second frame.

4. When the entire sequence is preprocessed, bring each frame into the system as a BITMAPINFO structure and stream of bits. Since only the first frame is in the DIB format, the memory requirement is quite low. Moreover, frames that contain an identical set of colors can share the BITMAPINFO structure, only the biCompression and biSizeImage fields must be changed.

5. At display time, load the first frame as a DIB. Then use the SetDIBitsToDevice function to display subsequent frames in the sequence. Because this function requires much less information, the sequence can be animated much more quickly and smoothly.

Additional reference words: 3.00 3.10 3.50 4.00 95 rle
KBCategory: kbprg
KBSubcategory: GdiBmp

Using RPC Callback Functions

PSS ID Number: Q96781
Authored 25-Mar-1993 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The standard remote procedure call (RPC) model has a server containing one or more exported function calls, and a client, which calls the server's exported functions. However, Microsoft's implementation of RPC defines callbacks as a special interface definition language (IDL) attribute allowing a server to call a client function.

Callbacks can be used only in the context of a server call. Thus, a server may call a client's callback function only when the server is performing a client's remote procedure call (before it returns from processing). For example:

```
                CLIENT                SERVER
                ------                ------

       Client makes RPC call. --->
                                  <--- Server calls callback procedure.
Client returns from callback. --->
                                  <--- Server calls callback procedure.
Client returns from callback. --->
                                  <--- Server returns from original RPC call.
```

MORE INFORMATION

Callbacks are declared in the RPC .IDL file and defined in the source of the client. The following demonstrates how callbacks are declared and defined:

```
[ SAMPLE.IDL ]
[
   uuid(9FEE4F51-0396-101A-AE4F-08002B2D0065),
   version(1.0),
   pointer_default( unique )
]

{
 void RPCProc( [in, string] unsigned char *pszStr );
 [callback] void CallbackProc([in,string] unsigned char *pszStr);
}
```

```
[ SAMPLEC.C (Client)]
/*
   Callback RPC call (initiated from server, executed on client).
*/
void CallbackProc( unsigned char *pszString )
{
  printf("Call from server, printed on client: %s", pszStr );
}


[ SAMPLES.C (Server)]
/*
   "Standard" RPC call (initiated from client, executed on server).
   Makes a call to client callback procedure, CallbackProc().
*/
void RPCProc( unsigned char *pszStr )
{
  printf("About to call Callback() client function.."
  CallbackProc( pszStr );
  printf("Called callback function.");
}
```

In the makefile for the sample, the "-ms_ext" switch must be used for the MIDL compile. For example:

```
   midl -ms_ext -cpp_cmd $(cc) -cpp_opt "-E" sample.idl
```

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: NtwkRpc

Using SendMessage() As Opposed to SendDlgItemMessage()

PSS ID Number: Q12273
Authored 16-Oct-1987 Last modified 10-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1 and 3.5

The following information describes under what circumstances it is appropriate to use either the SendMessage() or SendDlgItemMessage() function.

Both SendMessage() and SendDlgItemMessage() can be used to add strings to a list box. SendMessage() is used to send a message to a specific window using the handle to the list box. SendDlgItemMessage() is used to send a message to the child window of a given window using the list box resource ID. SendDlgItemMessage() is most often used in dialog box functions that have a handle to the dialog box and not to the child window control.

The SendDlgItemMessage() call

```
SendDlgItemMessage (hwnd, id, msg, wParam, lParam)
```

is equivalent to the following SendMessage() call:

```
hwnd2 = GetDlgItem (hwnd, id);
SendMessage (hwnd2, msg, wParam, lParam);
```

Please note that PostMessage() should never be used to talk to the child windows of dialog boxes for the following reasons:

1. PostMessage() will only return an error if the message was not posted to the control's message queue. Since many messages are sent to control return information, PostMessage() will not work, since it does not return the information to the caller.

2. 16-bit only: Messages such as the WM_SETTEXT message that include a far pointer to a string can potentially cause problems if posted using the PostMessage() function. The far pointer may point into a buffer that is inside the DS (data segment). Because PostMessage() does not process the message immediately, the DS might get moved. If the DS is moved before the message is processed, the far pointer to the buffer will be invalid.

Additional reference words: 3.00 3.10 3.50
KBCategory: kbprg
KBSubcategory: UsrMsg

Using SendMessageTimeout() in a Multithreaded Application

PSS ID Number: Q106716
Authored 14-Nov-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

SendMessageTimeout() is new to the Win32 application programming interface (API). The function sends a message to a window and does not return until either the window procedure processes the message or the timeout occurs.

MORE INFORMATION

This article uses the following scenario to illustrate the behavior of this function:

Suppose that Win1 and Win2 are created by the same application and that the following code is included in their window procedures:

```
   <WindowProc for window Win1>
      ...
      case <xxx>:
         ...
         SendMessageTimeout(      hWnd2,   // window handle
                            WM_USER+1,   // message to send
                               wParam,   // first message parameter
                               lParam,   // second message parameter
                            SMTO_NORMAL,   // flag *
                                    100,   // timeout (in milliseconds)
                                   &ret ); // return value
        ...
         break;

      case WM_USER+2:
        <time-consuming procedure>
         break;
```

* Note that the SMTO_NORMAL flag indicates that the calling thread can process other requests while waiting for the API to return.

```
   <WindowProc for window Win2>
      ...
      case WM_USER+1:
         ...
         SendMessage(     hWnd1,    // window handle
                      WM_USER+2,    // message to send
                         wParam,    // first message parameter
```

```
                    lParam );  // second message parameter
        OtherStuff();
        ...
        break;
```

If Win1 executes this SendMessageTimeout() and Win2 uses SendMessage() to send a message to Win1, Win1 can process the message because SMTO_NORMAL was specified. If the SendMessageTimeout() expires while the execution is currently in the window procedure for Win1, the state of the system will depend on who owns the windows.

If both windows were created by the same thread, the timeout is not used and the process proceeds exactly as if SendMessage() was being used. If the windows are owned by different threads, the results can be unpredictable, because the timeout is restarted whenever a message or some other system event is received and processed. In other words, the receipt by Win1 of WM_USER+2 causes the timeout to restart after the message is processed. If the function executed by Win2, OtherStuff(), then uses up more than 100 milliseconds without awakening the thread that created Win1, the original SendMessageTimeout() will timeout and return. The OtherStuff() function continues to completion but any value that was to be returned to Win1 will be lost. Note that the code paths will always complete.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMisc

Using Serial Communications Under Win32s

PSS ID Number: Q105759
Authored 24-Oct-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.0, 1.1, and 1.2

Windows NT and Windows provide significantly different serial communications application programming interfaces (APIs). Win32s does not support the Win32 Communications API.

A good approach to take in writing a Win32-based application targetted for Win32s that uses serial communications is to create a pair of dynamic-link libraries (DLLs) with the same name. One DLL will use Win32 Communications APIs and be installed under Windows NT. The other DLL will use the Universal Thunk to call a 16-bit DLL that will call the Windows Communications API. This DLL will be installed under Win32s.

For more information on the Universal Thunk, see the "Win32s Programmers Guide" included with the Software Development Kit (SDK). In addition, there is a sample in MSTOOLS\WIN32S\UT\SAMPLES\UTSAMPLE.

Additional reference words: 1.00 1.10 1.20 comm
KBCategory: kbprg
KBSubcategory: W32s

Using SetClassLong Function to Subclass a Window Class

PSS ID Number: Q32519
Authored 06-Jul-1988 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

It is possible to subclass an entire window class by using the SetClassLong() function. However, doing so will only subclass windows of that class created after the call to the SetClassLong() function. Windows created before the call to the SetClassLong() function are not affected.

NOTE: In Win32, SetClassLong() only affects Windows in the same address space. For example, if you subclass EDIT, only edit controls created in your application will be subclassed.

MORE INFORMATION

Calling the SetClassLong() function with the GCL_WNDPROC index changes the class function address for that window class, creating a subclass of the window class. When a subsequent window of that class is created, the new class function address is inserted into its window structure, subclassing the new window. Windows created before the call to the SetClassLong() function (in other words, before the class function address was changed) are not subclassed.

An application should not use the SetClassLong() function to subclass standard Windows controls such as edit controls or buttons. If, for example, an application were to subclass the entire "edit" class, then subsequent edit controls created by other applications would be subclassed.

An application can subclass individual standard Windows controls that it has created by calling the SetWindowLong() function.

Additional reference words: listbox scrollbar 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: UsrWndw

Using SetThreadLocale() for Language Resources

PSS ID Number: Q99392
Authored 27-May-1993 Last modified 24-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

Under Windows NT, each resource loading application programming interface (API) is based on the thread's locale. Each thread has a locale--usually the default system locale.

You can change the thread locale by calling SetThreadLocale(). To obtain the language resource you want, just set the thread locale to the locale you want, then call the normal resource loading API.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: UsrNls WintlDev

Using Temporary File Can Improve Application Performance

PSS ID Number: Q103237
Authored 19-Aug-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

The use of temporary files can significantly increase the performance of an application.

MORE INFORMATION

By using CreateFile() with the FILE_ATTRIBUTE_TEMPORARY flag, you let the system know that the file is likely to be short lived. The temporary file is created as a normal file. The system needs to do a minimal amount of lazy writes to the file system to keep the disk structures (directories and so forth) consistent. This gives the appearance that the file has been written to the disk. However, unless the Memory Manager detects an inadequate supply of free pages and starts writing modified pages to the disk, the Cache Manager's Lazy Writer may never write the data pages of this file to the disk. If the system has enough memory, the pages may remain in memory for any arbitrary amount of time. Because temporary files are generally short lived, there is a good chance the system will never write the pages to the disk.

To further increase performance, your application might mark the file as FILE_FLAG_DELETE_ON_CLOSE. This indicates to the system that when the last handle of the file is closed, it will be deleted. Although the system generally purges the cache to ensure that a file being closed is updated appropriately, because a file marked with this flag won't exist after the close, the system foregoes the cache purge.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseFileio

Using Text Bullets in a Rich Edit Control

PSS ID Number: Q129859
Authored 08-May-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.51 and 4.0
- Microsoft Win32s, version 1.3

The Rich Edit control contains a built-in text bullet feature that is used to add paragraph bullets to the text. To use this feature, you must send the Rich Edit control a EM_SETPARAFORMAT message. The EM_SETPARAFORMAT message takes a pointer to a PARAFORMAT structure as its lParam parameter. In the PARAFORMAT structure, it is necessary to zero out the structure and fill in the following members:

UINT cbSize - Contains the size of the structure. Use sizeof(PARAFORMAT).

DWORD dwMask - Contains the attributes to set. For bullets, use PFM_NUMBERING | PFM_OFFSET.

WORD wNumbering - Contains the value that specifies numbering options. For bullets, set this member equal to PFN_BULLET.

LONG dxOffset - Contains the value that specifies indentation of the second line and subsequent lines, relative to the starting indentation. For bullets, this value must be positive because the bullet is displayed in the area between the starting indentation and the offset. If this value is too small, the bullets are not displayed.

Additional reference words: 1.30 4.00 95
KBCategory: kbprg kbui
KBSubcategory: UsrCtl W32s

Using the C Run-Time

PSS ID Number: Q94248
Authored 31-Dec-1992 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

This document contains the following sections:

Section 1: Three Forms of C Run-Time (CRT) Library Are Available
Section 2: Using the CRT Libraries When Building a DLL
Section 3: Using NTWIN32.MAK to Simplify the Build Process
Section 4: Problems Encountered When Using Multiple CRT Libraries
Section 5: Mixing Library Types

MORE INFORMATION

Section 1: Three Forms of C Run-Time (CRT) Libraries Are Available

There are three forms of the C Run-time library provided with the Win32 SDK:

- LIBC.LIB is a statically linked library for single-threaded programs.

- LIBCMT.LIB is a statically linked library that supports multithreaded programs.

- CRTDLL.LIB is an import library for CRTDLL.DLL that also supports multithreaded programs. CRTDLL.DLL itself is part of Windows NT.

Microsoft Visual C++ 32-bit edition contains these three forms as well, however, the CRT in a DLL is named MSVCRT.LIB. The DLL is redistributable. Its name depends on the version of VC++ (ie MSVCRT10.DLL or MSVCRT20.DLL). Note however, that MSVCRT10.DLL is not supported on Win32s, while CRTDLL.LIB is supported on Win32s. MSVCRT20.DLL comes in two versions: one for Windows NT and the other for Win32s.

Section 2: Using the CRT Libraries When Building a DLL

When building a DLL which uses any of the C Run-time libraries, in order to ensure that the CRT is properly initialized, either

1. the initialization function must be named DllMain() and the entry point must be specified with the linker option -entry:_DllMainCRTStartup@12

- or -

2. the DLL's entry point must explicitly call CRT_INIT() on process attach and process detach

This permits the C Run-time libraries to properly allocate and initialize C Run-time data when a process or thread is attaching to the DLL, to properly clean up C Run-time data when a process is detaching from the DLL, and for global C++ objects in the DLL to be properly constructed and destructed.

The Win32 SDK samples all use the first method. Use them as an example. Also refer to the Win32 Programmer's Reference for DllEntryPoint() and the Visual C++ documentation for DllMain(). Note that DllMainCRTStartup() calls CRT_INIT() and CRT_INIT() will call your application's DllMain(), if it exists.

If you wish to use the second method and call the CRT initialization code yourself, instead of using DllMainCRTStartup() and DllMain(), there are two techniques:

1. if there is no entry function which performs initialization code, simply specify CRT_INIT() as the entry point of the DLL. Assuming that you've included NTWIN32.MAK, which defines DLLENTRY as "@12", add the following option to the DLL's link line:

   ```
   -entry:_CRT_INIT$(DLLENTRY)
   ```

   - or -

2. if you *do* have your own DLL entry point, do the following in the entry point:

   a. Use this prototype for CRT_INIT():

      ```
      BOOL WINAPI _CRT_INIT(HINSTANCE hinstDLL, DWORD fdwReason,
         LPVOID lpReserved);
      ```

      For information on CRT_INIT() return values, see the documentation DllEntryPoint; the same values are returned.

   b. On DLL_PROCESS_ATTACH and DLL_THREAD_ATTACH (see "DllEntryPoint" in the Win32 API reference for more information on these flags), call CRT_INIT(), first, before any C Run-time functions are called or any floating-point operations are performed.

   c. Call your own process/thread initialization/termination code.

   d. On DLL_PROCESS_DETACH and DLL_THREAD_DETACH, call CRT_INIT() last, after all C Run-time functions have been called and all floating-point operations are completed.

   Be sure to pass on to CRT_INIT() all of the parameters of the entry point; CRT_INIT() expects those parameters, so things may not work reliably if they are omitted (in particular, fdwReason is required to determine whether process initialization or termination is needed).

Below is a skeleton sample entry point function that shows when and how to make these calls to CRT_INIT() in the DLL entry point:

```
BOOL WINAPI DllEntryPoint(HINSTANCE hinstDLL, DWORD fdwReason,
    LPVOID lpReserved)
{
if (fdwReason == DLL_PROCESS_ATTACH || fdwReason == DLL_THREAD_ATTACH)
    if (!_CRT_INIT(hinstDLL, fdwReason, lpReserved))
        return(FALSE);

if (fdwReason == DLL_PROCESS_DETACH || fdwReason == DLL_THREAD_DETACH)
    if (!_CRT_INIT(hinstDLL, fdwReason, lpReserved))
        return(FALSE);
return(TRUE);
}
```

NOTE that this is *not* necessary if you are using DllMain() and -entry:_DllMainCRTStartup@12.

Section 3: Using NTWIN32.MAK to Simplify the Build Process

There are macros defined in NTWIN32.MAK that can be used to simplify your makefiles and to ensure that they are properly built to avoid conflicts. For this reason, Microsoft highly recommends using NTWIN32.MAK and the macros therein.

For compilation, use:

$(cvarsdll)	for apps/DLLs using CRT in a DLL

For linking, use one of the following:

$(conlibsdll)	for console apps/DLLs using CRT in a DLL
$(guilibsdll)	for GUI apps using CRT in a DLL

Section 4: Problems Encountered When Using Multiple CRT Libraries

If an application that makes C Run-time calls links to a DLL that also makes C Run-time calls, be aware that if they are both linked with one of the statically-linked C Run-time libraries (LIBC.LIB or LIBCMT.LIB), the .EXE and DLL will have separate copies of all C Run-time functions and global variables. This means that C Run-time data cannot be shared between the .EXE and the DLL. Some of the problems that can occur as a result are:

- Passing buffered stream handles from the .EXE/DLL to the other module
- Allocating memory with a C Run-time call in the .EXE/DLL and reallocating or freeing it in the other module
- Checking or setting the value of the global errno variable in the .EXE/DLL and expecting it to be the same in the other module. A related problem is calling perror() in the opposite module from where the C Run-time error occurred, since perror() uses errno.

To avoid these problems, link both the .EXE and DLL with CRTDLL.LIB or MSVCRT.LIB, which allows both the .EXE and DLL to use the common set of functions and data contained within CRT in a DLL, and C Run-time data such as stream handles can then be shared by both the .EXE and DLL.

Section 5: Mixing Library Types

You can link your DLL with CRTDLL.LIB/MSVCRT.LIB regardless of what your .EXE is linked with if you avoid mixing CRT data structures and passing CRT file handles or CRT FILE* pointers to other modules.

When mixing library types adhere to the following:

- CRT file handles may only be operated on by the CRT module that created them.

- CRT FILE* pointers may only be operated on by the CRT module that created them.

- Memory allocated with the CRT function malloc() may only be freed or reallocated by the CRT module that allocated it.

To illustrate this, consider the following example:

  - .EXE is linked with MSVCRT.LIB
  - DLL A is linked with LIBCMT.LIB
  - DLL B is linked with CRTDLL.LIB

  If the .EXE creates a CRT file handle using _create() or _open(), this file handle may only be passed to _lseek(), _read(), _write(), _close(), etc. in the .EXE file. Do not pass this CRT file handle to either DLL. Do not pass a CRT file handle obtained from either DLL to the other DLL or to the .EXE.

  If DLL A allocates a block of memory with malloc(), only DLL A may call free(), _expand(), or realloc() to operate on that block. You cannot call malloc() from DLL A and try to free that block from the .EXE or from DLL B.

  NOTE: If all three modules were linked with CRTDLL.LIB or all three were linked with MSVCRT.LIb, these restrictions would not apply.

When linking DLLs with LIBC.LIB, be aware that if there is a possibility that such a DLL will be called by a multithreaded program, the DLL will not support multiple threads running in the DLL at the same time, which can cause major problems. If there is a possibility that the DLL will be called by multithreaded programs, be sure to link it with one of the libraries that support multithreaded programs (LIBCMT.LIB, CRTDLL.LIB or MSVCRT.LIB).

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsMisc

Using the Call-Attributed Profiler (CAP)

PSS ID Number: Q118890
Authored 01-Aug-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, versions 3.1 and 3.5

SUMMARY

The call-attributed profiler (CAP) allows you to profile function calls within an application.

MORE INFORMATION

To profile an application using CAP, perform the following steps:

1. Replace the Windows NT system DLLs that your application uses with DLLs that contain debugging information. To find out which DLLs your application uses, run the APF32CVT.EXE utility provided with the Win32 SDK. The syntax to use is as follows:

   ```
   apf32cvt <application>
   ```

   This will list the DLLs to which your program is linked. The system DLLs that contain debugging information can be found in the \SUPPORT\DEBUGDLL\i386 of the Win32 SDK CD. Rename the DLLs in your WINNT\SYSTEM32 directory and copy the debugging DLLs to the WINNT\SYSTEM32 directory. You will need to reboot the machine to use the DLLs.

2. Recompile your application. If you are using the NTWIN32.MAK file in your makefile, all you need to do is to set the environment variable PROFILE=on. Otherwise, add /Gh and /Zd to the compiler options yourself and be sure that you are linking with the options "-debugtype:coff" and "-debug:partial,mapped".

3. Place a CAP.INI file in either the root directory of the drive, the application directory, the WINDOWS directory, or the root of the C drive. The CAP.INI file specifies the applications for which the profiler will gather information. At minimum, CAP.ini must contain the following:

   ```
   [EXES]
   <app>.exe
   [PATCH IMPORTS]
   <app>.exe
   [PATCH CALLERS]
   ```

   where <app> is the application to be profiled. The file CAP.TXT

included in the \BIN directory of the SDK provides an excellent example.

4. Run the application. The profiling information is gathered and stored in a file with the same base name as the application and a .END extension. This information is in an ASCII format and can be viewed by any text editor. You can also use the CAPVIEW sample to view a graphical representation of the information.

Walter Oney points out in "Removing Bottlenecks from Your Program with Windows NT Performance-tuning Tools," from the April 1994 edition of "Microsoft Systems Journal," that the Visual C++ linker does not correctly generate debugging information that CAP can use. This is not correct. The problem is that the SDK 3.1 linker uses "-debug:mapped" by default, but the Visual C++ linker does not. Adding the switch to the link line (as in step 2, above) corrects this problem.

A common problem is for the profiling output to have "??? : ???" in place of the function names from your application. For example:

```
   1   ??? : ??? (Address=0x77889a1b)    1        4717        4717
             4717        4717        4717         n/a         n/a
```

This occurs if you use the wrong linker options. You should use "-debugtype:coff" and "-debug:partial,mapped".

Another common problem is to have function pointers instead of the Win32 API names. For example:

```
   1    0x77e9b10f                        1      1577         1577
             1577        1577        1577         n/a         n/a
```

This happens when you do not replace the system DLLs that your application calls with the DLLs that contain debugging information.

REFERENCES

The release notes for CAP.TXT can be found in the MSTOOLS\BIN directory.

The best source of information is "Optimizing Windows NT" by Russ Blake in the "Windows NT Resource Kit, Vol. 3".

Additional reference words: 3.10 3.50
KBCategory: kbtool
KBSubcategory: TlsMisc

Using the DeferWindowPos Family of Functions

PSS ID Number: Q87345
Authored 29-Jul-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

In the Microsoft Windows graphical environment, an application can use the BeginDeferWindowPos, DeferWindowPos, and EndDeferWindowPos functions when it moves or sizes a set of windows simultaneously. Using these functions avoids unnecessary screen painting, which would occur if the windows were moved individually.

The eighth parameter to the DeferWindowPos function can be any one of eight flag values that affect the size and position of each moved or sized window. One of the flags, SWP_NOREDRAW, disables repainting and prevents Windows from displaying any changes to the screen. This flag effects both the client and nonclient areas of the window. Any portion of its parent window uncovered by the move or size operation must be explicitly invalidated and redrawn.

If the moved or sized windows are child windows or pop-up windows, then the SWP_NOREDRAW flag has the expected effect. However, if the window is an edit control, a combo box control, or a list box control, then specifying SWP_NOREDRAW has no effect; the control is drawn at its new location and its previous location is not erased. This behavior is caused by the manner in which these three control classes are painted. Buttons and static controls function normally.

To work around this limitation and move a group of edit, list box, and combo box controls in a visually pleasing manner, perform the following three steps:

1. Use the ShowWindow function to hide all of the controls.

2. Move or size the controls as required with the MoveWindow and SetWindowPos functions.

3. Use the ShowWindow function to display all of the controls.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95 combobox listbox
KBCategory: kbprg
KBSubcategory: UsrWndw

Using the Document Properties Dialog Box

PSS ID Number: Q118622
Authored 25-Jul-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

A number of applications display a printer-configuration dialog box titled "Document Properties" when the user chooses File from the application menu, chooses Print, and then chooses the Setup button.

The DocumentProperties() API is used to display this dialog box. You need to call DocumentProperties() with the DM_IN_PROMPT bit set in the last parameter (fMode). You also need to call OpenPrinter() before calling DocumentProperties().

MORE INFORMATION

The following code demonstrates how to call DocumentProperties():

Sample Code

```
   HDC         hPrnDC;
   LPDEVMODE   lpDevMode = NULL;
   LPDEVNAMES  lpDevNames;
   LPSTR       lpszDriverName;
   LPSTR       lpszDeviceName;
   LPSTR       lpszPortName;
   PRINTDLG    pd;
   HANDLE      hPrinter;
   int         nDMSize;
   HANDLE      hDevMode;
   NPDEVMODE   npDevMode;
   DEVMODE     DevModeIn;

   // Get the defaults without displaying any dialog boxes.

   pd.Flags = PD_RETURNDEFAULT;
   pd.hDevNames = NULL;
   pd.hDevMode = NULL;
   pd.lStructSize = sizeof(PRINTDLG);
   PrintDlg((LPPRINTDLG)&pd);

   lpDevNames = (LPDEVNAMES)GlobalLock(pd.hDevNames);
   lpszDriverName = (LPSTR)lpDevNames + lpDevNames->wDriverOffset;
   lpszDeviceName = (LPSTR)lpDevNames + lpDevNames->wDeviceOffset;
```

```
   lpszPortName   = (LPSTR)lpDevNames + lpDevNames->wOutputOffset;

   OpenPrinter(lpszDeviceName,&hPrinter,NULL);

   // A zero for last param returns the size of buffer needed.

   nDMSize = DocumentProperties(hWnd,hPrinter,lpszDeviceName,NULL,NULL,0);
   if ((nDMSize < 0) || !(hDevMode = LocalAlloc (LHND, nDMSize)))
      return NULL;

   npDevMode = (NPDEVMODE) LocalLock (hDevMode);

   // Fill in the rest of the structure.

   lstrcpy (DevModeIn.dmDeviceName, lpszDeviceName);
   DevModeIn.dmSpecVersion    = 0x300;
   DevModeIn.dmDriverVersion  = 0;
   DevModeIn.dmSize           = sizeof (DevModeIn);
   DevModeIn.dmDriverExtra    = 0;

   // Display the "Document Properties" dialog box.

   DocumentProperties(hWnd,hPrinter,lpszDeviceName,npDevMode,&DevModeIn,
      DM_IN_PROMPT|DM_OUT_BUFFER);

   // Get the printer DC.

   hPrnDC = CreateDC
(lpszDriverName,lpszDeviceName,lpszPortName,(LPSTR)npDevMode);
   LocalUnlock (hDevMode);

   //  Use the printer DC.

   ...
```

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiPrn

Using the DRAWPATTERNRECT Escape in Windows

PSS ID Number: Q75380
Authored 19-Aug-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.1 and 3.0
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The DRAWPATTERNRECT escape is implemented by the PCL/HP driver for Hewlett-Packard (HP) LaserJet printers and compatibles. The escape is used to draw a patterned rectangle without using the graphics banding bitmap. Using this escape can enhance the performance for many applications, particularly when a majority of users have LaserJet-compatible printers.

MORE INFORMATION

The HP LaserJet Plus and all later LaserJets implement a command called a rule. A rule is a rectangle filled with some pattern, such as a black rule, a quasi-halftone gray scale, or a hatch pattern.

The output does not go through the graphics banding bitmap (it is actually sent to the printer in the text band). The DRAWPATTERNRECT escape can be used to print line and block graphics in the text band without using graphics banding at all. Because many applications use only horizontal and vertical lines or blocks, this is a significant optimization.

An application should determine support for rules using the QUERYESCSUPPORT escape. In particular, the application should not check for the PCL/HP driver, since other page printer drivers may implement the escape as well.

There are some limitations to the escape. First, rules drawn with DRAWPATTERNRECT are not subject to clipping regions in the Device Context (DC). Second, rules cannot be opaqued; no white pixel in the graphics band will erase a pixel drawn by a rule (the same limitation occurs for PCL text). Once a rule is drawn, it cannot be erased.

If these limitations are acceptable, if all graphics on the page are likely to be horizontal and vertical lines, and if a significant number of users are expected to have LaserJet-type printers (which is the case for most Windows-based applications), the DRAWPATTERNRECT escape should be used.

If for any reason DRAWPATTERNRECT cannot be used, then the application should generally use the PatBlt function. If the device is a

plotter, the Rectangle function should be used. In the case of the PatBlt function, if a black rectangle is to be printed, the BLACKNESS raster operator (ROP) should be used to avoid the overhead of selecting and later deselecting a black brush into the printer DC.

Additional reference words: 3.00 3.10 3.50 4.00 95 HPPCL HP-PCL
KBCategory: kbprg
KBSubcategory: GdiPrn

Using the DS_SETFONT Dialog Box Style

PSS ID Number: Q87344
Authored 29-Jul-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

In the Microsoft Windows graphical environment, an application can affect the appearance of a dialog box by specifying the DS_SETFONT style bit. DS_SETFONT is available only when the application creates a dialog box dynamically from a memory-resident dialog box template using the CreateDialogIndirect, CreateDialogIndirectParam, DialogBoxIndirect, or DialogBoxIndirectParam function. The second parameter to each of these functions is the handle to a global memory object that contains a DLGTEMPLATE dialog box template data structure. The dwStyle (first) member of the DLGTEMPLATE structure contains style information for the dialog box.

When an application creates a dialog box using one of these functions, Windows determines whether the template contains a FONTINFO data structure by checking for the DS_FONTSTYLE bit in the dwStyle member of the DLGTEMPLATE structure. If this bit is set, Windows creates a font for the dialog box and its controls based on the information in the FONTINFO structure. Otherwise, Windows uses the default system font to calculate the size of the dialog box and the placement and text of its controls.

If Windows creates a font based on the FONTINFO data structure, it sends a WM_SETFONT message to the dialog box. If Windows uses the system default font, it does not send a WM_SETFONT message. A dialog box can change the font of one or more of its controls by creating a font and sending a WM_SETFONT message with the font handle to the appropriate controls.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDlgs

Using the FORCEFONT .HPJ Option

PSS ID Number: Q93395
Authored 09-Dec-1992 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

The FORCEFONT Help project file (HPJ) option can only be used with the following font names:

```
Helv
Helvetica
Courier
Tms Rmn
Times
Symbol
```

Though Helv, Helvetica, Times, and Tms Rmn are not Windows 3.1 fonts, each of them is mapped to a 3.1 font in the [FontSubstitutes] section of WIN.INI by Windows Setup. They are mapped as follows:

```
Helv=MS Sans Serif
Tms Rmn=MS Serif
Times=Times New Roman
Helvetica=Arial
```

To force a Help file to use MS Sans Serif, the FORCEFONT option should be:

```
FORCEFONT=Helv
```

Additional reference words: 3.10 3.50 4.00 95 HC30 HC31 HCP help compiler MAPFONTSIZE
KBCategory: kbtool
KBSubcategory: TlsHlp

Using the GetWindow() Function

PSS ID Number: Q33161
Authored 20-Jul-1988 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

An application can use the GetWindow function to enumerate the windows that have a specified relationship to another window. For example, an application can determine the windows that are children of the application's main window.

MORE INFORMATION

The GetWindow function returns NULL when no more windows match the specified criteria. Given a window handle, hWnd, the following code determines how many siblings the associated window has:

```
int CountSiblings(HWND hWnd)
   {
   HWND hWndNext;
   short nCount = 0;

   hWndNext = GetWindow(hWnd, GW_HWNDFIRST);
   while (hWndNext != NULL)
      {
      nCount++;
      hWndNext = GetWindow(hWndNext, GW_HWNDNEXT);
      }

   return nCount;
   }
```

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrWndw

Using the Registry Under Win32s

PSS ID Number: Q129542
Authored 27-Apr-1995 Last modified 29-Apr-1995

The information in this article applies to:

- Microsoft Win32s, version 1.2

SUMMARY

A Win32s-based application running under Win32s is limited to the Windows version 3.1 view of the registry. This means that there are no named values. In addition, HKEY_CLASSES_ROOT is available, but not HKEY_LOCAL_MACHINE or HKEY_CURRENT_USER.

The Windows version 3.1 registry is very limited in size. Microsoft recommends that you store only the OLE registration information in the Windows registry.

MORE INFORMATION

While Win32s does support some Registry APIs that Windows does not, it does not support all of the Win32 Registry APIs. For a complete list of the Registry APIs supported under Win32s, please see the file WIN32API.CSV, which is included with the Win32 Software Development Kit (SDK) and in Microsoft Visual C++ (32-bit edition).

NOTE: All supported Registry APIs, except for RegQueryValueA() and RegQueryValueExA(), return the error codes defined for Windows verion 3.1, not the codes defined by the Win32 API.

Additional reference words: 1.20
KBCategory: kbprg
KBSubcategory: W32s

Using the Windows 95 Common Controls on Windows NT and Win32s

PSS ID Number: Q125672
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)
- Microsoft Win32s version 1.3

SUMMARY

To provide greater compatibility between Windows 95, Windows NT, and Win32s, the common control library from Windows 95 has been ported to Windows NT and Win32s starting with Windows NT version 3.51 and Win32s version 1.3.

MORE INFORMATION

The COMCTL32.DLL that provides the common controls in Windows 95 is not compatible with Windows NT or Win32s, so adding the controls to Windows NT is not as simple as copying the DLL. Also, the COMCTL32.DLL that comes with Windows NT version 3.51 and with Win32s is not redistributable. Customers that want to run programs using the new controls must be running Windows NT version 3.51 or Win32s version 1.3.

Windows NT version 3.51 and Win32s version 1.3 are targeted to be released before the release of Windows 95, so any changes in the functionality of these controls between the release of Windows NT and Win32s and the release of Windows 95 will be added to Windows NT and Win32s in their next updates.

Additional reference words: 4.00 1.30 3.51
KBCategory: kbui
KBSubcategory: UsrCtl

Using the WM_VKEYTOITEM Message Correctly

PSS ID Number: Q108941
Authored 20-Dec-1993 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

The WM_VKEYTOITEM message is sent by a list box with the LBS_WANTKEYBOARDINPUT style to its owner in response to a WM_KEYDOWN message. The return value from this message specifies the action that the application performed in response to the message. A return value of -2 indicates that the application handled all aspects of selecting the item and requires no further action by the list box.

This is true only for certain keys such as the UP ARROW (VK_UP), DOWN ARROW (VK_DOWN), PAGE DOWN (VK_NEXT) and PAGE UP (VK_PREVIOUS) keys. All other keys are handled in the normal way by the list box, regardless of the return value from the WM_VKEYTOITEM message.

MORE INFORMATION

The documentation for the WM_VKEYTOITEM message indicates that the owner of a list box control can trap the WM_VKEYTOITEM message that is generated in response to a WM_KEYDOWN message and return -2 if the application does not want the default list box window procedure to take further action.

This is valid only for keys that are not translated into a character by the list box control in Windows. If the WM_KEYDOWN message translates to a WM_CHAR message and the application processes the WM_VKEYTOITEM message generated as a result of the keydown, the list box ignores the return value (it will go ahead and do the default processing for that character).

WM_KEYDOWN messages generated by keys such as VK_UP (UP ARROW), VK_DOWN (DOWN ARROW), VK_NEXT (PAGE DOWN) and VK_PREVIOUS (PAGE UP) are not translated to WM_CHAR messages. In such cases, trapping the WM_VKEYTOITEM message and returning a -2 prevents the list box from doing the default processing for that key.

For example, if an application traps the DOWN ARROW key and does some nondefault processing (such as moving the selection to the item two indexes below the currently selected item) and then returns -2, the list box control will not do any more processing with this message.

Alternatively, if the application trapped the "A" key, does some nondefault processing, and returns -2, the list box code will still do the default processing. The list box will select an item in the list box that starts with an "A" (if one is present).

To trap keys that generate a char message and do special processing, the application must subclass the list box and trap both the WM_KEYDOWN and WM_CHAR messages, and process the messages appropriately in the subclass procedure.

NOTE: This discussion is for regular list boxes that are created with the LBS_WANTKEYBOARDINPUT style. If the list box is owner draw, the application must process the WM_CHARTOITEM message. For more information on the WM_CHARTOITEM message, refer to the SDK documentation.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95 listbox
KBCategory: kbprg
KBSubcategory: UsrCtl

Using UnregisterClass When Removing Custom Control Class

PSS ID Number: Q67248
Authored 28-Nov-1990 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

In the Microsoft Windows environment, when no running application requires a custom control class, the class should be removed from memory to free the system resources it uses.

If an application registers a control class for temporary use, the application should use the UnregisterClass function when the control is no longer needed. If the application is terminated, Windows automatically removes any classes that the application registered; therefore, explicit use of UnregisterClass is not required. However, pairing calls to the RegisterClass and UnregisterClass functions is a good programming practice.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrCtl

Using volatile to Prevent Optimization of try/except

PSS ID Number: Q91149
Authored 29-Oct-1992 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

The following is an example of a valid optimization that may take programmers by surprise.

1. A variable (temp) used only within the try-except body is declared outside it, and therefore is global with respect to the try.

2. Assignment to the variable (temp) is in the program only for a possible side effect of doing a read memory access through the pointer.

MORE INFORMATION

For example:

```
VOID
puRoutine( PULONG pu )
{
   ...
   ULONG temp;       // Just for probing
   ...
   try {
      temp = *pu;    // See if pu is a valid argument
   }

   except {
      // Handle exception
   }
}
```

The compiler optimizes and eliminates the entire try-except statement because temp is not used later.

If the value of temp were used globally, the compiler should treat the assignment to temp as volatile and do the assignment immediately even if it is overwritten later in the body of the try. The reasoning is that, at almost any point in the try body, control may jump to the except (or an exception filter). Presumably the programmer accessing the variable outside the try wants to get the current (most recently assigned) value.

The way to prevent the compiler from performing the optimization is:

```
   temp = (volatile ULONG) *pu;
```

If a temporary variable is not needed, given the example, the read access should still be specified as volatile, for example:

```
   *(volatile PULONG) pu;
```

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseExcept

Using VxDs and Software Interrupts Under Win32s

PSS ID Number: Q105760
Authored 24-Oct-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.0, 1.1, and 1.2

Calling VxDs directly from a Win32-based application is not supported under Win32s. Win32s does not support the VxD interfaces, so the call is handled by the underlying Windows system. The Win32-based application runs with 32-bit stack and code sections, but Windows expects only 16-bit segments. Therefore, the calls to the VxD cannot be handled by Windows as expected.

To call software interrupts (such as Interrupt 2F) from a Win32-based application running under Windows 3.1 via Win32s, place the call in a 16-bit dynamic-link library (DLL) and use the Universal Thunks to access this DLL. To convert the addresses between segmented and linear address, use UTSelectorOffsetToLinear() and UTLinearToSelectorOffset().

Additional reference words: 1.00 1.10 1.20
KBCategory: kbprg
KBSubcategory: W32s

Using Windows Sockets Under Win32s and WOW

PSS ID Number: Q105757
Authored 24-Oct-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.1 and 1.2

SUMMARY

Win32s versions 1.1 and later provide a thunking layer for Windows Sockets. A 16-bit Windows Sockets 1.1 package must be installed on the Windows machine. Otherwise, the system will report that WINSOCK.DLL was not found. Windows NT provides a versions 1.0- and 1.1-compliant WOW (Windows on Win32) thunking layer.

MORE INFORMATION

There are a number of vendors that sell Windows Sockets packages. Windows Sockets support is available from Microsoft for LAN Manager version 2.2 for MS-DOS and Windows 3.1 at no additional cost. Similar support is also being shipped in "Microsoft TCP/IP for Windows For Workgroups" and the "Microsoft Network Client."

If you have LAN Manager with Microsoft TCP/IP, you can pick up everything you need from ftp.microsoft.com. Or, call Microsoft Sales at (800) 426-9400.

The following is information on how to subscribe to an Internet mailing list for Windows Socket programming as of 07/94:

Send mail to majordomo@mailbag.intel.com with a body that has

SUBSCRIBE WINSOCK <your_full_internet_email_address>.

If you want to be on the winsock hackers mailing list (for implementors of Windows sockets), use the following body in a separate piece of email.

SUBSCRIBE WINSOCK-HACKERS <your_full_internet_email_address>.

Other lists available include:

winsnmp
winsock-2
ws2-app-review-board
ws2-appletalk
ws2-conn-oriented-media
ws2-decnet
ws2-generic-api-ext
ws2-ipx-spx

```
  ws2-name-resolution
  ws2-oper-framework
  ws2-osi
  ws2-spec-clarif
  ws2-tcp-ip
  ws2-transp-review-board
  ws2-wireless

Use the command 'info <list>' to get more information about a specific
list.

Additional reference words: 1.10 1.20
KBCategory: kbprg
KBSubcategory: W32s
```

Validating User Account Passwords Under Windows NT

PSS ID Number: Q98891
Authored 18-May-1993 Last modified 23-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1, 3.5, and 3.51

Windows NT stores account names and passwords in the security accounts manager (SAM) database. Windows NT checks this database to validate passwords when users log on.

In Windows NT 3.1 and 3.5, there is no nonprivileged service that takes a user name and a password and returns an indication of whether or not the user account password is valid. There is a privileged service that handles this password validation; it is for use by logon processes such as winlogon.

Windows NT 3.51 introduces new Win32 APIs for logon support:

```
LogonUser
ImpersonateLoggedOnUser
CreateProcessAsUser
```

MORE INFORMATION

The SAM application programming interface (API) functions were not exposed due to their changing nature. Microsoft is working on a developer's kit that will provide guidelines and tutorial information about most of the security API functions, including the SAM APIs.

Exposing the SAM API will not compromise security because the passwords are encrypted within SAM; they are one-way encrypted such that not even SAM can decrypt them. Even a dictionary attack (encrypt an entire dictionary and see if any of the words match) would not be easy, because there is no SAM API function that will read the encrypted password.

Additional reference words: 3.10 3.50 non-privileged
KBCategory: kbprg
KBSubcategory: BseSecurity

Validating User Accounts (Impersonation)

PSS ID Number: Q96005
Authored 03-Mar-1993 Last modified 23-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1, 3.5, and 3.51

SUMMARY

Some applications need the ability to execute processes in the context of another user. This impersonation restricts (or expands) the permissions of the account in which the application was executed (file access, permission to change system time, permission to shut down the system, and so forth).

For example, an administrator executes a network server program that allows remote users to log on to the system and perform actions, as if they were logged on to the system locally. Because the administrator initiated the server program and is currently logged on, all actions the server program performs will be in the security context of the administrator. If a guest user logs on remotely, he/she will have all the permissions the administrator account has.

With the Win32 API under Windows NT 3.1 and 3.5, impersonating a remote client is possible only via the ImpersonateDDEClientWindow(), ImpersonateNamedPipeClient() and RpcImpersonateClient() APIs.

Windows NT 3.51 introduces new Win32 APIs (Logon Support APIs) to deal with this problem:

LogonUser
ImpersonateLoggedOnUser
CreateProcessAsUser

MORE INFORMATION

For versions of Windows NT prior to 3.51

A common application of impersonation is network server programs (daemons). For example, a remote login daemon needs a user to be able to to log in to a remote host and have the host impose all restrictions of the client login account.

If the daemon is using named pipes, dynamic data exchange (DDE), or a remote procedure call (RPC) (using the named pipes transport), the client account may be impersonated on the server daemon, which will impose all the restrictions of the client's user account.

Using other network interfaces (such as Windows sockets--network

programming interfaces), security cannot be monitored by the system. A workaround would be to impose password-level security on "login" to the application. The passwords would be maintained by the application in a private accounts database. However, none of the user actions are performed in the security context of the actual client user's account. Therefore, after the server/daemon has validated the client, the server must be careful to only perform actions on behalf of the client that the server knows the client should be allowed to do.

Another option is to create a network share with restricted access. The WNetAddConnection2() API can verify access to this system resources [disk or printer network resource (share)]. If the network share was set up to allow access by a restricted group of people, the WNetAddConnection2() could validate actual user accounts, maintained by Windows NT. As with the previous option, the daemon must be careful to perform only restricted actions on behalf of the client. This option could be used for file server daemons.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseSecurity

Value Returned by GetWindowLong(hWnd, GWL_STYLE)

PSS ID Number: Q83366
Authored 09-Apr-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

GetWindowLong(hWnd, GWL_STYLE) returns the window style information stored at the GWL_STYLE offset of the window data structure identified by hWnd. This offset contains the current state of the window, rather than the style specified when the window was created.

Windows can set and clear the following styles over the lifetime of a window: WS_CLIPSIBLINGS, WS_DISABLED, WS_HSCROLL, WS_MAXIMIZE, WS_MINIMIZE, WS_SYSMENU (for MDI child windows), WS_THICKFRAME, WS_VISIBLE, and WS_VSCROLL. Windows will not dynamically set or clear any of the other styles. An application can modify the style state at the GWL_STYLE offset at any time by calling SetWindowLong(hWnd, GWL_STYLE, dwNewLong), but Windows will not be aware that the style has changed. Windows maintains some internal flags on the window style and may use these rather than checking the GWL_STYLE offset of the window data structure.

GetWindowLong(hWnd, GWL_STYLE) returns a LONG value, which contains the currently active styles combined by the Boolean OR operator. An application can change the information stored at the GWL_STYLE offset by calling SetWindowLong(hWnd, GWL_STYLE, dwNewLong).

MORE INFORMATION

The following table lists the windows styles that Windows updates throughout the life span of a window. In the Change column below, an "S" indicates that the style is set and a "C" indicates that the style is cleared. The Where column lists the source module involved. Windows will change a window's styles as follows:

Style	Change	Window	Where	Why
WS_CLIPSIBLINGS	S	Overlapped	WMCREATE.C	Set on creation.
WS_CLIPSIBLINGS	S	Popup	WMCREATE.C	Set on creation.

WS_DISABLED	S & C	Any	WMACT.C	Set or cleared when EnableWindow function disables or enables window.
WS_HSCROLL	S & C	SB_HORZ	WINSBCTL.C	Set or cleared as scroll bar range changed (for window scroll bar, not the SCROLL BAR class).
WS_HSCROLL	S & C	SB_HORZ	SBRARE.C	Set or cleared in ShowScrollBar (for window scroll bar, not the SCROLL BAR class).
WS_HSCROLL	S & C	List boxes	LBOXCTL1.C	Set if scroll bar required to see contents.
WS_HSCROLL	S & C	MDI frame	MDIWIN.C	Set if required to see children.
WS_MAXIMIZE	C	Any	WMCREATE.C	On creation (reset immediately in WMMINMAX.C).
WS_MAXIMIZE	C	Any	WMMOVE.C	Cleared if window resized.
WS_MAXIMIZE	S & C	Any	WMMINMAX.C	Set if window maximized, cleared if no longer maximized.
WS_MINIMIZE	C	Any	WMCREATE.C	On creation (reset immediately in WMMINMAX.C).
WS_MINIMIZE	C	Any	WMMOVE.C	Cleared if window resized.
WS_MINIMIZE	S & C	any	WMMINMAX.C	Set if window minimized, cleared if no longer minimized.
WS_SYSMENU	S & C	MDI child	MDIMENU.C	Cleared if child is maximized and uses frame menu. Set when child no longer maximized.
WS_THICKFRAME	S & C	Any	WINSBCTL.C	State changed for only a few instructions during painting.
WS_VISIBLE	C	Any	WMCREATE.C	Cleared and then reset by call to ShowWindow.

WS_VISIBLE	S & C	Any	WMSHOW.C	Cleared if window hidden in ShowWindow, set if shown.
WS_VISIBLE	S & C	Any	WMSWP.C	Cleared if window hidden in ShowWindow, set if shown.
WS_VISIBLE	C	MDI client	MDIWIN.C	Cleared when current MDI child is maximized, and new child is activated. Immediately reset with call to ShowWindow.
WS_VISIBLE	S	Desktop	INLOADW.C	Ensure desktop visible.
WS_VISIBLE	S & C	Any	MSDWP.C	Set or cleared when DefWindowProc receives WM_SETREDRAW message to turn drawing on or off, respectively.
WS_VISIBLE	S & C	MDI client	MDIWIN.C	Cleared and then immediately reset to optimize painting.
WS_VSCROLL	S & C	SB_VERT	WINSBCTL.C	Set or cleared as scroll bar range changed (for window scroll bar, not the SCROLL BAR class).
WS_VSCROLL	S & C	List boxes	LBOXCTL1.C	Set if scroll bar required to see entire contents.
WS_VSCROLL	S & C	MDI frame	MDIWIN.C	Set if required to see children.
WS_VSCROLL	S & C	SB_VERT	SBRARE.C	Set/cleared in ShowScrollBar (for window scroll bar, not the SCROLL BAR class).

In addition to the information above, the GetWindowLong function always reports some style bits to be clear, as follows:

- Combo boxes always report the following styles as clear:

  CBS_HASSTRINGS, CBS_SORT, WS_BORDER, WS_HSCROLL, and WS_VSCROLL

- All edit controls report the WS_BORDER style clear.

- Multiline edit controls report the WS_HSCROLL style clear if the control contains centered or right-justified text.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95

KBCategory: kbprg
KBSubcategory: UsrWndw

Various Ways to Access Submenus and Menu Items

PSS ID Number: Q71454
Authored 18-Apr-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

In calls to Microsoft Windows functions that create, modify, and destroy menus, an application can access an individual menu item by either its position or its item ID. A pop-up menu must be accessed by its position because it does not have a menu-item ID.

Specifically, when an application calls the EnableMenuItem() function to enable, disable, or dim (gray) an individual menu item, the application can specify either the MF_BYPOSITION or the MF_BYCOMMAND flag in the wEnable parameter. When the application calls EnableMenuItem() to access a pop-up menu, it must specify the MF_BYPOSITION flag.

The information below provides examples of the following:

- Retrieving a menu handle for a submenu
- Accessing a submenu
- Accessing a menu item

MORE INFORMATION

The following resource-file menu template provides the basis for the source code examples in this article. The template describes a top-level menu with two pop-up submenus. One of the submenus contains a third, nested submenu.

```
GenericMenu MENU
BEGIN
   POPUP "&Help"
   BEGIN
      MENUITEM "&About Generic...", IDM_ABOUT
   END

   POPUP "&Test"
   BEGIN
      POPUP "&Nested"
      BEGIN
         MENUITEM "&1 Beep", IDM_1BEEP
         MENUITEM "&2 Beeps", IDM_2BEEPS
      END
   END
END
```

Retrieving the Handle to a Submenu

Code such as the following can be used to obtain handles to the menus:

```
HMENU hMainMenu, hHelpPopup, hTestPopup, hNestedPopup;

<other program lines>

hMainMenu = GetMenu(hWnd);
hHelpPopup = GetSubMenu(hMainMenu, 0);
hTestPopup = GetSubMenu(hMainMenu, 1);
hNestedPopup = GetSubMenu(hTestPopup, 0);
```

The second parameter of the GetSubMenu() function, nPos, is the position of the desired submenu. Positions are numbered starting at zero for the first menu item.

Disabling a Submenu

The following call disables and dims the nested pop-up menu:

```
EnableMenuItem (hTestPopup, 0, MF_BYPOSITION | MF_GRAYED);
```

The following call disables and dims the Test pop-up menu:

```
EnableMenuItem (hMainMenu, 1, MF_BYPOSITION | MF_GRAYED);
```

The second parameter of the EnableMenuItem() function, wIDEnabledItem, is the position of the submenu. As above, positions are numbered starting at zero. Note that the call must specify the MF_BYPOSITION flag because a pop-up menu does not have a menu-item ID.

Disabling a Menu Item

The 1 Beep menu item can be disabled and dimmed by using any one of the following calls:

```
EnableMenuItem(hMainMenu, IDM_1BEEP, MF_BYCOMMAND | MF_GRAYED);
EnableMenuItem(hTestPopup, IDM_1BEEP, MF_BYCOMMAND | MF_GRAYED);
EnableMenuItem(hNestedPopup, IDM_1BEEP, MF_BYCOMMAND | MF_GRAYED);
EnableMenuItem(hNestedPopup, 0, MF_BYPOSITION | MF_GRAYED);
```

A menu item can be specified by either by its menu-item ID value (using the MF_BYCOMMAND flag) or by its position (using the MF_BYPOSITION) flag. If the application specifies the menu-item ID value, Windows must walk the menu structure and search for a menu item with the correct ID. This implies the each menu-item ID value must be unique for a given menu.

Other Windows Menu Functions

Although the EnableMenuItem() function is used in the example above, the same general approach is used for all Windows menu functions; access pop-up menus by position, and access menu items by position or menu-item ID.

Additional reference words: 3.00 3.10 3.50 4.00 95 dimmed unavailable
KBCategory: kbprg
KBSubcategory: UsrMen

Video for Windows 1.1--Automatic Window Shift Problem

PSS ID Number: Q118390
Authored 17-Jul-1994 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 SDK, version 4.0

When a video playback window is created using MCI, the Video for Windows (VfW) system software may shift the window a small distance vertically or horizontally from the original location.

The video software does this to align the window on even-byte boundaries in video memory for optimal playback speed. When the window is located on a byte boundary, the video software is not required to perform bit-intensive calculations during video playback, increasing the playback speed.

This behavior is particular to Video for Windows and cannot be controlled by the application being affected.

Additional reference words: 3.10 4.00 95
KBCategory: kbmm kbprg
KBSubcategory: MMVideo

Viewing Globals Out of Context in WinDbg

PSS ID Number: Q105583
Authored 20-Oct-1993 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

When viewing global variables (either with the ? command or via the Watch window) and the variables go out of context, their values become:

```
CXX0017 Error: symbol not found
```

An example of this is when a common dialog box is open in the application. If you break into an application that is inside COMDLG32.DLL and try to do a ?gVar, where gVar is a global variable in the application, WinDbg will not find the symbol because the context is wrong. To view the value of gVar in MYAPP, use the following:

```
?{,,myapp}gVar
```

WinDbg will then have no trouble locating the symbolic information.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsWindbg

Virtual Memory and Win32s

PSS ID Number: Q124137
Authored 19-Dec-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.1, 1.15, 1.15a, and 1.2

SUMMARY

This article discusses virtual memory issues under Win32s. These issues include how memory is managed, file mappings, and VirtualAlloc().

MORE INFORMATION

Memory Management

Win32s does not manage the virtual memory by itself. Win32s sits on top of the Win386 virtual memory manager (VMM). The Win32s VxD and Win386 must cooperatively manage the memory.

Windows sets the amount of virtual memory (VM) at boot time, based on the amount of free physical RAM. Windows is also responsible for managing the physical pages.

The pages that map to the Win32-based application's code and data are reserved at application initialization time. This decreases the available virtual memory but not necessarily the available RAM. In 16-bit Windows-based applications, the selectors are initialized at initialization time, but they are marked as discarded and are loaded only when the segments are touched. This may lead you to think that a Win32 version of you application takes significantly more memory than its 16-bit Windows version. However, in reality, the available memory drops more when the Win32-based application is loaded, even though the actual RAM usage during execution may be lower.

NOTE: Code pages are never backed by the .EXE file, but are always backed by the pagefile.

File Mappings

Because of the way memory is managed, you cannot have a file mapping that is larger than the amount of virtual memory available on Win32s. This works fine on Windows NT and Windows 95. Win32s allocates regular virtual memory for the memory mapped section even though it does not need swap space, and the amount of VM set by Windows is too small to use for mapping large files.

VirtualAlloc()

VirtualAlloc() reserves or commits pages in virtual memory. When virtual memory is allocated (committed), the page is still not present. Touching the page will make it present and initialized to zero. Unlike Windows NT, some of the address space on Win32s is not allocated by VirtualAlloc(). You can use VirtualQuery() for every address in the USER address space, as reported by GetSystemInfo().

VirtualAlloc() allocates private memory, so one process cannot commit or free memory reserved by another process. This is true for all Win32 platforms. However, on Win32s, the memory is still accessible by all processes because there is a shared address space provided by Windows.

NOTE: A "not enough memory" error occurs if you reserve memory using VirtualAlloc() in one application and then try to commit the memory from a second application. This happens because the call to commit from the second application is interpreted as a call to reserve. You need to commit at an address that is not available such as an address already reserved by the first application.

Additional reference words: 1.10 1.20
KBCategory: kbprg
KBSubcategory: W32s

VirtualLock() Only Locks Pages into Working Set

PSS ID Number: Q94996
Authored 28-Jan-1993 Last modified 03-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

VirtualLock() causes pages to be locked into an application's working set (virtual memory); it does not lock them absolutely into physical memory. VirtualLock() essentially means "this page is always part of the process's working set."

The system is free to swap out any virtually locked pages if it swaps out the whole process. And when the system swaps the process back in, the virtually locked pages (similar to any virtual pages) may end up residing in different real pages.

It is wise to use VirtualLock() very sparingly because it reduces the flexibility of the system.

Depending upon memory demands on the system, the memory manager may vary the number of pages a process can lock. Under typical conditions you can expect to be able to VirtualLock() approximately 28 to 32 pages.

In Windows NT 3.5, you can use SetProcessWorkingSize() to increase the size of the working set, and therefore increase the number of pages that VirtualLock() can lock.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseMm

Watching Local Variables That Are Also Globally Declared

PSS ID Number: Q98288
Authored 02-May-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5. 3.51, and 4.0

SUMMARY

Consider debugging the following program in WinDbg:

```
int x = 1;
int y = 2;

void main()
{
   int x = 2;
   x++;
   y++;
}
```

Notice that there is a global variable x and a local variable x.

Before you step into main, if you set watchpoints on x and y, the Watch window will display a value for y but for x will say "Expression cannot be evaluated." To see the value for x, use ::x and x will evaluate to the local x in main once you've stepped into main.

MORE INFORMATION

When debugging an application, the X86 C++ evaluator is loaded. Given this, you can use the scope resolution operator in a watch statement to view a hidden global variable. Without the use of the scope resolution operator, there is no way (short of watching it in a memory window) to watch a hidden global variable.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsWindbg

When to Use Synchronous Socket Handles & Blocking Hooks

PSS ID Number: Q131623
Authored 14-Jun-1995 Last modified 15-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

By default, all socket handles are opened as overlapped handles so that asynchronous I/O can be performed on them. However, in many situations you may find it preferable to have nonoverlapped (synchronous) socket handles.

For example, only nonoverlapped handles can be used with the C run-time libraries or used as standard I/O handles for a process. Under Windows NT and Windows 95, the SO_OPENTYPE socket option allows an application to open non-overlapped socket handles.

MORE INFORMATION

There are some Windows Sockets features that you cannot use with synchronous sockets. Here is an extract from the Winsock Help file:

> The WSAAsyncSelect call cannot be used with synchronous sockets and will fail with the error WSAEINVAL. It is also not possible to set the SO_SNDTIMEO and SO_RCVTIMEO socket options on synchronous sockets; setsockopt with these options on synchronous sockets fails with WSAEINVAL as well.

Due to the non-preemptive nature of Windows version 3.1 and Windows for Workgroups version 3.11, the Winsock specification details a mechanism by which a Winsock application can "yield" processor time. For more information, please search for WSASetBlockingHook() in the Winsock Help file.

NOTE: Use of a blocking hook is not recommended on a 32-bit platform. If a 32-bit application chooses to install a blocking hook, the blocking hook will be disabled if the application is run under Windows NT, but it will remain enabled if the application is run under Windows 95.

REFERENCES

Online winsock help file

Additional reference words: 3.10 3.50 3.51 4.00
KBCategory: kbnetwork
KBSubcategory: NtwkWinsock

Where to Get the Microsoft SNMP Headers and Libraries

PSS ID Number: Q127902
Authored 21-Mar-1995 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.5, 3.51, and 4.0
- Microsoft Windows for Workgroups SDK, version 3.11

The 32-bit SNMP headers and libraries are available in both the Microsoft Win32 SDK and Microsoft Visual C++ version 2.0 and later. These files are for use with the Microsoft implementation of SNMP. They will not work with other implementations because Microsoft does not conform to the WINSNMP standard of management APIs.

The headers and libraries for the SNMP agent (parts of SNMP.H and SNMP.LIB) can be used with Windows NT and Windows 95. The headers and libraries for the Management APIs (parts of SNMP.H, MGMTAPI.H, parts of SNMP.LIB, and MGMTAPI.LIB) are for use with Windows NT only.

Microsoft offers no 16-bit SNMP for Windows for Workgroups, however, there are other companies that do offer SNMP for Windows for Workgroups. Here are two companies that we know of that offer 16-bit SNMP:

Company: NetManage
10725 N. De Anza Blvd.
Cupertino, CA 95014
Product: Chameleon Utilities
Phone: (408) 973-7171
Fax: (408) 257-6405

FTP Software
Corporate Headquarters
2 High Street
North Andover, MA 01845-2620
Phone: (508) 685-4000
Fax: (508) 794-4488

The third-party products discussed here are manufactured by vendors independent of Microsoft; we make no warranty, implied or otherwise, regarding these products' performance or reliability.

Additional reference words: 3.50 4.00 95
KBCategory: kbprg kb3rdparty
KBSubcategory: NtwkSnmp

Which Windows NT (Server or Workstation) Is Running?

PSS ID Number: Q124305
Authored 27-Dec-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

You can determine which variant of Windows NT (Windows NT Server or Windows NT Workstation) is running by using the technique described in this article. Then you can use this information to execute code based on which variant is running.

MORE INFORMATION

To find out which product is currently running, you need to determine the value of the following registry entry:

```
HKEY_LOCAL_MACHINE\SYSTEM\
CurrentControlSet
Control\
ProductOptions
```

Use the following table to determine which product is running:

ProductType	Product
WINNT	Windows NT Workstation is running
SERVERNT	Windows NT Server is running
LANMANNT	Windows NT Advanced Server is running

The sample code creates a WhichNTProduct() function to indicate whether Windows NT Server or Windows NT Workstation in currently running. The following table gives the meaning for each return value:

Return Value	Meaning
RTN_SERVER	Windows NT Server is running
RTN_WORKSTATION	Windows NT Workstation is running
RTN_NTAS	Windows NT Advanced Server is running
RTN_UNKNOWN	Unknown product type was encountered
RTN_ERROR	Error occurred

To get extended error information, call GetLastError(). Some error checking is omitted, for brevity.

Sample Code

```
#define RTN_UNKNOWN 0
#define RTN_SERVER 1
#define RTN_WORKSTATION 2
#define RTN_NTAS 3
#define RTN_ERROR 13

unsigned int WhichNTProduct(void)
{
    #define MY_BUFSIZE 32 // arbitrary. Use dynamic allocation
    HKEY hKey;
    char szProductType[MY_BUFSIZE];
    DWORD dwBufLen=MY_BUFSIZE;

    if(RegOpenKeyEx(HKEY_LOCAL_MACHINE,
                    "SYSTEM\\CurrentControlSet\\Control\\ProductOptions",
                    0,
                    KEY_EXECUTE,
                    &hKey) != ERROR_SUCCESS) return(RTN_ERROR);

    if(RegQueryValueEx(hKey,
                    "ProductType",
                    NULL,
                    NULL,
                    szProductType,
                    &dwBufLen) != ERROR_SUCCESS) return(RTN_ERROR);

    RegCloseKey(hKey);

    // check product options, in order of likelihood
    if(lstrcmpi("WINNT", szProductType) == 0) return(RTN_WORKSTATION);
    if(lstrcmpi("SERVERNT", szProductType) == 0) return(RTN_SERVER);
    if(lstrcmpi("LANMANNT", szProductType) == 0) return(RTN_NTAS);

    // else return Unknown
    return(RTN_UNKNOWN);
}
```

Additional reference words: 3.10 3.50
KBCategory: kbprg kbcode
KBSubcategory: BseMisc

Why LoadLibraryEx() Returns an HINSTANCE

PSS ID Number: Q102128
Authored 28-Jul-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

In the Win32 Help files, LoadLibrary() is typed to return a HANDLE, while LoadLibraryEx() is prototyped to return a HINSTANCE.

An HINSTANCE return from LoadLibraryEx() is useful because processes that load dynamic-link libraries (DLLs) do not necessarily want the overhead of having to page in code for a DllEntryPoint routine when the DLL does not need to initialize information. This is especially useful when you have multiple threads that attach to already loaded DLLs. In this case, you may want to not implicitly load via LoadLibrary() and instead use LoadLibraryEx() to explicitly load without having to page in the code for every attach.

LoadLibraryEx() is also useful if you want to retrieve resources from a DLL or an EXE. In this case, you would use LoadLibraryEx() to load the module you want into your address space, without executing DLLEntryPoint, and then use the resource application programming interfaces (APIs) to access the data.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseDll

Win32 Drag and Drop Server

PSS ID Number: Q105530
Authored 20-Oct-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, 4.0

The supported method for creating a drag and drop server is to use OLE (Object Linking and Embedding) version 2.0. This works on Windows, Windows NT, and Windows 95 and will work on future versions of these operating systems.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDnd

Win32 Equivalents for C Run-Time Functions

PSS ID Number: Q99456
Authored 30-May-1993 Last modified 13-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

Many of the C Run-time functions have direct equivalents in the Win32 application programming interface (API). This article lists the C Run-time functions by category with their Win32 equivalents or the word "none" if no equivalent exists.

MORE INFORMATION

NOTE: the functions that are followed by an asterisk (*) are part of the 16-bit C Run-time only. Functions that are unique to the 32-bit C Run-time are listed separately in the last section. All other functions are common to both C Run-times.

Buffer Manipulation

_memccpy	none
memchr	none
memcmp	none
memcpy	CopyMemory
_memicmp	none
memmove	MoveMemory
memset	FillMemory, ZeroMemory
_swab	none

Character Classification

isalnum	IsCharAlphaNumeric
isalpha	IsCharAlpha, GetStringTypeW (Unicode)
__isascii	none
iscntrl	none, GetStringTypeW (Unicode)
__iscsym	none
__iscsymf	none
isdigit	none, GetStringTypeW (Unicode)
isgraph	none
islower	IsCharLower, GetStringTypeW (Unicode)
isprint	none
ispunct	none, GetStringTypeW (Unicode)
isspace	none, GetStringTypeW (Unicode)
isupper	IsCharUpper, GetStringTypeW (Unicode)
isxdigit	none, GetStringTypeW (Unicode)
__toascii	none

tolower	CharLower
_tolower	none
toupper	CharUpper
_toupper	none

Directory Control

_chdir	SetCurrentDirectory
_chdrive	SetCurrentDirectory
_getcwd	GetCurrentDirectory
_getdrive	GetCurrentDirectory
_mkdir	CreateDirectory
_rmdir	RemoveDirectory
_searchenv	SearchPath

File Handling

_access	none
_chmod	SetFileAttributes
_chsize	SetEndOfFile
_filelength	GetFileSize
_fstat	See Note 5
_fullpath	GetFullPathName
_get_osfhandle	none
_isatty	GetFileType
_locking	LockFileEx
_makepath	none
_mktemp	GetTempFileName
_open_osfhandle	none
remove	DeleteFile
rename	MoveFile
_setmode	none
_splitpath	none
_stat	none
_umask	none
_unlink	DeleteFile

Creating Text Output Routines

_displaycursor*	SetConsoleCursorInfo
_gettextcolor*	GetConsoleScreenBufferInfo
_gettextcursor*	GetConsoleCursorInfo
_gettextposition*	GetConsoleScreenBufferInfo
_gettextwindow*	GetConsoleWindowInfo
_outtext*	WriteConsole
_scrolltextwindow*	ScrollConsoleScreenBuffer
_settextcolor*	SetConsoleTextAttribute
_settextcursor*	SetConsoleCursorInfo
_settextposition*	SetConsoleCursorPosition
_settextwindow*	SetConsoleWindowInfo
_wrapon*	SetConsoleMode

Stream Routines

clearerr	none

fclose	CloseHandle
_fcloseall	none
_fdopen	none
feof	none
ferror	none
fflush	FlushFileBuffers
fgetc	none
_fgetchar	none
fgetpos	none
fgets	none
_fileno	none
_flushall	none
fopen	CreateFile
fprintf	none
fputc	none
_fputchar	none
fputs	none
fread	ReadFile
freopen (std handles)	SetStdHandle
fscanf	none
fseek	SetFilePointer
fsetpos	SetFilePointer
_fsopen	CreateFile
ftell	SetFilePointer (check return value)
fwrite	WriteFile
getc	none
getchar	none
gets	none
_getw	none
printf	none
putc	none
putchar	none
puts	none
_putw	none
rewind	SetFilePointer
_rmtmp	none
scanf	none
setbuf	none
setvbuf	none
_snprintf	none
sprintf	wsprintf
sscanf	none
_tempnam	GetTempFileName
tmpfile	none
tmpnam	GetTempFileName
ungetc	none
vfprintf	none
vprintf	none
_vsnprintf	none
vsprintf	wvsprintf

Low-Level I/O

_close	_lclose, CloseHandle
_commit	FlushFileBuffers

_creat	_lcreat, CreateFile
_dup	DuplicateHandle
_dup2	none
_eof	none
_lseek	_llseek, SetFilePointer
_open	_lopen, CreateFile
_read	_lread, ReadFile
_sopen	CreateFile
_tell	SetFilePointer (check return value)
_write	_lread

Console and Port I/O Routines

_cgets	none
_cprintf	none
_cputs	none
_cscanf	none
_getch	ReadConsoleInput
_getche	ReadConsoleInput
_inp	none
_inpw	none
_kbhit	PeekConsoleInput
_outp	none
_outpw	none
_putch	WriteConsoleInput
_ungetch	none

Memory Allocation

_alloca	none
_bfreeseg*	none
_bheapseg*	none
calloc	GlobalAlloc
_expand	none
free	GlobalFree
_freect*	GlobalMemoryStatus
_halloc*	GlobalAlloc
_heapadd	none
_heapchk	none
_heapmin	none
_heapset	none
_heapwalk	none
_hfree*	GlobalFree
malloc	GlobalAlloc
_memavl	GlobalMemoryStatus
_memmax	GlobalMemoryStatus
_msize*	GlobalSize
realloc	GlobalReAlloc
_set_new_handler	none
_set_hnew_handler*	none
_stackavail*	none

Process and Environment Control Routines

abort	none

assert	none
atexit	none
_cexit	none
_c_exit	none
_exec functions	none
exit	ExitProcess
_exit	ExitProcess
getenv	GetEnvironmentVariable
_getpid	GetCurrentProcessId
longjmp	none
_onexit	none
perror	FormatMessage
_putenv	SetEnvironmentVariable
raise	RaiseException
setjmp	none
signal (ctrl-c only)	SetConsoleCtrlHandler
_spawn functions	CreateProcess
system	CreateProcess

String Manipulation

strcat, wcscat	lstrcat
strchr, wcschr	none
strcmp, wcscmp	lstrcmp
strcpy, wcscpy	lstrcpy
strcspn, wcscspn	none
_strdup, _wcsdup	none
strerror	FormatMessage
_strerror	FormatMessage
_stricmp, _wcsicmp	lstrcmpi
strlen, wcslen	lstrlen
_strlwr, _wcslwr	CharLower, CharLowerBuffer
strncat, wcsncat	none
strncmp, wcsncmp	none
strncpy, wcsncpy	none
_strnicmp, _wcsnicmp	none
_strnset, _wcsnset	FillMemory, ZeroMemory
strpbrk, wcspbrk	none
strrchr, wcsrchr	none
_strrev, _wcsrev	none
_strset, _wcsset	FillMemory, ZeroMemory
strspn, wcsspn	none
strstr, wcsstr	none
strtok, wcstok	none
_strupr, _wcsupr	CharUpper, CharUpperBuffer

MS-DOS Interface

_bdos*	none
_chain_intr*	none
_disable*	none
_dos_allocmem*	GlobalAlloc
_dos_close*	CloseHandle
_dos_commit*	FlushFileBuffers
_dos_creat*	CreateFile

_dos_creatnew*	CreateFile
_dos_findfirst*	FindFirstFile
_dos_findnext*	FindNextFile
_dos_freemem*	GlobalFree
_dos_getdate*	GetSystemTime
_dos_getdiskfree*	GetDiskFreeSpace
_dos_getdrive*	GetCurrentDirectory
_dos_getfileattr*	GetFileAttributes
_dos_getftime*	GetFileTime
_dos_gettime*	GetSystemTime
_dos_getvect*	none
_dos_keep*	none
_dos_open*	OpenFile
_dos_read*	ReadFile
_dos_setblock*	GlobalReAlloc
_dos_setdate*	SetSystemTime
_dos_setdrive*	SetCurrentDirectory
_dos_setfileattr*	SetFileAttributes
_dos_setftime*	SetFileTime
_dos_settime*	SetSystemTime
_dos_setvect*	none
_dos_write*	WriteFile
_dosexterr*	GetLastError
_enable*	none
_FP_OFF*	none
_FP_SEG*	none
_harderr*	See Note 1
_hardresume*	See Note 1
_hardretn*	See Note 1
_int86*	none
_int86x*	none
_intdos*	none
_intdosx*	none
_segread*	none

Time

asctime	See Note 2
clock	See Note 2
ctime	See Note 2
difftime	See Note 2
_ftime	See Note 2
_getsystime	GetLocalTime
gmtime	See Note 2
localtime	See Note 2
mktime	See Note 2
_strdate	See Note 2
_strtime	See Note 2
time	See Note 2
_tzset	See Note 2
_utime	SetFileTime

Virtual Memory Allocation

_vfree*	See Note 3

_vheapinit*	See Note 3
_vheapterm*	See Note 3
_vload*	See Note 3
_vlock*	See Note 3
_vlockcnt*	See Note 3
_vmalloc*	See Note 3
_vmsize*	See Note 3
_vrealloc*	See Note 3
_vunlock*	See Note 3

32-Bit C Run Time

_beginthread	CreateThread
_cwait	WaitForSingleObject w/ GetExitCodeProcess
_endthread	ExitThread
_findclose	FindClose
_findfirst	FindFirstFile
_findnext	FindNextFile
_futime	SetFileTime
_get_osfhandle	none
_open_osfhandle	none
_pclose	See Note 4
_pipe	CreatePipe
_popen	See Note 4

NOTE 1: The _harderr functions do not exist in the Win32 API. However, much of their functionality is available through structured exception handling.

NOTE 2: The time functions are based on a format that is not used in Win32. There are specific Win32 time functions that are documented in the Help file.

NOTE 3: The virtual memory functions listed in this document are specific to the MS-DOS environment and were written to access memory beyond the 640K of RAM available in MS-DOS. Because this limitation does not exist in Win32, the standard memory allocation functions should be used.

NOTE 4: While _pclose() and _popen() do not have direct Win32 equivalents, you can (with some work) simulate them with the following calls:

_popen	CreatePipe
	CreateProcess
_pclose	WaitForSingleObject
	CloseHandle

NOTE 5: GetFileInformationByHandle() is the Win32 equivalent for the _fstat() C Run-time function. However, GetFileInformationByHandle() is not supported by Win32s version 1.1. It is supported in Win32s 1.2. GetFileSize(), GetFileAttributes(), GetFileTime(), and GetFileTitle() are supported by Win32s 1.1 and 1.2.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg

KBSubcategory: BseMisc

Win32 Graphical Setup Over Network Drives

PSS ID Number: Q98838
Authored 13-May-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

The graphical setup for the Win32 Software Development Kit (SDK) does not include network drives in the list of drives to choose from, however, the setup program does support installing to network drives.

In addition, you can use the manual install program, manual.bat, to install to local or network drives.

Additional reference words: 3.10 3.50
KBCategory: kbsetup
KBSubcategory: Setins

Win32 Priority Class Mechanism and the START Command

PSS ID Number: Q90910
Authored 26-Oct-1992 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

The Win32 priority class mechanism is exposed through CMD.EXE's START command.

START accepts the following switches:

/LOW	-	Start the command in the idle priority class.
/NORMAL	-	Start the command in the normal priority class (this is the default).
/HIGH	-	Start the command in the high priority class.
/REALTIME	-	Start the command in the real-time priority class.

For a complete list of START switches, type the following command at the Windows NT command prompt:

```
start /?
```

Win32 has also been modified to inherit priority class if the parent's priority class is idle; thus, a command such as

```
start /LOW nmake
```

causes build and all descendants (compiles, links, and so on) to run in the idle priority class. Use this method to do a real background build that will not interfere with anything else on your system.

A command such as

```
start /HIGH nmake
```

runs BUILD.EXE in the high priority class, but all descendants run in the normal priority class.

MORE INFORMATION

Be very careful with START /HIGH and START /REALTIME. If you use either of these switches to start applications that require a lot of cycles, the applications will get all the cycles they ask for, which may cause the

system to appear hung.

Additional reference words: 3.10
KBCategory: kbprg
KBSubcategory: BseProcThrd

Win32 SDK Knowledge Base Available as Help Files (JUN 94)

PSS ID Number: Q106544
Authored 10-Nov-1993 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for the Windows NT version 3.1

The Microsoft Knowledge Base has been compiled into help files for Windows. The Win32 SDK Knowledge Base is compiled into two versions. The only difference is that one of the versions has a full-text search index.

The Win32 SDK Knowledge Base is divided into help files as follows:

ARCHIVE NAME	FILENAME	TOPIC	MODIFY DATE
WIN32KBI.EXE	WIN32KB.HLP WIN32KB.IND	Win32 SDK with a full-text search index.	06-JUN-1994
WIN32KB.EXE	WIN32KB.HLP	Win32 SDK without a full-text search index.	06-JUN-1994

NOTE: The full-text search index makes the resulting file over twice the size of a standalone help file.

Win32 SDK with Full-Text Search

Download WIN32KBI.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for WIN32KBI.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download WIN32KBI.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \SOFTLIB\MSLFILES directory
  Get WIN32KBI.EXE

Win32 SDK without Full-Text Search

Download WIN32KB.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe

GO MSL
Search for WIN32KB.EXE
Display results and download

- Microsoft Download Service (MSDL)
Dial (206) 936-6735 to connect to MSDL
Download WIN32KB.EXE

- Internet (anonymous FTP)
ftp ftp.microsoft.com
Change to the \SOFTLIB\MSLFILES directory
Get WIN32KB.EXE

Additional reference words: DSKBGuide 3.10
KBCategory: kbref kbtlc kbfile
KBSubcategory: GenSDK

Win32 Shell Dynamic Data Exchange (DDE) Interface

PSS ID Number: Q105446
Authored 20-Oct-1993 Last modified 23-Feb-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1, 3.5, and 3.51

SUMMARY

Information on the DDE Interface to Program Manager can be found in the Win32 application programming interface (API) reference under the topic "Shell Dynamic Data Exchange Interface."

The SDK contains a sample program that interfaces with Program Manager. The sample can be found in MSTOOLS\SAMPLES\DDEML\DDEPROG.

MORE INFORMATION

AppProperties cannot be used to get the item icon, description, or working directory, as it can in Windows 3.1. Therefore, GetIcon(), GetDescription(), and GetWorkingDir() do not work in Windows NT. However, AppProperties can still be used to dump out the contents of a group, by specifying the group name in lParam.

Here's how a Win32-based application can get the item icon, the description, and the working directory:

1. Initiate a conversation with the Shell as follows

```
SendMessage( -1, WM_DDE_INITIATE, hWndApp, lParam );
```

   where lParam points to an atom representing:

```
LOWORD | HIWORD
-----------------------------------------------------------

Shell  | AppIcon        : To get an item's icon
Shell  | AppDescription : To get an item's description
Shell  | AppWorkingDir  : To get an item's working directory
```

2. Get the item DDE number.

   The DDE number is stored by Program Manager in the STARUPINFO structure of the application when the application is started. The application can get the startup information with:

```
GetStartupInfo( &StartupInfo );
```

   The field lpReserved in the STARUPINFO structure is in the

following format

```
dde.#, hotkey.##
```

where the DDE number is # and the hot key for the item is ##.

3. Request data as follows

```
SendMessage( hwndProgMan, WM_DDE_REQUEST, hwndApp, lParam );
```

where the lParam HIWORD is the item's DDE number obtained in step 2.

4. The data is returned in lParam of WM_DDE_DATA message. The DDE data value is a string for AppDescription and AppWorkingDir DDE transactions. For AppIcon, the data value has the following structure:

```
typedef struct _PMIconData {
   DWORD dwResSize;
   DWORD dwVer;
   BYTE iResource;  // icon resource
} PMICONDATA, *LPPMICONDATA;
```

To create the icon, the application must call:

```
hIcon = CreateIconFromResource((LPBYTE)&(lpPMIconData->iResource),
            lpPMIconData->dwResSize,
            TRUE,
            lpPMIconData->dwVer
        );
```

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: UsrDde

Win32 Software Development Kit Buglist

PSS ID Number: Q95804
Authored 17-Jan-1994 Last modified 18-May-1995

The information in this article applies to:

- FastTips for the Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5

INSTRUCTIONS

Microsoft FastTips is available 24 hours a day, 7 days a week, from a touch-tone telephone. To order items from this catalog, first select the items you want to receive, noting the five- to six-digit number of the Item ID shown below for each item, and then:

- Dial the toll-free FastTips number (800) 936-4300.
- When prompted, select the Win32 Software Development Kit.
- Press one (1) on your phone keypad to select Express Order Service.
- When prompted, select the delivery method, fax.
- When prompted, enter your three-digit area code and seven-digit fax number on your phone keypad.
- When prompted, enter the number of the Item ID and press #, for up to five items.

When finished, simply hang up. If you have problems receiving a fax, please call (206) 635-3105.

ARTICLE LISTING

End of listing.

Additional reference words: DSKBGuide 3.50
KBCategory: kbref kbtlc
KBSubcategory: BseMisc GdiMisc NtwkMisc UsrMisc

Win32 Subsystem Object Cleanup

PSS ID Number: Q89290
Authored 16-Sep-1992 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

The Win32 subsystem guarantees that all Win32 objects owned by a process will be freed when an application terminates. To accomplish this, the Win32 subsystem keeps track of who owns these objects; it also keeps a reference count. Reference counts are used when the object is owned by more then one process. For example, a memory mapped file can be used to provide interprocess communication, where more than one process would own that object. The subsystem must make sure that the reference count is zero before the object can be freed.

Freeing of Win32 objects can occur at different times. In general, it occurs at process termination, but for some objects, it occurs at thread termination.

NOTE: When running Win32-based applications with Windows 3.1 using the Win32s environment, it is the responsibility of the Win32-based application to ensure that all allocated GDI objects are deleted before the program terminates. This is different from the behavior of the application with Windows NT. With Windows NT, the GDI subsystem cleans up all orhphaned GDI objects. Because there is no GDI subsystem with Windows 3.1, this behavior is not supported.

MORE INFORMATION

At process or thread termination, the Win32 subsystem searches its lists to find objects owned by this process or thread. Those that are owned by the terminating process or thread and whose reference counts will be set to zero when the process or thread is fully terminated will be freed.

The freeing of objects is slightly different for Win32-based applications running under Win32s on Windows 3.1. The 16-bit objects (GDI objects, windows, global memory, etc.) follow the same clean-up rules as Windows-based applications do under Window 3.1. The 32-bit objects, such as memory allocated via VirtualAlloc(), shared memory via mapped file I/O, 32-bit modules, thunks allocated on the fly (for hook procedures, wndprocs etc.) are all handled by Win32s and freed at process termination.

The following is a list of Win32 objects. Note that it may not be complete.

BASE: console, event, file (including file mapping), mutex, semaphore, thread, process, pipe (including named pipes)

GDI: device context (DC), bitmap, pen, brush, font, region, palette

USER: window, cursor, icon, menu, accelerator table, desktop,
DDE communication objects, DDE conversation objects, dialog

Additional reference words: 3.10
KBCategory: kbprg
KBSubcategory: SubSys

Win32s and LAN Manager APIs

PSS ID Number: Q109204
Authored 28-Dec-1993 Last modified 25-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.0, 1.1, 1.15, and 1.2

NOTE: Win32s provides support for NetBIOS and Windows Sockets.

There is a NETAPI32.DLL shipping with Win32s. However, this doesn't have support for Microsoft LAN Manager application programming interfaces (APIs). This is not a bug because LAN Manager APIs are not supported on Win32s. However, if NETAPI32.DLL is not in the system and an application uses LAN Manager APIs, NETAPI32.DLL will not load because the loader cannot resolve the entry-points from the nonexistent dynamic-link library (DLL).

One of the solutions is to use Universal Thunks. Using Universal Thunks, calls can be made to 16-bit DLLs. Therefore, the 16-bit NETAPI.DLL that ships with LAN Manager can be accessed by this mechanism.

Sample programs using Universal Thunks are provided in the Win32 SDK and the Visual C++ SDK. The sample on the Win32 SDK is in the MSTOOLS\WIN32S\UT\SAMPLES directory. On the Visual C++ 32-bit Edition CD-ROM, the sample is located in the MSVC32S\WIN32S\UT\SAMPLES directory.

Additional reference words: 1.00 1.10 1.20
KBCategory: kbprg
KBSubcategory: W32s

Win32s and Windows NT Timer Differences

PSS ID Number: Q105758
Authored 24-Oct-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.0, 1.1, and 1.2

Under Windows NT, timers are system objects; as such, they are not owned by an application. SetTimer() can be called from within one application with a handle to a window that was created by a different application. This application would process the WM_TIMER messages in the window procedure. The timer event will continue to occur even after the application that created the timer has terminated. Note that it is fairly uncommon for a Win32-based application to create a timer for another application, but this method does work.

Because Win32s runs on top of Windows 3.1 and shares many of its characteristics, timers are owned by the application that calls SetTimer(). The timer event terminates when the application that owns the timer terminates.

Additional reference words: 1.00 1.10 1.20
KBCategory: kbprg
KBSubcategory: W32s

Win32s Device-Independent Bitmap Limit

PSS ID Number: Q115084
Authored 18-May-1994 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Win32s, versions 1.1, 1.15, 1.2, and 1.25a

Under versions of Win32s prior to 1.25, a device-independent bitmap (DIB) is limited to a size of 2.3 MB. This size was chosen to accomodate a bitmap of 1024 by 768 pixels at 24 bits per pixel.

In Win32s 1.25a, this limit was increased to 1280 by 1024 pixels at 24 bits per pixel.

This limit can cause a variety of problems to occur, such as painting problems with SetDIBitstoDevice() if a larger bitmap is used.

Additional reference words: 1.10 1.20
KBCategory: kbprg
KBSubcategory: W32s

Win32s Message Queue Checking

PSS ID Number: Q97918
Authored 25-Apr-1993 Last modified 22-Mar-1995

The information in this article applies to:

- Microsoft Win32s versions 1.1 and 1.2

Win32-based applications that are designed to run with Win32s need to check the message queue through a GetMessage() or PeekMessage() call to avoid locking up the system. With Windows NT this is not a problem, because the input model is desynchronized. That is, each thread has its own input event queue rather than having one queue for the entire system. In a synchronous input model, one application can block all of the others by allowing the single system queue to fill up with its messages. With Windows NT, an application that lets its input queue fill up will not affect other applications.

For more information, please see chapter 3 of the "Win32s Programmer's Reference".

Additional reference words: 1.10 1.20
KBCategory: kbprg
KBSubcategory: W32s

Win32s NetBIOS Programming Considerations

PSS ID Number: Q104314
Authored 13-Sep-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s version 1.0, 1.1, and 1.2

This article addresses common questions about NetBIOS programming under Win32s.

Under Windows 3.1, you normally do not issue a RESET command due to the common name table. When running your Win32-based application under Windows version 3.1, you may issue a NetBIOS RESET command. Win32s keeps an internal list of all the names added in the Windows 3.1 NetBIOS name table by Win32 processes. A RESET on Win32s mimics the RESET on Windows NT by clearing the names added by that process from the system-wide name table. As a result, issuing a RESET command as the first NetBIOS command (as required by Windows NT) on Win32s does not clear out all of the names.

The Windows 3.1 NetBIOS VxD expects memory allocated for the NetBIOS Control Block (NCB) and the data buffer (NCB.ncb_buffer) to be allocated with GlobalAlloc(). The VxD will lock the specified memory page. If a Win32-based application running under Win32s passes the Netbios() command virtual memory, Netbios() will return error 0x22, indicating that there are too many commands outstanding. On Win32s, each piece of memory that might pass through Netbios() must be allocated with GlobalAlloc().

The Win32s NetBIOS thunk layer translates the ncb_buffer pointer in the NCB itself. The translation back from a 16-bit to original 32-bit pointer is done for asynchronous commands in the Win32s private post routine. A problem might occur when an application checks the NCB and finds that the netbios() command is completed (ncb_retcode == NRC_GOODRET), but the Win32s post routine was not called yet. The ncb_buffer has not been translated back. The best way to avoid problems is to define and use a post routine. At that point you are sure that the netbios() command is completed and the NCB is correct. If you don't want to use a post routine, you should make sure that the command was completed by checking the the ncb_retcode field and verifying that the ncb_buffer pointer is a 32-bit pointer.

There is also no need to page lock the NetBIOS post routine code under Win32s. NetBIOS post routine on Win32s is not called at interrupt time. The post routine is called in the context of the process, and therefore there is no need to page lock the post routine code.

Additional reference words: 1.00 1.10 1.20
KBCategory: kbprg
KBSubcategory: W32s

Win32s OLE 16/32 Interoperability

PSS ID Number: Q123422
Authored 01-Dec-1994 Last modified 10-Jan-1995

The information in this article applies to:

- Microsoft Win32s version 1.20

The OLE support provided in Win32s version 1.2 provides full 16/32 interoperabililty for local servers (EXE servers). Therefore, you can embed a 32-bit object implemented by a local server in a 16-bit container and vice-versa.

There is no built-in support in Win32s for 16/32 interoperability for in-process servers (DLL servers). However, you can use Universal Thunks to load a 16-bit DLL in the context of a 32-bit process. This allows you to embed a 16-bit object implemented by a DLL server in a 32-bit container. However, it is quite complicated to write this code because:

- Any OLE interface has a hidden "this" pointer which you must handle in your thunking code.
- OLE uses callbacks. If your 32-bit container calls IDataObject:DAdvise, then your 16-bit server may call back into the 32-bit side with the Advise interface. Your thunking code will have to handle this type of conversation.

NOTE: Embedding a 32-bit object implemented by an in-process server in a 16-bit container is supported under Windows NT 3.5. However, this functionality may not work correctly for your in-process server because IDispatch and any custom interfaces do not work.

Additional reference words: 1.20
KBCategory: kbole
KBSubcategory: W32s

Win32s Translated Pointers Guaranteed for 32K

PSS ID Number: Q100833
Authored 29-Jun-1993 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32s versions 1.0, 1.1, and 1.2

SUMMARY

Translated pointers are guaranteed to be valid only for 32K, rather than 64K, which selectors are usually limited to. This limitation is for performance reasons.

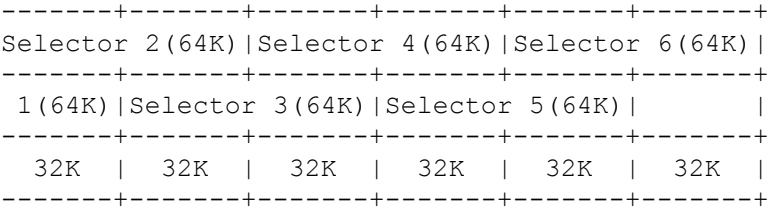
Selectors are tiled every 32K. A 0:32 pointer can be quickly translated into a 16:16 pointer, which will be valid for a minimum of 32K. In other words, the offset portion of the 16:16 pointer is not guaranteed to be 0 (zero) when translated. As a result, even though the translated selectors have a limit of 64K, the offset passed to the 16-bit side may be as large as 32K-1.

Selectors are created on a 32K alignment so that if you pass several pointers to the same range, the Universal Thunk (UT) uses the same selector. Selectors are freed when application terminates.

The alternative is to create a selector for each and every translation, which is very slow.

MORE INFORMATION

For any given address, there are two selectors that point to it, but only one has a limit less than 32K:



Under Win32s, 16-bit and 32-bit applications share the same global data space; therefore, it is possible to share a buffer of up to 64K in size with a far pointer or more than 64K with a huge pointer by doing the following:

1. Do a GlobalAlloc() on the 32-bit side. Be sure to use GMEM_MOVEABLE.
2. Copy the data.
3. Send the handle to the 16-bit side.
4. Get a pointer to the data on the 16-bit side by using GlobalLock().

When you pass a pointer to a block that was allocated via GlobalAlloc() from the 32-bit side, it costs no selectors. The translated pointer is valid until the memory is freed.

Additional reference words: 1.00 1.10 1.20
KBCategory: kbprg
KBSubcategory: W32s

Win32s Version 1.25a Limitations

PSS ID Number: Q131896
Authored 22-Jun-1995 Last modified 26-Jun-1995

The information in this article applies to:

- Microsoft Win32s, version 1.25a

SUMMARY

The following lists the limitations of Win32s, not including the complete list of unsupported Win32 APIs. For information on which Win32 APIs are and are not supported under Win32s, please see the "Win32 Programmer's Reference" or the WIN32API.CSV file.

MORE INFORMATION

- Thread creation is not supported.

- Win32s uses the Windows version 3.1 nonpreemptive scheduling mechanism.

- 32-bit processes should be started from a 16-bit application through WinExec() or Int 21 4B. LoadModule() does not start a 32-bit process.

- Win32-based applications cannot use the EM_SETHANDLE or EM_GETHANDLE messages to access the text in a multiline edit control. These messages allow the sharing of local memory handles between an application and USER.EXE. In Win32s, an application's local heap is 32-bit, so USER.EXE cannot interpret a local handle. To read and write multi-line edit control text, an application should use WM_GETTEXT and WM_SETTEXT.

  Text for multi-line edit controls is stored in the 16-bit local heap in DGROUP, which is allocated by Windows version 3.1 for the WIN32S.EXE stub loaded before each Win32-based application is loaded. This means that the size of edit control text is limited to somewhat less than 64K.

- The CBT hook thunks do not copy back the contents of structures, so any changes are lost.

- Win32-based applications should not call through the return from SetWindowsHook(). SetWindowsHook() returns a pointer to the next hook in the chain. To call this function, the application should call DefHookProc(), and pass a pointer to a variable where this address is stored. Because Win32s simulates the old hook API in terms of the new, SetWindowsHook() actually passes back a hook handle, not a function pointer.

- Resource integer IDs must be 16-bit values.

- PostMessage() and PeekMessage() do not thunk structures. These functions do not allocate space for repacking structures, because there is no way

to know when to free up the space.

- Private application messages should not use the high word of wParam. Win32s uses Windows to deliver window messages. For unknown messages, wParam is truncated to make it fit into 16-bits. Therefore, Win32-based applications should not put 32-bit quantities into the wParam of privately-defined messages, which are unknown to the thunk layer.

- After calling FindText(), an application cannot directly access the FINDREPLACE structure. When an application calls the common dialog function FindText(), it passes a FINDREPLACE structure. The FindText() dialog communicates with the owner window through a registered message, in which the lParam points to the structure. The thunks repack this structure in place, so that unless the application is processing the registered message, it should not access the structure.

- Subclassing a window that owns a FindText() common dialog does not work. The FindText() thunk repacks the FINDREPLACE structure in place. This means that 32-bit window procedures cannot subclass 16-bit owners of FindText() dialogs without likely trashing four bytes beyond the end of the structure, because the 32-bit FINDREPLACE is four bytes bigger than the 16-bit version.

  For 32-bit owners of the dialog box, there are also problems. Whenever the structure might be referenced by 16-bit code, it needs to be in the 16-bit format. However, when the FindText() dialog box terminates, the structure needs to be in 32-bit format so that the application can read the final values. The dialog box indicates to its owner that it is about to be destroyed by sending the commdlg_FindReplace message with the FR_DIALOGTERM bit set in the Flags field of the FINDREPLACE structure. After sending this message, the FindText() dialog box does not reference the structure anymore, so the commdlg_FindReplace thunk leaves it in 32-bit format.

  The problem occurs when the owner has been subclassed. Once the message has moved to the 32-bit side, the structure will not be reconverted to 16-bits. Suppose the dialog box owner was subclassed by a 16-bit window procedure, which was in turn subclassed by a 32-bit window procedure. When a message is sent to the owner, it will first go to 32-bits, then to 16-bits, then back to 32-bits to the original window procedure:

  1. 16->32 message sent to 32-bit subclasser (struct repacked to 32-bits)
  2. 32->16 message sent to 16-bit subclasser (struct not repacked)
  3. 16->32 message sent to 32-bit original WndProc (struct not repacked)

  The 16-bit subclasser cannot interpret the message parameters because they are in 32-bit format. Between steps 2 and 3, the structure is not repacked because it's already in 32-bit format.

  Therefore, when a Win32-based application calls FindText(), allocate the following:

  1. A structure that contains the htask of WIN32S.EXE.

  2. The original 32-bit FINDREPLACE pointer.

3. A count of the number of times the structure has been thunked without returning.

4. A 16-bit format copy of FINDREPLACE.

The structure is repacked into the 16-bit version and this copy is passed to Windows. The count is initially 0. When the commdlg_FindReplace message is sent in either direction (16->32 or 32->16), repack the structure on the stack. If it was originally 32-bit, increment the count. When returning in either direction, unpack the structure and, if originally 32-bit, decrement the count.

If the Flags field of the structure has the FR_DIALOGTERM bit set, the structure is originally 32-bit, and the count goes to 0 on a return, then repack the structure back to the original 32-bit space.

- DDE messages are always posted, not sent, except for two: WM_DDE_INITIATE and WM_DDE_ACK (responding to the former). The sent form of WM_DDE_ACK is different from the posted form. When sent, no thunking of lParam is necessary when going between 16- and 32-bit format. However, when posting, the lParam parameter is repacked as two DWORDS into private memory, allocated by COMBO.DLL (or by USER.DLL in Windows NT). The thunk layer makes this distinction through the InSendMessage() function. Therefore, a Win32-based application should not do anything in the processing of a WM_DDE_INITIATE message that would cause the return code from InSendMessage() to be FALSE.

- The following DDE messages are handled differently by applications under Windows and Win32:

  WM_DDE_ACK
  WM_DDE_ADVISE
  WM_DDE_DATA
  WM_DDE_EXECUTE
  WM_DDE_POKE

  Because of the widening of handles, there is not enough space in the Win32 message parameters for all the Windows data. Win32-based applications are required to call helper functions that package the data into memory (DDEPACK structure) referenced by a special handle. Win32s implements the helper functions and allocates the DDEPACK structure on behalf of 16-bit code when necessary.

  As with other memory handles shared through DDE, there are rules concerning who frees the memory allocated by these helper routines.

```
lParam = PackDDElParam(...)
if (PostMessage(...,lParam))
   receiving window has obligation of freeing memory
   lParam now invalid for this process
else
   FreeDDElParam(...,lParam)
```

A Win32s hook procedure that has called CallNextHookProc() should not

use the lParam handle as the later's return code indicates that the message has been processed. The hooks that could possibly process a DDE message are:

```
WH_DEBUG
WH_HARDWARE
WH_KEYBOARD
WH_MOUSE
WH_MSGFILTER
WH_SYSMSGFILTER
```

In each case, if the hook proc (and therefore CallNextHookEx()) returns a non-zero value, the message was either discarded or processed. Windows-based procedures should not use the lParam of a DDE message after handing it off to any other message API because they cannot know if the message has been processed, so they must assume that it has been.

- WDEB386 does not support FreeSegment() for 32-bit segments. Win32s Kernel Debugger support depends on it.

- The maximum size of shared memory is limited by Windows memory.

```
CreateFileMapping(hFile,
                  lpsa,
                  fdwProtect,
                  dwMaximumSizeHigh,
                  dwMaximumSizeHigh,
                  lpszMapName)

lpsa               - Ignored.
dwMaximumSizeHigh - Must be zero.


MapViewOfFile(hMapObject,
              fdwAccess,
              dwOffsetHigh,
              dwOffsetLow,
              cbMap)

dwOffsetHigh - Must be zero.
```

- SetClipbrdData() must be used only with a global handle. Otherwise, the data can't be accessed by other applications.

- Win32s supports printing exactly as Windows version 3.1 does. Win32s does not add beziers, paths, or transforms to GDI; to allow applications to use these features on PostScript printers, you must call the Escape function with the appropriate escape codes.

  Applications link to Windows version 3.1 printer drivers through LoadLibrary() and GetProcAddress(), which have special support for this purpose. There is no general mechanism allowing a Win32-based application to link to a 16-bit DLL.

- Win32s does not support these escape codes:

```
BANDINFO                ;24
GETSETPAPERBINS         ;29
ENUMPAPERMETRICS        ;34
EXTTEXTOUT              ;512
SETALLJUSTVALUES        ;771
```

- Integer atoms must be in the range 0-0x3FFF. This restriction is necessary for the current implementation of the window properties API thunks: SetProp(), GetProp(), RemoveProp(), EnumProps(), and EnumPropsEx(). Integer atoms above 0x4000 are rejected by the thunk layer.

- Arrays must fit in 64K after converting to 16-bit format. An array passed to a function such as SetCharABCWidths() or Polyline() must fit within 64K after its elements have been converted to their 16-bit form. Its 32-bit form may be bigger than 64K.

- All Windows version 3.1 APIs that return void, return 1 to the Win32-based application.

  You can simulate a boolean return value by validating the API parameters and returning FALSE if one is bad. However, you can't always do complete validation, and if the API is called, you must assume it succeeded unless there is a method to verify whether or not it succeeded.

- Win32 child window IDs must be sign-extended 16-bit values. This excludes the use of 32-bit pointer values as child IDs.

- When calling PeekMessage(), a Win32-based application should not filter any messages for Windows internal window classes (button, edit, scrollbar, and so on). The messages for these controls are mapped to different values in Win32, and checking for the necessity of mapping is time-consuming.

- The dwThreadId parameter in SetWindowsHookEx() is ignored. The dwThreadId is translated to hTask in Windows 3.1. There's a bug in Windows version 3.1 where if hTask!=NULL, the call may fail.

- Floating point (FP) emulation by exception cannot be performed in 16-bit applications. When tasks are switched between applications, the CR0-EM bit state is not preserved in order to support 32-bit application FP emulation by exception without breaking the existing 16-bit applications that use FP instructions. The CR0-EM bit is assumed to be cleared during execution of 16-bit application FP instructions. Upon executing a 16-bit application FP instruction, the bit is cleared and reset when switching back to a 32-bit application. The CR0-EM bit management is done in the Win32s VxD, thus disabling the possibility of getting an int 7 exception just by setting the CR0-EM bit in a 16-bit application.

- EndDialog() nResult parameter is sign-extended. Applications specify the return value for the DialogBox() function by way of the nResult parameter to the EndDialog() API. This parameter is of type int, which is 32-bit in Win32s. However, this value is thunked through to the Windows version 3.1 EndDialog() API, which truncates it to a 16-bit

value. Win32s sign-extends the return code from DialogBox().

- GetClipBox() returns SIMPLEREGION(2) and COMPLEXREGION(3). Because Windows NT is a preemptive multitasking system, GetClipBox() on Windows NT never returns SIMPLEREGION(2). The reason for this is that between the time the API was called and the time the application gets the result, the region may change. Win32s can return both SIMPLEREGION(2) and COMPLEXREGION(3).

- PeekMessage() filtering for posted messages (hWnd==-1) is not supported. The hWnd is replaced with NULL.

- Message queue length is limited to Windows default: 8 or whatever length was set by DefaultQueueSize=n in the WIN.INI file. This limit may be increased in the future to a larger size, but there will always be a limit.

- GetFileTime() and SetFileTime() process only the lpLastWriteTime parameter and return an error if this parameter is not supplied. In the DEBUG version, supplying the other parameters causes a warning message to be displayed.

- The precision of the time of a file is two seconds (MS-DOS limitation).

- CreateProcess has the following limitations:

  - fdwCreate - Only DEBUG_PROCESS and DEBUG_ONLY_THIS_PROCESS are supported.

  - Process priority is always NORMAL.

  - lpsaProcess, lpsaThread - Security information ignored.

- Always use device-independent bitmaps for color bitmaps. Win32s supports the four Win32 device-dependent bitmap APIs. These are device-dependent in the sense that the bitmap bits are supplied without a color table to explain their meaning.

  CreateBitmap
  CreateBitmapIndirect
  GetBitmapBits
  SetBitmapBits

  These are well defined and fully supported for monochrome bitmaps. For color bitmaps, these APIs are not well defined and Win32s relies on the Windows display driver for their support. This means that an application cannot know the format of the bits returned by GetBitmapBits() and should not attempt to directly manipulate them. The values returned by GetDeviceCaps() for PLANES and BITSPIXEL and the values returned by GetObject() for a bitmap do not necessarily indicate the format of the bits returned by GetBitmapBits(). It is possible for the GDI DIB APIs to be unsupported on some displays. However, it is now rare for display drivers to not support DIBs. The one case where you may encounter a lack of DIB support is with printer drivers, which may not support the GetDIBits() API, though most do support the SetDIBits() API.

Win32s does not transform the bits in any way when passing them on to a Windows version 3.1 API. When running an application that creates a device-dependent bitmap via CreateBitmap() or CreateBitmapIndirect(), be aware that the bits it is passing in may not be in the right format for the device. Windows NT takes care of this by treating the bits as a DIB whose format is consistent with the PLANES and BITSPIXEL values; but Win32s simply passes them through.

- GetPrivateProfileString() and GetProfileString() return an error when the lpszSection parameter is NULL. Under Windows NT, they give all the sections in the .INI file.

- String resources are limited to a length of 255, as they were in Windows version 3.1.

- TLS locations are the same in all processes for a specific DLL. This is because Win32s does not support per-instance data for DLLs. The TLS locations are unique per DLL. Each DLL should call TlsAlloc() only once if it is runing on Win32s. The index returned will be valid for all Win32 processes.

- GlobalCompact() is thunked through to Windows version 3.1 GlobalCompact(). This API has no effect on memory allocated through VirtualAlloc(), which does not come from the Windows global heap.

- GetVolumeInformation() does not support the Volume ID.

- GetFileInformationByHandle() create time and access time are always 0 (MS-DOS limitations). The volume id, file index low/high are also 0 (Win32s limitations). This affects the CRT fstat() as well.

- CreatePolyPolygonRgn() requires a closed polygon, as it does under Windows version 3.1. If the polygons are not closed, the Windows NT call closes the polygons for you. In Windows version 3.1 or Win32s, if the polygons are not closed, the call does not create the region correctly, or it returns an error for an invalid parameter.

- Win32s does not support the DIB_PAL_INDICES option for SetDIBits(). It will be supported in a future release. DIB_PAL_PHYSINDICES and DIB_PAL_LOGINDICES are not supported either.

- The WH_FOREGROUNDIDLE hook type is not supported. Windows version 3.1 does not provide the necessary support.

- The brush styles BS_DIBPATTERNPT and BS_PATTERN8X8 are not supported and cause an error to be returned.

- The hFile in DLL and PROCESS DEBUG events is not supported in Win32s because there is no support for duplicating file handles between processes (basically an MS-DOS limitation).

  There are two way to access the image bytes: ReadProcessMemory() or open the file in compatibility mode using the name provided in lplpImageName.

- Under Windows NT, NetBIOS keeps a different name table for each process. On Win32s, there is only one NetBIOS name table for the system. Each name (group or unique) added by a process is kept in a doubly-linked list. Through NCBRESET or by destroying the process, you delete all the names that were added by the process. Win32s does not implement the full NetBIOS 3.0 specification, which has a separate name table per process.

- When calling 32-bit code from 16-bit code with UT (for example, from an interrupt routine), the stack must be at least 10K. The interrupt routine must assure that the stack will be big enough.

- GetThreadContext() and SetThreadContext() can be called only from within an exception handler or an exception debug event. At all other times, these functions return FALSE and the error code is set to ERROR_CAN_NOT_COMPLETE.

- CreateProcess() PROCESS_INFORMATION is not supported for 16-bit applications. A valid structure must be passed to CreateProcess(), but the function fills all the fields with NULLs and zeroes.

- Win32s does not support the Windows NT event mechanism, therefore the ncb_event field in NCB structure is not supported.

- CreateFileMapping() does not support SEC_NOCACHE or SEC_NOCOMMIT. The call fails with ERROR_INVALID_PARAMETER.

- WaitForDebugEvent() does not fully support the dwTimeout parameter. If the parameter is zero, WaitForDebugEvent() behaves the same as under Windows NT. Otherwise, the parameter is treated as if it were INFINITE. However, the function returns if any messages arrive. If a message arrives, the return value is FALSE. The calling process should call SetLastError(0) before calling WaitForDebugEvent() and examine GetLastError() if WaitForDebugEvent() returns FALSE. If the error is zero, it means that a message arrived and the process should process the message. Otherwise, the process should handle the error.

- If a section contains duplicated keys, GetPrivateProfileSection() returns the duplicated keys, but all values are the same as the value of the first key.

  Suppose a section contains these keys:

  ```
  key1=x1
  key2=x2
  key2=x3
  key2=x4
  key3=x5
  key2=x6
  ```

  The values returned are:

  ```
  key1=x1
  key2=x2
  key2=x2
  key2=x2
  ```

```
      key3=x5
      key2=x2
```

- String IDs of resources should not be longer than 255 characters.

- String IDs must be in the English language. The resources themselves can be mulitilingual.

- GetDlgItemInt() only translates up to 16-bit int/unsigned values. This is because it gets its value from Windows, which translates only 16-bit values. As a workaround, call GetDlgItem() and translate the value using atoi() or sscanf().

Additional reference words: limits
KBCategory: kbprg
KBSubcategory: W32s

WinDbg Message "Breakpoint Not Instantiated"

PSS ID Number: Q99953
Authored 10-Jun-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

The WinDbg message "Breakpoint Not Instantiated" indicates that the debugger could not resolve an address. This can happen for several reasons:

- A specified symbol does not exist. In this case, check for misspelling and check the state of the "ignore case" option if the symbol contains mixed case.

  -or-

- The symbol exists, but the EXE or DLL was built with the wrong debugging information, or none at all. Use the -Zi and -Od compiler options and use the -debug:full, -debugtype:cv, and -pdb:none linker options.

  -or-

- The symbol exists, but it is in a module that has not yet been loaded. If the symbol is in a DLL that is dynamically loaded the breakpoint was probably set before the DLL was loaded. The message is harmless, because WinDbg will instantiated the BP when the module is loaded.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbtool
KBSubcategory: TlsWindbg

Window Message Priorities

PSS ID Number: Q96006
Authored 03-Mar-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

Under Windows NT, messages have the same priorities as with Windows 3.1.

With normal use of GetMessage() (passing zeros for all arguments except the LPMSG parameter) or PeekMessage(), any message on the application queue is processed before user input messages. And input messages are processed before WM_TIMER and WM_PAINT "messages."

MORE INFORMATION

For example, PostMessage() puts a message in the application queue. However, when the user moves the mouse or presses a key, these messages are placed on another queue (the system queue in Windows 3.1; a private, per-thread input queue in Win32).

GetMessage() and its siblings do not look at the user input queue until the application queue is empty. Also, the WM_TIMER and WM_PAINT messages are not handled until there are no other messages (for the thread) to process. The WM_TIMER and WM_PAINT messages can be thought of as boolean toggles, because multiple WM_PAINT or WM_TIMER messages waiting in the queue will be combined into one message. This reduces the number of times a window must repaint itself.

Under this scheme, prioritization can be considered tri-level. All posted messages are higher priority than user input messages because they reside in different queues. And all user input messages are higher priority than WM_PAINT and WM_TIMER messages.

The only difference in the Windows NT model from the Windows versions 3.x model is that there is effectively a system queue per thread (for user input messages) rather than one global system queue. The prioritization scheme for messages is identical.

For information concerning SendMessage() from one thread to another, please see the following article(s) in the Microsoft Knowledge Base:

ARTICLE-ID: Q95000
TITLE : SendMessage() in a Multithreaded Environment

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg

KBSubcategory: UsrWndw

Window Owners and Parents

PSS ID Number: Q84190
Authored 04-May-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

In the Windows environment, two relationships that can exist between windows are the owner-owned relationship and the parent-child relationship.

The owner-owned relationship determines which other windows are automatically destroyed when a window is destroyed. When window A is destroyed, Windows automatically destroys all of the windows owned by A.

The parent-child relationship determines where a window can be drawn on the screen. A child window (that is, a window with a parent) is confined to its parent window's client area.

This article discusses these relationships and some Windows functions that provide owner and parent information for a given window.

MORE INFORMATION

A window created with the WS_OVERLAPPED style has no owner and no parent. Alternatively, the desktop window can be considered the owner and parent of a WS_OVERLAPPED-style window. A WS_OVERLAPPED-style window can be drawn anywhere on the screen and Windows will destroy any existing WS_OVERLAPPED-style windows when it shuts down.

A window created with the WS_POPUP style does not have a parent by default; a WS_POPUP-style window can be drawn anywhere on the screen. A WS_POPUP-style window will have a parent only if it is given one explicitly through a call to the SetParent function.

The owner of a WS_POPUP-style window is set according to the hWndParent parameter specified in the call to CreateWindow that created the pop-up window. If hWndParent specifies a nonchild window, the hWndParent window becomes the owner of the new pop-up window. Otherwise, the first nonchild ancestor of hWndParent becomes the owner of the new pop-up window. When the owner window is destroyed, Windows automatically destroys the pop up. Note that modal dialog boxes work slightly differently. If hWndParent is a child window, then the owner window is the first nonchild ancestor that does not have an owner (its top-level ancestor).

A window created with the WS_CHILD style does not have an explicit owner; it is implicitly owned by its parent. A child window's parent is the window specified as the hWndParent parameter in the call to CreateWindow that created the child. A child window can be drawn only within its parent's client area, and is destroyed along with its parent.

An application can change a window's parent by calling the SetParent function after the window is created. Windows does not provide a method to change a window's owner.

Windows provides three functions that can be used to find a window's owner or parent:

- GetWindow(hWnd, GW_OWNER)
- GetParent(hWnd)
- GetWindowWord(hWnd, GWW_HWNDPARENT)

GetWindow(hWnd, GW_OWNER) always returns a window's owner. For child windows, this function call returns NULL. Because the parent of the child window behaves similar to its owner, Windows does not maintain owner information for child windows.

The return value from the GetParent function is more confusing. GetParent returns zero for overlapped windows (windows with neither the WS_CHILD nor the WS_POPUP style). For windows with the WS_CHILD style, GetParent returns the parent window. For windows with the WS_POPUP style, GetParent returns the owner window.

GetWindowWord(hWnd, GWW_HWNDPARENT) returns the window's parent, if it has one; otherwise, it returns the window's owner.

Two examples of how Windows uses different windows for the parent and the owner to good effect are the list boxes in drop-down combo boxes and the title windows for iconic MDI (multiple document interface) child windows.

Due to its size, the list box component of a drop-down combo box may need to extend beyond the client area of the combo box's parent window. Windows creates the list box as a child of the desktop window (hWndParent is NULL); therefore, the list box will be clipped only by the size of the screen. The list box is owned by the first nonchild ancestor of the combo box. When that ancestor is destroyed, the list box is automatically destroyed as well.

When an MDI child window is minimized, Windows creates two windows: an icon and the icon title. The parent of the icon title window is set to the MDI client window, which confines the icon title to the MDI client area. The owner of the icon title is set to the MDI child window. Therefore, the icon title is automatically destroyed with the MDI child window.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrWndw

WindowFromPoint() Caveats

PSS ID Number: Q65882
Authored 26-Sep-1990 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

When the coordinates passed to the WindowFromPoint() function correspond to a disabled, hidden, or transparent child window, the handle of that window's parent is returned.

To retrieve the handle of a disabled, hidden, or transparent child window, given a point, the ChildWindowFromPoint() function must be used.

MORE INFORMATION

The following code fragment demonstrates the use of the ChildWindowFromPoint() function during the processing of a WM_MOUSEMOVE message. This code finds the topmost child window at a given point, regardless of the current state of the window.

In this fragment, hWnd is the window receiving this message and is assumed to have captured the mouse via the SetCapture() function.

```
HWND  hWndChild, hWndPoint;
POINT pt;
    .
    .
    .
    case WM_MOUSEMOVE:
        pt.x = LOWORD(lParam);
        pt.y = HIWORD(lParam);

        /*
         * Convert point to screen coordinates. When the mouse is
         * captured, mouse coordinates are given in the client
         * coordinates of the window with the capture.
         */
        ClientToScreen(hWnd, &pt);

        /*
         * Get the handle of the window at this point. If the window
         * is a control that is disabled, hidden, or transparent, then
         * the parent's handle is returned.
         */
        hWndPoint = WindowFromPoint(pt);
```

```
        if (hWndPoint == NULL)
            break;

        /*
         * To look at the child windows of hWnd, screen coordinates
         * need to be converted to client coordinates.
         */
        ScreenToClient (hWndPoint, &pt);

        /*
         * Search through all child windows at this point. This
         * will continue until no child windows remain.
         */
        while (TRUE)
            {
            hWndChild = ChildWindowFromPoint(hWndPoint, pt);

            if (hWndChild && hWndChild != hWndPoint)
                hWndPoint = hWndChild;
            else
                break;
            }

        // Do whatever processing is desired on hWndPoint

        break;
```

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrWndw

Windows 95 Network Programming Support

PSS ID Number: Q125702
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

Windows 95 supports the network programming interfaces listed in the table below:

Interface	Comments
Microsoft RPC	Supports a subset of the protocol sequences specified in Microsoft RPC 1.0. For additional information, please see the following article(s) in the Microsoft Knowledge Base: ARTICLE-ID:Q125701 TITLE :Windows 95 RPC: Supported Protocol Sequences
Windows Sockets (WinSock)	Supports the TCP/IP and IPX/SPX network transports. For additional information, please see the following article(s) in the Microsoft Knowledge Base: ARTICLE-ID:Q125704 TITLE :Multiprotocol Support for Windows Sockets
NetBIOS	Supports NetBIOS v3.0, including the Microsoft extension function NCBENUM.
Network DDE (NetDDE)	Includes the NetDDE agent and a 16 bit NetDDE API DLL. 32-bit applications must thunk to NDDEAPI.DLL. For additional information, please see the following article(s) in the Microsoft Knowledge Base: ARTICLE-ID:Q125703 TITLE :Windows 95 Support for Network DDE

Additional reference words: 4.00
KBCategory: kbprg kbnetwork
KBSubcategory: NtwkMisc

Windows 95 RPC: Supported Protocol Sequences

PSS ID Number: Q125701
Authored 01-Feb-1995 Last modified 06-Apr-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

The Windows 95 RPC runtime libraries support a subset of the protocol sequences introduced in Microsoft RPC 1.0. Here is the list of supported protocol sequences:

Protocol Sequence	Comments
ncacn_ip_tcp	Requires TCP/IP installation
ncacn_nb_nb	NetBEUI is the only supported NetBIOS transport
ncacn_np	Support for client applications only
ncacn_spx	Requires IPX/SPX installation
ncalrpc	Local communication only

Regarding ncacn_np: Windows 95 does not support server-side named pipes, so ncacn_np is not a valid protocol sequence for RPC servers running on Windows 95. However, ncacn_np is valid for RPC client applications.

Additional reference words: 4.00 NETWORKING
KBCategory: kbnetwork
KBSubcategory: NtwkRpc

Windows 95 Support for LAN Manager APIs

PSS ID Number: Q125700
Authored 01-Feb-1995 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

Windows 95 supports a subset of the Microsoft LAN Manager API. This is a list of the supported functions:

```
NetAccessAdd
NetAccessCheck
NetAccessDel
NetAccessEnum
NetAccessGetInfo
NetAccessGetUserPerms
NetAccessSetInfo
NetConnectionEnum
NetFileClose2
NetFileEnum
NetSecurityGetInfo
NetServerGetInfo
NetServerSetInfo
NetSessionDel
NetSessionEnum
NetSessionGetInfo
NetShareAdd
NetShareDel
NetShareEnum
NetShareGetInfo
NetShareSetInfo
```

Windows 95 support for these functions differs from Windows NT in two ways. First, since Windows 95 doesn't support Unicode, these functions require ANSI strings. Second, Windows 95 exports the Lan Manager functions from SVRAPI.DLL instead of NETAPI32.DLL. If an attempt is made to run a native Windows NT application on Windows 95, the following error will result:

> "The <application> file is linked to missing export NETAPI32.DLL
> <Net...API>"

To handle these differences, applications targeted to both Windows NT and Windows 95 should do the following:

1. Avoid importing Lan Manager functions from NETAPI32.DLL at link time. Instead, applications should do a run time version check and dynamically link to NETAPI32.DLL for Windows NT or SVRAPI.DLL for Windows 95.

   For additional information on version checking, please see the following article(s) in the Microsoft Knowledge Base:

ARTICLE-ID:Q92395
TITLE :Determining System Version from a Win32-based Application

2. Make sure the application doesn't depend on the presence of unsupported API's.

3. When calling Lan Manager API's, pass strings using a character set appropriate for the host operating system. Use Unicode strings for Windows NT and ANSI strings for Windows 95.

If you are only targetting Windows 95 and wish to use SVRAPI.DLL, SVRAPI.H and SVRAPI.LIB are included in the Windows 95 DDK. NOTE: The formal parameter lists for the LAN Manager APIs may be slightly different between the header files for Windows NT and Windows 95.

Additional reference words: 4.00 95
KBCategory: kbnetwork
KBSubcategory: NtwkLmapi

Windows 95 Support for Network DDE

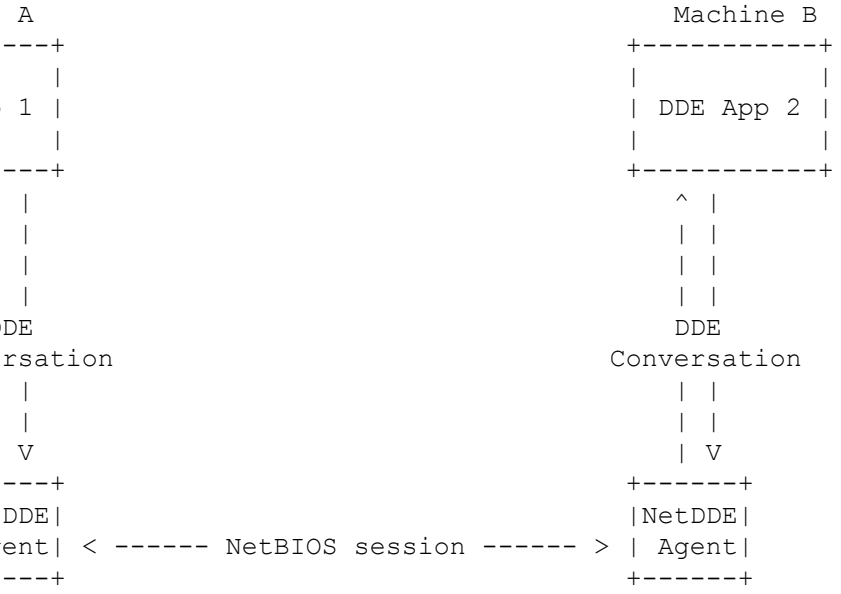
PSS ID Number: Q125703
Authored 01-Feb-1995 Last modified 26-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)

Network DDE is a technology that allows applications that use the DDE transport to transparently exchange data over a network. It consists of two major components:

1. The NetDDE agent. This is a service that acts as a proxy for the remote DDE application. It communicates with all local DDE applications, and with remote NetDDE agents using NetBIOS as shown in this chart:



2. A DLL that implements NetDDE API functions such as NDdeShareAdd, NDdeShareDel, and so on. This DLL is usually named NDDEAPI.DLL.

In the interests of backwards compatibility, Windows 95 includes a NetDDE agent and a 16-bit NetDDE API DLL. However, Windows 95 does not include a 32-bit NetDDE API DLL. Consequently, 32-bit applications that use NetDDE API functions will need to thunk to the 16-bit NETAPI.DLL.

Additional reference words: 4.00
KBCategory: kbnetwork kbprg
KBSubcategory: UsrNetDde

Windows Dialog-Box Style DS_ABSALIGN

PSS ID Number: Q11590
Authored 23-Oct-1987 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

The Windows dialog-box style DS_ABSALIGN (used in WINDOWS.H) means "Dialog Style ABSolute ALIGN." Specifying this style in the dialog template tells Windows that the dtX and dtY values of the DLGTEMPLATE struct are relative to the screen origin, not the owner of the dialog box. When this style bit is not set, the dtX and dtY fields are relative to the origin of the parent window's client area.

Use this term if the dialog box must always start in a specific part of the display, no matter where the parent window is on the screen.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDlgs

Windows Does Not Support Nested MDI Client Windows

PSS ID Number: Q74041
Authored 11-Jul-1991 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

The Microsoft Windows implementation of the multiple document interface (MDI) does not support nested MDI client windows. In other words, neither an MDI client window nor an MDI child window can have an MDI client window as a child.

MORE INFORMATION

A Windows MDI client window is a member of the MDIClient window class, and the Windows MDI model assumes that the parent of an MDIClient window is a top-level frame window with a valid menu bar. This assumption is necessary to implement the basic functionality defined by the MDI interface, and it precludes the possibility of using nested MDIClient windows. However, an application can have multiple top-level windows, and each top-level window can have a separate MDIClient window as a child.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMdi

Windows Help PositionWindow Macro Documented Incorrectly

PSS ID Number: Q83911
Authored 23-Apr-1992 Last modified 24-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK, versions 3.5 and 3.51

SUMMARY

In source files for Windows Help, the help author can specify the size, position, and state of a window with the PositionWindow macro.

MORE INFORMATION

According to page 326 of the "Microsoft Windows Software Development Kit: Programmer's Reference, Volume 4: Resources," the state parameter to the PositionWindow macro is either 0 to specify a normal sized window or 1 to specify a maximized window. This information is incorrect. The state parameter can assume any of ten different values, as follows:

Value	Corresponding Windows Constant	Action
0	SW_HIDE	Hides the window and passes activation to another window.
1	SW_SHOWNORMAL	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position.
2	SW_SHOWMINIMIZED	Activates the window and displays it as an icon.
3	SW_SHOWMAXIMIZED	Activates the window and displays it as a maximized window.
4	SW_SHOWNOACTIVATE	Displays a window in its most recent size and position. The window that is currently active remains active.
5	SW_SHOW	Activates a window and displays it in its current size and position.
6	SW_MINIMIZE	Minimizes the specified window and activates the top-level window in the window-manager's list.

7	SW_SHOWMINNOACTIVE	Displays the window as iconic. The window that is currently active remains active.
8	SW_SHOW	Displays the window in its current state. The window that is currently active remains active.
9	SW_RESTORE	Same as SW_SHOWNORMAL.

Note that these constants are valid only for Windows and may change if Help is ported to another platform.

Additional reference words: 3.10 3.50
KBCategory: kbtool
KBSubcategory: TlsHlp

Windows NT 3.5 Hives Not Compatible with Windows NT 3.1

PSS ID Number: Q117872
Authored 11-Jul-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT version 3.5

Hives (registry data) saved from a Windows NT 3.5 machine with the RegSaveKey() API (application programming interface) cannot be restored on a Windows NT 3.1 machine with RegLoadKey(), because the hive format for Windows NT 3.5 is different than the Windows NT 3.1 format (it has been made more efficient). However, Windows NT 3.5 supports both the new and old (Windows NT 3.1) hive formats, so hives created on a Windows NT 3.1 machine can be used on a Windows NT 3.5 machine.

To create a hive on Windows NT 3.5 that can be used on Windows NT 3.1, use the Win32 registry APIs to read the registry key by key, and save the values in your own text format. Use the APIs to restore the registry information on either version of Windows NT.

Additional reference words: 3.50
KBCategory: kbprg
KBSubcategory: BseMisc

Windows NT Servers in Locked Closets

PSS ID Number: Q90083
Authored 08-Oct-1992 Last modified 07-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

Some installations are required to restrict access to a server such that access to the server's keyboard/mouse is unavailable to most personnel. Such a server is referred to as a server in a locked closet.

The server administrators may provide an emergency reset button to end users (for example, factory floor workers) in case the system locks up and no administrators are present. In the case where an emergency reset button cannot be provided, an administrator must come and physically unlock the closet in order to get the system up again.

Windows NT requires that the user press CTRL+ALT+DEL to log on. This requirement implies that Windows NT doesn't lend itself well to the server in a locked closet situation. Indeed, someone must press CTRL+ALT+DEL and enter a user ID and a password in order to log on and use the keyboard or mouse to interact locally with a Windows NT machine. However, it is possible to configure the machine as a server in a locked closet such that an administrator is not required to unlock the door to reset the system. The administrator can configure the system so that services are started automatically during boot. Once all the services are started, then the system is fully functional and the administrator need not intervene. If certain services fail to come up, but network service does come up, then the system can be remotely administered.

MORE INFORMATION

Remote administration is possible, assuming that the required basic system services are running. The machine must be on the network. The process requires only the Windows NT product. In other words, Windows NT Advanced Server is not an additional requirement.

Make sure that the following steps have been taken to start system services automatically at system boot and to enable remote administration in case of failure:

1. Use the Service Control Manager to install any application code that must be started as soon as the Windows NT machine reboots.

   Write an application that installs the services and specifies that they should be started automatically. To find more information on the Win32 APIs that support Services, search on "Services Overview" in the Win32

API help text.

Once this is done, the necessary application code can be made to start automatically upon system reboot, without anyone needing to press CTRL+ALT+DEL to log on or to take any other action using the server's local mouse/keyboard.

2. Make sure that the Workstation and Server services start automatically upon reboot.

   Use the Services application in Control Panel to ensure that both the Workstation and Server services start automatically upon reboot. Choose the Help button for instructions on how to install a service and how to configure it to start automatically.

   This will permit an authorized person to remotely administer the system from another machine on the network. Thus, if something from step 1 goes wrong, the administrator still does not need to physically unlock the closet and log on. The administrator can log on to any machine on the network and use the tools on that machine to interact with the server.

Remote administration via dial-up telephone lines is available, but requires RAS (Microsoft Remote Access Service). RAS permits a machine to dial over telephone lines into a network, and to become a full participant on the network. In this way, a system dialing in over RAS can be used to remotely administer the system in the locked closet.

Note that while these steps allow servers locked in closets to be restored without an administrator, it is still preferable to install a UPS (uninterruptable power supply). Servers in locked closets usually need to provide uninterrupted service to their clients, so a UPS is a better solution. The capability to do remote administration serves as a backup in case of failure.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseMisc

Windows NT Support for the MS-DOS LAN Manager APIs

PSS ID Number: Q110776
Authored 28-Jan-1994 Last modified 05-Jan-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

There is no list that shows specifically which MS-DOS LAN Manager APIs (application programming interfaces) are supported in a VDM (virtual DOS machine) on Windows NT. This article discusses what influenced whether or not an API would be implemented.

The set of APIs that is supported is relatively small. It was necessary to implement named pipes, mailslots, NetServerEnum(), and the NetUseXXX APIs. The APIs that are commonly used in shipping applications were implemented, if possible. There were certain APIs that were impossible to implement from the VDM. The remaining APIs were not added either because Microsoft did not feel that they were used in applications or because they did not make sense in this context, such as the NetServiceXXX APIs.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: NtwkLmapi

Windows NT Virtual Memory Manager Uses FIFO

PSS ID Number: Q98216
Authored 29-Apr-1993 Last modified 02-Mar-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

On page 193 of "Inside NT," Helen Custer states that the Windows NT virtual memory manager uses a FIFO (first in, first out) algorithm, as opposed to a LRU (least recently used) algorithm, which the Windows virtual memory manager uses. While it is true that FIFO can result in a commonly used page being discarded or paged to the pagefile, there are reasons why this algorithm is preferable.

Here are some of the advantages:

- FIFO is done on a per-process basis; so at worst, a process that causes a lot of page faults will slow only itself down, not the entire system.

- LRU creates significant overhead--the system must update its page database every single time a page is touched. However, the database may not be properly updated in certain circumstances. For example, suppose that a program has good locality of reference and uses a page constantly so that it is always in memory. The operating system will not keep updating the timestamp in the page database, because the process is not hitting the page table. Therefore this page may age even though it is in nearly constant use.

- Pages that are "discarded" are actually kept in memory for a while, so if a page is really used frequently, it will be brought back into memory before it is written to disk.

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseMm

Windows Regions Do Not Scale

PSS ID Number: Q71229
Authored 10-Apr-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows, versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.5, 3.51, and 4.0

SUMMARY

The coordinates of Windows regions are in device units, not logical units. Therefore, if the mapping mode is not MM_TEXT, region coordinates do not scale as would be expected. This causes particular problems in metafiles because metafiles often use MM_ISOTROPIC or MM_ANISOTROPIC modes to make pictures appear the same on devices with different resolutions.

To work around this problem, applications should avoid using regions if the mapping mode is changed from MM_TEXT. Regions should also be avoided in metafiles, unless the application knows the scaling factor and can adjust region coordinates itself.

MORE INFORMATION

If the applications that will read and write the metafile are developed together, the writing application can include an MFCOMMENT escape in the metafile to store the region components. This information can be used by the reading application to scale the metafile.

However, this comment is not standard across all applications. This method would not be expected to work if the metafile is imported into a commercial application.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiDc

Windows Socket API Specification Version

PSS ID Number: Q85965
Authored 23-Jun-1992 Last modified 20-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

The Windows Sockets API Specification version 1.1 is now available in two formats. WINSOCK.DOC is a Word for Windows, version 2.0, document and WINSOCK.TXT is an ASCII-text-format document.

These files contain the Windows Sockets API Specification version 1.1, which defines a standard binary interface for tcp/ip transports based on the Berkeley Sockets interface originally in Berkeley UNIX with Windows-specific extensions. This specification has been endorsed by 20 leading companies, and defines the sockets interface in Windows NT. The specification will be supported by various vendors in their upcoming tcp/ip product releases for Windows for MS-DOS.

The Windows Sockets API Specification is contained in a file called WINSOCK (contains ASCII text version) and a file called WINSOCKW (contains a Word for Windows version) which is located in the Microsoft Software Library.

Download WINSOCK.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for WINSOCK.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download WINSOCK.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \softlib\mslfiles directory
  Get WINSOCK.EXE

Download WINSOCKW.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for WINSOCKW.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download WINSOCKW.EXE

- Internet (anonymous FTP)
    ftp ftp.microsoft.com
    Change to the \softlib\mslfiles directory
    Get WINSOCKW.EXE

Additional reference words: 1.00 1.10 3.10 3.50 4.00 95
KBCategory: kbref
KBSubcategory: NtwkWinsock

Windows WM_SYSTIMER Message Is an Undocumented Message

PSS ID Number: Q108938
Authored 20-Dec-1993 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

The WM_SYSTIMER message in Windows is an undocumented system message; it should not be trapped or relied on by an application. This message can occasionally be viewed in Spy or in CodeView while debugging.

Windows uses the WM_SYSTIMER message internally to control the scroll rate of highlighted text (text selected by the user) in edit controls, or highlighted items in list boxes.

NOTE: The WM_SYSTIMER message is for Windows's internal use only and can be changed without prior notice.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrMsg

WM_CHARTOITEM Messages Not Received by Parent of List Box

PSS ID Number: Q72552
Authored 30-May-1991 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

When keyboard input is sent to a list box that has the LBS_WANTKEYBOARDINPUT style bit set, its parent does not receive WM_CHARTOITEM messages; however, WM_VKEYTOITEM messages are received. This is because the list box has the LBS_HASSTRINGS style bit set.

This behavior is by design. Windows sets the LBS_HASSTRINGS style bit for all list boxes except owner-draw list boxes. An owner-draw list box can be created with this style bit turned on or off. For owner-draw list boxes, the state of the LBS_HASSTRINGS style bit determines which messages are sent. WM_CHARTOITEM messages and WM_VKEYTOITEM messages are mutually exclusive.

The documentation for WM_CHARTOITEM states:

> Only owner-draw list boxes that do not have the LBS_HASSTRINGS style can receive this message.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95 listbox
KBCategory: kbprg
KBSubcategory: UsrInp

WM_COMMNOTIFY is Obsolete for Win32-Based Applications

PSS ID Number: Q94561
Authored 10-Jan-1993 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK), versions 3.1, 3.5, 3.51, and 4.0

SUMMARY

Under Windows version 3.1, the WM_COMMNOTIFY message is posted by a communication device driver whenever a COM port event occurs. The message indicates the status of a window's input or output queue.

This message is not supported for Win32-based applications. However, WOW supports the EnableCommNotification() API for 16-bit Windows-based applications running on Windows NT.

MORE INFORMATION

To duplicate the Windows 3.1 functionality for a Win32-based application, refer to the TTY sample, included with the SDK. The TTY sample is a common code base sample, which uses EnableCommNotification() under Windows 3.1 to tell COMM.DRV to post messages to the TTY window.

In Win32, this behavior is simulated with a secondary thread which uses WaitCommEvent() to block on the port and PostMessage() to indicate when the desired event has occurred.

TTY.C defines WM_COMMNOTIFY if WIN32 is defined. Using this method, WM_COMMNOTIFY notifications are simulated but use the same message definition as Windows 3.1.

The TTY sample is located on the Win32 SDK CD in \MSTOOLS\SAMPLES\COMM.

Additional reference words: 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: BseCommapi

WM_CTLCOLORxxx Message Changes for Windows 95

PSS ID Number: Q130952
Authored 31-May-1995 Last modified 06-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) version 4.0

SUMMARY

In previous versions of Windows, an application could change the background, text, and/or text background colors of controls by performing certain actions in response to WM_CTLCOLORxxx messages.

However, in Windows 95, the messages sent by different types of controls are somewhat different.

MORE INFORMATION

The following list outlines the changes in WM_CTLCOLORxxx messages sent by standard controls in Windows 95:

WM_CTLCOLORBTN

Sent By: command buttons (regular and default)

Changes made during this message have no effect on command buttons. Command buttons always use system colors for drawing themselves.

WM_CTLCOLORSTATIC

Sent By: Any control that displays text which would be displayed using the default dialog/window background color. This includes check boxes, radio buttons, group boxes, static text, read-only or disabled edit controls, and disabled combo boxes (all styles).

The changes affect the text drawn in the control. Changes do not affect the checkmarks on the buttons or the outline of the group box.

WM_CTLCOLOREDIT

Sent By: Enabled, non-read-only edit controls and enabled combo boxes (all styles)

The changes affect the background, text, and text background of these controls. For combo boxes, the changes made in this message affect only the "edit" portion of the control. The list portion is affected by the WM_CTLCOLORLISTBOX message.

In previous versions of Windows, radio buttons, check boxes and group boxes would send WM_CTLCOLORBTN messages and paint themselves accordingly. In Windows 95, these controls send WM_CTLCOLORSTATIC messages instead.

These changes were implemented to make changing the appearance of controls more logical (text on the dialog background is now classified as "static").

Additional reference words: 4.00
KBCategory: kbprg
KBSubcategory: UsrCtl

WM_DDE_EXECUTE Message Must Be Posted to a Window

PSS ID Number: Q77842
Authored 28-Oct-1991 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK)versions 3.0 and 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

Chapter 15 of the "Microsoft Windows Software Development Kit Reference, Volume 2" documents the dynamic data exchange (DDE) protocol. The following statement is found on page 15-2:

> An application calls the SendMessage() function to issue the WM_DDE_INITIATE message or a WM_DDE_ACK message sent in response to WM_DDE_INITIATE. All other messages are sent using the PostMessage() function.

In the book "Windows 3: A Developer's Guide" by Jeffrey M. Richter (M & T Computer Books), the sample setup program uses the SendMessage() function to send itself a WM_DDE_EXECUTE message that violates the DDE protocol and may not work in future versions of Windows.

In Richter's sample, no real DDE conversation exists. The correct method to achieve the desired result is to use the SendMessage() function to send a user-defined message to the window procedure. When this message is processed, proceed accordingly.

For more information on user-defined messages, see chapter 6 of the "Microsoft Windows Software Development Kit Reference, Volume 1" for the Windows SDK version 3.0 and chapter 2 of the "Programmer's Reference, Volume 3: Messages, Structures, and Macros" from the Windows SDK version 3.1.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDde

WM_SIZECLIPBOARD Must Be Sent by Clipboard Viewer App

PSS ID Number: Q74274
Authored 15-Jul-1991 Last modified 12-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) versions 3.1 and 3.0
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

The WM_SIZECLIPBOARD message is not generated by the Windows graphical environment; the clipboard viewer generates this message.

An application that can display the clipboard contents, such as the CLIPBRD.EXE application in Windows version 3.0, is a clipboard viewer. When the size of the clipboard viewer's client area changes and the clipboard contains a data handle for the CF_OWNERDISPLAY format, the clipboard viewer must send the WM_SIZECLIPBOARD message to the current clipboard owner. The GetClipboardOwner function returns the window handle of the current clipboard owner. This window handle is the handle in the last call to OpenClipboard.

When sending the WM_SIZECLIPBOARD message, the clipboard viewer must specify two parameters. The wParam parameter identifies the clipboard viewer's window handle. The low-order word of the lParam parameter contains the handle to a block of global memory that holds a RECT data structure. The RECT structure defines the area in the clipboard viewer that the clipboard owner should paint.

Additional reference words: 3.00 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrClp

WM_SYSCOLORCHANGE Must Be Sent to Windows 95 Common Controls

PSS ID Number: Q129595
Authored 30-Apr-1995 Last modified 01-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK)
- Microsoft Win32s version 1.3

SUMMARY

You must ensure that applications that use the new common controls introduced in Windows 95 and Windows NT version 3.51 forward the WM_SYSCOLORCHANGE message to the controls.

MORE INFORMATION

Windows 95 makes it very easy for the user to change the colors of common user interface objects, therefore it is critical that applications not rely on particular colors being constant. When the user changes the color settings, Windows 95 will send a WM_SYSCOLORCHANGE message to all top level windows. Because this message is sent only to top level windows, the common controls will not be notified of the color change unless the application forwards the WM_SYSCOLORCHANGE message to the control.

An example of why this is important is the toolbar control. If the color settings are such that the "3D Objects" color is set to light gray, the toolbar will create its buttons to light gray. However if the WM_SYSCOLORCHANGE message is not forwarded to the toolbar and the 3D Object color is changed to blue, the toolbar buttons will remain light gray while all the other buttons in the system change to blue.

Additional reference words: 1.30 4.00 grey
KBCategory: kbprg
KBSubcategory: UsrCtl

Working Set Size, Nonpaged Pool, and VirtualLock()

PSS ID Number: Q108449
Authored 12-Dec-1993 Last modified 03-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT versions 3.1 and 3.5

SUMMARY

This article discusses memory management issues such as working set size, nonpaged pool, and locking pages into physical memory via VirtualLock().

MORE INFORMATION

Working Set

A process's working set is the set of pages that it has currently in memory. The values for maximum working set and minimum working set are hard-coded in Windows NT and are thus impossible to change. There are three values hard-coded for the maximum working set. The value used for maximum working set depends on whether your machine is considered to be a small, medium, or large machine:

Small machine--less than 16 megabytes (MB)
Medium machine--between 16 MB and 20 MB
Large machine--greater than 20 MB

The system tries to keep about 4 MB free to reduce the paging that occurs when loading a new process, allocating memory, and so forth. Any free RAM beyond what the system requires is available for the system to use to increase your working set size if your process is doing a lot of paging.

Process pages that are paged out of your process space are moved into the "standby list," where they remain until sufficient free RAM is available, or until system memory is low and they need to be reused. If these pages are accessed by your process while they are still on the standby list and more RAM has become available, they will be "soft-faulted" back into the working set. This does not require any disk access, so it is very quick. Therefore, even though you have an upper limit to the size of your working set, you can still have quite a few process pages in memory that can be pulled back into your working set very quickly.

To minimize working set requirements and increase performance, use the Working Set Tuner, WST.EXE, to order the functions within your code. One way to help an application receive a larger working set is to use the Network application in the Control Panel to set the server configuration to "Maximize Throughput for Network Applications."

Nonpaged Pool

System memory is divided into paged pool and nonpaged pool. Paged pool can be paged to disk, whereas nonpaged pool is never paged to disk. In Windows NT 3.1, the default amount of nonpaged pool also depends on whether your machine is considered small, medium, or large. In other words, you will have X amount of nonpaged pool on a 16 MB machine, Y amount of nonpaged pool on a 20 MB machine, and Z amount of nonpaged pool on a machine with more than 20 MB (the exact values for X, Y, and Z were not made public).

Important system data is stored in nonpaged pool. For example, each Windows NT object created requires a block of nonpaged pool. In fact, it is the availability of nonpaged pool that determines how many processes, threads, and other such objects can be created. The error that you will receive if you have too many object handles open is:

```
1816 (ERROR_NOT_ENOUGH_QUOTA)
```

Many 3.1 applications ran into this error because of the limited amount of nonpaged pool. This limit were addressed in Windows NT 3.5. We found that

- Some objects were too large
- Sharing an object caused excessive quota charges
- The quota limits were artificial and fixed

The resources used by each object were evaluated and many were drastically reduced in Windows NT 3.5.

In Windows NT 3.1, every time an object was shared, quota was charged for each shared instance. For example, if you opened a file inheritable and then spawn a process and have it inherit your handle table, the quota charged for the file object was double. Each handle pointing to an object cost quota. Most applications experienced this problem. Under Windows NT 3.5, quota is only charged once per object rather than once per handle.

Windows NT 3.1 had a fixed quota for paged and nonpaged pool. This was determined by the system's memory size, or could be controlled by the registry. The limits were artificial. This was due to the poor design of quotas with respect to sharing. It was also affected by some objects lying about their actual resource usage. In any case, Windows NT 3.5 has revised this scheme.

The Windows NT 3.1 "Resource Kit, Volume I" documents that it is possible to change the amount of nonpaged pool by modifying the following registry entry:

```
HKEY_LOCAL_MACHINE\SYSTEM\
   CurrentControlSet\
   Control\
   Session Manager\
   Memory Management\
   NonPagedPoolSize
```

WARNING: This modification can cause the system to crash, and therefore

Microsoft does not recommend that this registry entry be changed.

Quotas can still be controlled in Windows NT 3.5 using these Windows NT 3.1 registry values. However, this technique is now almost never needed. The new quota mechanism dynamically raises and lowers your quota limits as you bump into the limits. Before raising a limit, it coordinates this with the memory manager to make sure you can safely have your limit raised without using up all of the systems resources.

VirtualLock()

To lock a particular page into memory so that it cannot be swapped out to disk, use VirtualLock(). The documentation for VirtualLock() states the following:

> Locking pages into memory may degrade the performance of the system by reducing the available RAM and forcing the system to swap out other critical pages to the paging file. There is a limit on the number of pages that can be locked: 30 pages. The limit is intentionally small to avoid severe performance degradation.

There is no way to raise this limit in Windows NT 3.1--it is fixed at 30 pages (the size of your working set). The reason that you see a severe performance degradation when an application locks these pages is that Windows NT must reload all locked pages whenever there is a context switch to this application. Windows NT was designed to minimize page swapping, so it is often best to let the system handle swapping itself, unless you are writing a device driver that needs immediate access to memory.

Windows NT 3.5 allows processes to increase their working set size by using SetProcessWorkingSetSize(). This API is also useful to trim your minimum working set size if you want to run many processes at once, because each process has the default minimum working set size reserved, no matter how small the process actually is. The limit of 30 pages does not apply to VirtualLock() when using SetProcessWorkingSetSize().

Additional reference words: 3.10 3.50
KBCategory: kbprg
KBSubcategory: BseMm

Writing Code that Works with Different International Formats

PSS ID Number: Q130056
Authored 10-May-1995 Last modified 16-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows version 3.1
- Microsoft Win32 Software Development Kit (SDK) version 3.5
- Microsoft Win32s version 1.2

SUMMARY

Some European coutries like Germany use a different floating point format that doesn't include the decimal point (.). This has caused problems whtn the actual numbers use the decimal point format (xxx.xx) and the German version of Microsoft Excel is expecting a comma (xxx,xx).

Some developers have made the mistake of hard-coding formats or searching for all periods and replacing them with commas. This is not good programming practice.

Microsoft Excel uses the Control Panel Number Format for its separators. So, regardless of what language version of Windows it runs on, it always uses the correct separators. Microsoft recommends that you use the same approach. Always use the separators defined in the number format of control panel.

MORE INFORMATION

All the international information is stored in the WIN.INI file under the [intl] heading. The application should look for the value of "sDecimal." It can query this value using the GetProfileString API. If the application uses this approach, it will have the correct separator for all countries.

To avoid potential problems, never make assumptions about number formats, currency formats (that is, don't hard-code the dollar symbol), date formats, or time formats. These are different for different countries. Use the settings from the Control Panel.

If an application does any text processing, such as sorting or upper/lowercase conversions, it should use the Windows APIs and not supply its own conversion tables and functions. For example, if an application use APIs such as AnsiLower, it will work regardless of language because it obtains the data from the language DLL.

To output floating point numbers in Windows, Microsoft recommends that you use wsprintf. The concept of an "international format string" only has meaning when the string is output to the user. Internal to the computer, for all calculations and manipulations, the concept does not apply. Thus the application can keep its floats in the native format and only convert

to strings before calling the wsprintf function.

There is no need to load conversions from float to string, and vice versa. In other words, there is no need to write any functions such as InterStringToFloat(), InterFloatToString(), and so on.

Here's a simple algorithm you could use.

1. Convert your float to a string by using the _fcvt or _gcvt standard C functions.

2. Get the sThousand and sDecimal separators from the WIN.INI file by using the GetProfileString API.

3. Call wsprintf to output the converted string, using sThousand and sDecimal in the formatting string.

The following is an example of a mistake made by a developer who was not thinking internationally:.

```
if language=English
   Output_English_Float ()
else
   Output_International_Float ()
```

This is not correct. Think of English as just another language and write a single Output_Float() function that covers all languages.

Also, for ease of localization, a developer should place all strings in the resource file. This avoids having to search through all the C files looking for strings to translate.

Additional reference words: 1.20 3.10 3.50 localization kbinf foreign
KBCategory: kbother
KBSubcategory: wintldev

Writing Multiple-Language Resources

PSS ID Number: Q89866
Authored 01-Oct-1992 Last modified 19-May-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) versions 3.1, 3.5, 3.51, and 4.0

When you are writing multiple-language resources, the dialog box identifiers need to be identical for each language instance, as demonstrated below.

```
#define DialogID       100

DialogID DIALOG  0, 0, 210, 10
LANGUAGE LANG_ENGLISH,SUBLANG_ENGLISH_US
.
.
.
DialogID DIALOG  0, 0, 210, 10
LANGUAGE LANG_FRENCH,SUBLANG_FRENCH
```

The FindResource() application programming interface (API) function is used by the system to fetch the dialog box. FindResource() gets the locale information for the process, then attempts to fetch the resource with that language identifier using FindResourceEx(), the language-specific API function for fetching resources. If FindResourceEx() fails to load the language-specific dialog box, FindResource() then attempts to load the neutral dialog box, which should fetch LANG_FRENCH,SUBLANG_FRENCH, if the locale is SUBLANG_FRENCH_CAN or similar.

The LANGUAGE identifiers and the VERSIONINFO language identifiers should also be identical. The code page for resources is always the Unicode code page. The system will translate from Unicode to the required code page.

The preferred method of developing multiple-language resources is to include a LANGUAGE statement for each language supported rather than using the CODEPAGE, LANGUAGE identifier, and VERSIONINFO information. Although the CODEPAGE information will work, the new method is easier to use.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrNls WintlDev

wsprintf() Buffer Limit in Windows

PSS ID Number: Q77255
Authored 10-Oct-1991 Last modified 25-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) for Windows versions 3.0 and 3.1
- Microsoft Win32 SDK, versions 3.1, 3.5, 3.51, and 4.0

The wsprintf(lpOutput, lpFormat [, argument] ...) and wvsprintf() functions format and store a series of characters and values in a buffer specified by the first parameter, lpOutput. This buffer is limited to 1K (1024 bytes); in other words, the largest buffer that wsprintf can use is 1K.

If an application tries to use a buffer larger than 1K, the string will be truncated automatically to a length of 1K.

Additional reference words: 3.00 3.10 3.50 4.00 95
KBCategory: kbprg
KBSubcategory: GdiDrw

XFONT.C from SAMPLES\OPENGL\BOOK Subdirectory

PSS ID Number: Q124870
Authored 15-Jan-1995 Last modified 23-Jun-1995

The information in this article applies to:

- Microsoft Win32 Software Development Kit (SDK) for Windows NT, version 3.5
- Microsoft Visual C++ for Windows NT, version 2.0

SUMMARY

XFONT.C from the SAMPLES\OPENGL\BOOK subdirectory is not in the MAKEFILE, and subsequently is never built.

MORE INFORMATION

XFONT.C should not be built because the sample uses X Windows system-specific functions. To use fonts with Windows NT version 3.5 OpenGL, use the wglUseFontBitmaps() and glCallLists() functions as described in the OpenGL online reference. Another alternative is to use auxDrawStr(). Note that auxDrawStr is defined in GLAUX.H as:

```
#ifdef UNICODE
#define auxDrawStr auxDrawStrW
#else
#define auxDrawStr auxDrawStrA
#endif
void APIENTRY auxDrawStrA(LPCSTR);
void APIENTRY auxDrawStrW(LPCWSTR);
```

REFERENCES

For further information on using fonts with OpenGL, please refer to the OpenGL online reference titled "Fonts and Text."

Additional reference words: 3.50
KBCategory: kbgraphic kbprg
KBSubcategory: GdiOpenGL

XTYP_EXECUTE and its Return Value Limitations

PSS ID Number: Q102574
Authored 04-Aug-1993 Last modified 18-May-1995

The information in this article applies to:

- Microsoft Windows Software Development Kit (SDK) version 3.1
- Microsoft Win32 SDK versions 3.5, 3.51, and 4.0

SUMMARY

A DDEML client application can use the XTYP_EXECUTE transaction to cause a server to execute a command or a series of commands. To do this, the client creates a buffer that contains the command string, and passes either a pointer to this buffer or a data handle identifying the buffer, to the DdeClientTransaction() call.

If the server application generates data as a result of executing the command it received from the client, the return value from the DdeClientTransaction() call does not provide a means for the client to access this data.

MORE INFORMATION

For an XTYP_EXECUTE transaction, the DdeClientTransaction() function returns TRUE to indicate success, or FALSE to indicate failure. In most cases, this provides inadequate information to the client regarding the actual result of the XTYP_EXECUTE command.

Likewise, the functionality that DDEML was supposed to provide through the lpuResult parameter of the DdeClientTransaction() function upon return is currently not supported, and may not be supported in future versions of DDEML. The lpuResult parameter was initially designed to provide the client application access to the server's actual return value (for example, DDE_FACK if it processed the execute, DDE_FBUSY if it was too busy to process the execute, or DDE_FNOTPROCESSED if it denied the execute).

In cases where the server application generates data as a result of an execute command, the client has no means to get to that data, nor does it have a means to determine the status of that execute command through the DdeClientTransaction() call.

An example of this might be one where the DDEML client application specifies a command to a server application such as "OpenFile <FileName>" to open a file, or "DIR C:\WINDOWS" to get a list of files in a given directory.

There are two ways that the client application can work around this limitation and gain access to the data generated from the XTYP_EXECUTE command:

Method 1

The client can issue an XTYP_REQUEST transaction (with the item name set to "ExecuteResult", for example) immediately after its XTYP_EXECUTE transaction call returns successfully. The server can then return a data handle in response to this request, out of the data generated from executing the command.

Method 2

The client can establish an ADVISE loop with the server (with topic!item name appropriately set to Execute!Result, for example) just before issuing the XTYP_EXECUTE transaction. As soon as the server then executes the command, it can immediately update the advise link by calling DdePostAdvise(), and return a data handle out of the data generated from executing the command. The client then receives the data handle in its callback function, as an XTYP_ADVDATA transaction.

Note that these workarounds apply only if one has access to the server application's code. Third-party server applications that provide no means to modify their code as described above can't obtain any data generated by the application as a result of an XTYP_EXECUTE back to the client.

Additional reference words: 3.10 3.50 3.51 4.00 95
KBCategory: kbprg
KBSubcategory: UsrDde

Legal Notice

Olivetti is a registered trademark of Ing. C. Olivetti.

PostScript is a registered trademark of Adobe Systems, Inc.

R4000 is a trademark of MIPS Computer Systems, Inc.

Roland is a registered trademark of Roland Corporation.

SCSI is a registered trademark of Security Control Systems, Inc.

Epson is a registered trademark of Seiko Epson Corporation, Inc.

Silicon Graphics is a registered trademark and OpenGL is a trademark of Silicon Graphics, Inc.

Stacker is a registered trademark of STAC Electronics.

Tandy is a registered trademark of Tandy Corporation.

Unicode is a registered trademark of Unicode, Incorporated.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company, Ltd.

VAX is a trademark of Digital Equipment Corporation

Yamaha is a registered trademark of Yamaha Corporation of America.

Paintbrush is a trademark of Wordstar Atlanta Technology Center.