Windows application developer troubleshooting

Welcome to Windows application troubleshooting. These articles explain how to determine, diagnose, and fix issues that you might encounter when you use Windows applications.

Component development

HOW-TO GUIDE

An application fails to create a COM+ component

CoInitializeEx fails after calling HtmlHelp on the same thread

COM+ supports automatic collection of process dump file

Desktop app UI development

HOW-TO GUIDE

Apps using message filters may not respond in Windows 10

Three- or four-finger touch interactions doesn't work in Windows 11

Call Shell APIs from a multithreaded apartment

Distributed transactions

HOW-TO GUIDE

Can't start transactions

Enable diagnostic tracing

MSDTC Service must run under NetworkService account

New functionality in the MSDTC service

Graphics and multimedia development

HOW-TO GUIDE

Converting PDF to bitmap causes partial data loss in the image

Can't print to XPS Document Writer

Security development

HOW-TO GUIDE

SignTool may corrupt PowerShell script file

A client application may intermittently receive an error message when it tries to create a COM+ component

Article • 12/19/2023

This article helps you resolve the problem where the client application might intermittently receive an error message when it creates a COM+ component.

Original product version: Windows
Original KB number: 911359

Symptoms

When a client application tries to create a Microsoft COM+ component, the client application may intermittently receive an error message.

- Microsoft C++ applications may receive the following error message:

  > E_INVALIDARG: "The parameter is incorrect" (0x80070057/-2147024809)

- Microsoft Visual Basic 6.0 applications may receive the following error message:

  > Run-time error '5': "Invalid procedure call or argument" (0x800a0005/-2146828283)

- Client applications that are built on the Microsoft .NET Framework may receive the following error message:

  > System.ArgumentException: "The parameter is incorrect." at System.Runtime.Type.CreateInstanceImpl(Boolean publicOnly) at System.Activator.CreateInstance(Type type, Boolean nonPublic) (with _HResult = 0x80070057/-2147024809)

The COM+ application will typically function without error for some time immediately after the COM+ application is opened. The problem occurs intermittently, but increases in frequency over time until the application eventually fails on every activation request.

Cause

This problem occurs because the Component Object Model (COM) initialization count for a thread is incremented and decremented when the `CoInitialize` function and the `CoUninitialize` function are called. When this count reaches zero after you call the `CoUninitialize` function, COM will be uninitialized on the thread. When a COM+ STA ThreadPool thread is uninitialized, the thread's apartment activator is destroyed. The next activation request that is routed to this thread will fail with the **E_INVALIDARG** error message. This problem occurs because the thread apartment activator is no longer available. Only activation requests that are routed to the particular uninitialized thread will fail. As more COM+ `ThreadPool` threads become uninitialized, a larger percentage of requests will fail. When all COM+ `ThreadPool` threads become uninitialized, all requests will fail. If the process is restarted, the problem will recover for some time. However, the cycle will be repeated.

Resolution

To resolve this problem, remove the `CoInitialize` calls and the `CoUninitialize` calls from the affected COM DLL. A COM DLL that exposes COM components and is loaded through COM calls from a client should not call the `CoInitialize` function or the `CoUninitialize` function on any thread that is used to load or to start the DLL. These threads are owned by the client application, by the COM runtime, or by the COM+ runtime. Only the COM DLL should call these APIs on additional threads that were explicitly created by the COM DLL. However, it is a common bug for COM DLLs to call the `CoInitialize` function and the `CoUninitialize` function in response to attach events and detach events in the `DllMain` function.

> **Note**
>
> It is correct for a standard Win32 DLL that isn't loaded through COM to call these APIs if the standard Win32 DLL plans to use COM APIs. However, these functions should not be called from the `DllMain` function.

For more information about the `CoInitialize` function, see CoInitialize function.

For more information about the `DllMain` function, see DllMain entry point.

> **Note**
>
> One Microsoft component that contains this bug is the *Cdo.dll* component. This component isn't supported in multithreaded environments. Instead, we

recommend that you use the *Cdosys.dll* component or the *Cdonts.dll* component.

More information

To determine which COM component contains the affected code, use the Microsoft Internet Information Services (IIS) Debug Diagnostic Tool (DebugDiag). To do this, follow these steps:

1. Download and install DebugDiag.

2. Create a crash rule in DebugDiag. To do this, follow these steps:
   a. Click **Start**, point to **Programs**, point to **IIS Diagnostics (32bit)**, point to **Debug Diagnostics Tool**, and then click **Debug Diagnostics Tool 1.0**.
   b. If the **Select Rule Type** dialog box opens, click **Cancel**.
   c. On the **Tools** menu, click **Options And Settings**.
   d. Click the **Folders and Search Paths** tab, and then enter `srv*C:\symsrv\*http://msdl.microsoft.com/download/symbols` in the **Symbol Search Path For Analysis** field.
   e. In the **Symbol Search Path For Debugging** field, enter `srv*C:\symsrv\*http://msdl.microsoft.com/download/symbols`, and then click **OK**.
   f. Click **Add Rule**.
   g. Click **Crash**, and then click **Next**.
   h. Click **A specific MTS/COM+ application (included high and medium isolation websites)**, and then click **Next**.
   i. Click the appropriate COM+ application, and then click **Next**.
   j. In the **Advanced Configuration (Optional)** dialog box, click **Breakpoints**.
   k. In the **Configure Breakpoints** dialog box, click **Add Breakpoint**.
   l. In the **Configure Breakpoint** dialog box, enter *ole32!CoInitializeEx* in the **Breakpoint Expression** field, enter *1000* in the **Action Type** field, keep **Log Stack Trace** in the **Action Type** field, and then click **OK**.
   m. In the **Configure Breakpoints** dialog box, click **Add Breakpoint**.
   n. In the **Configure Breakpoint** dialog box, enter *ole32!CoUninitialize* in the **Breakpoint Expression** field, enter *1000* in the **Action Limit** field, and then click **OK**.
   o. In the **Configure Breakpoints** dialog box, click **Save & Close**.
   p. In the **Advanced Configuration (Optional)** dialog box, click **Next**.
   q. In the **Select Dump Location And Rule Name (Optional)** dialog box, click **Next**.
   r. In the **Rule Completed** dialog box, click **Activate the rule now**, and then click **Finish**.

DebugDiag monitors the selected COM+ application when the application runs. Every time that DebugDiag experiences one of the selected breakpoints, DebugDiag adds data to a log file. By default, the log file will be named:
`C:\Program Files\IIS Resources\DebugDiag\Logs\dllhost__PID__\<pid>\__Date__\<date> __Time\_\<time> Log.txt`

When an error occurs in the application, review the appropriate log file to find the affected DLL. In the following example, the affected COM component is *Mybaddll.dll*. The following call stacks are examples of expected API calls that are directly from the COM+ runtime:

Console

```
[10/27/2005 10:03:42 AM] Breakpoint at ole32!CoInitializeEx caused by 3500
[10/27/2005 10:03:42 AM] Stack Trace
ChildEBP RetAddr Args to Child
0097ff38 7668c062 00000000 00000002 7c910732 ole32!CoInitializeEx
0097ff80 77c3a3b0 000de370 7c910732 00000005
COMSVCS!CSTAThread::WorkerLoop+0x6c
0097ffb4 7c80b50b 0003e018 7c910732 00000005 msvcrt!_endthreadex+0xa9
0097ffec 00000000 77c3a341 0003e018 00000000 kernel32!BaseThreadStart+0x37

[10/27/2005 10:13:29 AM] Breakpoint at ole32!CoUninitialize caused by 1300
[10/27/2005 10:13:29 AM] Stack Trace
ChildEBP RetAddr Args to Child
006dff08 766f965b 00000000 000b9838 00037138 ole32!CoUninitialize
006dff70 766f9742 000b9838 00037258 006dffb4
COMSVCS!WORK_QUEUE::WorkerLoop+0x248
006dff80 77c3a3b0 000b9838 00000000 7c9105c8
COMSVCS!WORK_QUEUE::ThreadLoop+0x19
006dffb4 7c80b50b 00037138 00000000 7c9105c8 msvcrt!_endthreadex+0xa9
006dffec 00000000 77c3a341 00037138 00000000 kernel32!BaseThreadStart+0x37

[10/27/2005 10:13:29 AM] Breakpoint at ole32!CoUninitialize caused by 2188
[10/27/2005 10:13:29 AM] Stack Trace
ChildEBP RetAddr Args to Child
0007fd98 0100128e 00092388 00000000 0007fdbc ole32!CoUninitialize
0007ff1c 010015b0 01000000 00000000 00092388 dllhost!WinMain+0xd0
0007ffc0 7c816d4f 00098610 005df0fc 7ffdb000 dllhost!WinMainCRTStartup+0x174
0007fff0 00000000 0100143c 00000000 78746341 kernel32!BaseProcessStart+0x23
```

The following call stacks are examples of unexpected API calls from a custom DLL:

Console

```
[10/27/2005 10:03:49 AM] Breakpoint at ole32!CoInitializeEx caused by 3500
[10/27/2005 10:03:42 AM] Stack Trace
ChildEBP RetAddr Args to Child
0097e684 1001349c 00000000 00000001 0097e7cc ole32!CoInitialize
0097e76c 100293ca 10000000 00000001 00000000 MyBadDLL!DllMain+0x4c
```

```
0097e7b8 7c9011a7 10000000 00000001 00000000
MyBadDLL!_DllMainCRTStartup+0xca
0097e7d8 7c91cbab 10011712 10000000 00000001 ntdll!LdrpCallInitRoutine+0x14
0097e8e0 7c916178 00000000 c0150008 00000000
ntdll!LdrpRunInitializeRoutines+0x344
0097eb8c 7c9162da 00000000 000c6a30 0097ee80 ntdll!LdrpLoadDll+0x3e5
0097ee34 7c801bb9 000c6a30 0097ee80 0097ee60 ntdll!LdrLoadDll+0x230
0097ee9c 7752e1b1 0097ef18 00000000 00000008 kernel32!LoadLibraryExW+0x18e
0097eec0 7752e0cd 0097ef18 0097eee4 0097eee8
ole32!CClassCache::CDllPathEntry::LoadDll+0x6c
0097eef0 7752d550 0097ef18 0097f1f4 0097ef10
ole32!CClassCache::CDllPathEntry::Create_rl+0x37
0097f13c 7752d473 00000001 0097f1f4 0097f16c
ole32!CClassCache::CClassEntry::CreateDllClassEntry_rl+0xd6
0097f184 7752d3d1 00000001 000ba820 0097f1ac
ole32!CClassCache::GetClassObjectActivator+0x195
0097f1b0 7752cf3b 0097f1f4 00000000 000d82c4
ole32!CClassCache::GetClassObject+0x23
0097f22c 7752cddf 77607150 00000000 000d82c4
ole32!CServerContextActivator::CreateInstance+0x106
0097f26c 76672d76 000d82c4 00000000 0097f33c
ole32!ActivationPropertiesIn::DelegateCreateInstance+0xf7
0097f2e0 7752cddf 000cba80 00000000 000d82c4
COMSVCS!CObjectActivator::CreateInstance+0x21c
0097f320 7759699a 000d82c4 00000000 0097f33c
ole32!ActivationPropertiesIn::DelegateCreateInstance+0xf7
0097f340 775877e1 000cf170 000b492c 00000000 ole32!DoServerContextCCI+0x1d
0097f38c 775880cf 00000000 000b492c 7759697d ole32!EnterForCallback+0xc2
0097f4ec 77563258 0097f3c4 7759697d 000cf170 ole32!SwitchForCallback+0x1a3
0097f518 7751929f 000b492c 7759697d 000cf170 ole32!PerformCallback+0x54
0097f5b0 7758a738 00099940 7759697d 000cf170
ole32!CObjectContext::InternalContextCallback+0x155
0097f5d0 77596ba1 00099940 7759697d 000cf170
ole32!CObjectContext::DoCallback+0x1c
0097f5fc 77567439 77607154 000d82c4 000d0ba0
ole32!CApartmentActivator::ContextCallHelper+0x4e
0097f658 77e79dc9 77607154 00000000 000d82c4
ole32!CApartmentActivator::CreateInstance+0x110
0097f67c 77ef321a 7752cfbc 0097f690 00000004 RPCRT4!Invoke+0x30
0097fa88 77ef3bf3 000d8ea0 000cfc04 000cecb4 RPCRT4!NdrStubCall2+0x297
0097fae0 77600c31 000d8ea0 000cecb4 000cfc04
RPCRT4!CStdStubBuffer_Invoke+0xc6
0097fb20 77600bdb 000cecb4 000b689c 00000000 ole32!SyncStubInvoke+0x33
0097fb68 7750f237 000cecb4 000d5de8 000d8ea0 ole32!StubInvoke+0xa7
0097fc40 7750f15c 000cfc04 00000000 000d8ea0
ole32!CCtxComChnl::ContextInvoke+0xe3
0097fc5c 7750fc79 000cecb4 00000001 000d8ea0 ole32!MTAInvoke+0x1a
0097fc88 77600e3b 000cecb4 00000001 000d8ea0 ole32!STAInvoke+0x4a
0097fcbc 776009bc 000cec60 000cfc04 000d8ea0 ole32!AppInvoke+0x7e
0097fd90 77600df2 000cec60 000b1b10 00000000
ole32!ComInvokeWithLockAndIPID+0x2e0
0097fdbc 7750fcb3 000cec60 00000400 000d1d88 ole32!ComInvoke+0x60
0097fdd0 7750fae9 000cec60 0097fe50 7750fa56 ole32!ThreadDispatch+0x23
0097fde8 77d48734 002f07c8 000c7d08 0000babe ole32!ThreadWndProc+0xfe
0097fe14 77d48816 7750fa56 002f07c8 00000400 USER32!InternalCallWinProc+0x28
```

```
0097fe7c 77d489cd 00000000 7750fa56 002f07c8
USER32!UserCallWinProcCheckWow+0x150
0097fedc 77d48a10 000de3fc 00000000 0097ff08
USER32!DispatchMessageWorker+0x306
0097feec 7668b57c 000de3fc 000de3f8 000de370 USER32!DispatchMessageW+0xf
0097ff08 7668b42d 000de390 000de370 000de3ec
COMSVCS!CSTAQueueLessMessageWork::DoWork+0x4e
0097ff20 7668bea5 000de3f8 7668c2d0 000de370 COMSVCS!CSTAThread::DoWork+0x18
0097ff40 7668c197 7c910732 0003c9a8 0003e018
COMSVCS!CSTAThread::ProcessQueueWork+0x47
0097ff80 77c3a3b0 000de370 7c910732 00000005
COMSVCS!CSTAThread::WorkerLoop+0x1a1
0097ffb4 7c80b50b 0003e018 7c910732 00000005 msvcrt!_endthreadex+0xa9
0097ffec 00000000 77c3a341 0003e018 00000000 kernel32!BaseThreadStart+0x37

[10/27/2005 10:08:42 AM] Breakpoint at ole32!CoUninitialize caused by 628
[10/27/2005 10:08:42 AM] Stack Trace
ChildEBP RetAddr Args to Child
0097f9f4 100134e5 0097fb7c 0097fb38 10011712 ole32!CoUninitialize
0097fad8 100293ca 10000000 00000000 00000000 MyBadDLL!DllMain+0x95
0097fb24 7c9011a7 10000000 00000000 00000000
MyBadDLL!_DllMainCRTStartup+0xca
0097fb44 7c91e6f4 10011712 10000000 00000000 ntdll!LdrpCallInitRoutine+0x14
0097fc3c 7c80aa7f 10000000 0097fc90 0097fdb0 ntdll!LdrUnloadDll+0x41c
0097fc50 77513442 10000000 0097fdd0 77513456 kernel32!FreeLibrary+0x3f
0097fc5c 77513456 0097fc9c 776067e0 00000000
ole32!CClassCache::CDllPathEntry::CFinishObject::Finish+0x2f
0097fc70 775135fe 774e1ab0 00000000 00000000
ole32!CClassCache::CFinishComposite::Finish+0x1d
0097fdd0 77513578 ffffffff 000c7d08 0097fe1c
ole32!CClassCache::FreeUnused+0x19d
0097fde0 775133a2 ffffffff 00000000 7668b359
ole32!CoFreeUnusedLibrariesEx+0x36
0097fdec 7668b359 77d48734 00000000 00000113 ole32!CoFreeUnusedLibraries+0x9
0097fdf0 77d48734 00000000 00000113 0000705b COMSVCS!STAFreeLibTimerProc+0x6
0097fe1c 77d49857 7668b353 00000000 00000113 USER32!InternalCallWinProc+0x28
0097fe84 77d49791 00000000 7668b353 00000000 USER32!UserCallWinProc+0xf3
0097fedc 77d48a10 000de3fc 00000000 0097ff08
USER32!DispatchMessageWorker+0x10e
0097feec 7668b57c 000de3fc 000de3f8 000de370 USER32!DispatchMessageW+0xf
0097ff08 7668b42d 000de390 000de370 000de3ec
COMSVCS!CSTAQueueLessMessageWork::DoWork+0x4e
0097ff20 7668bea5 000de3f8 7668c2d0 000de370 COMSVCS!CSTAThread::DoWork+0x18
0097ff40 7668c197 7c910732 0003c9a8 0003e018
COMSVCS!CSTAThread::ProcessQueueWork+0x47
0097ff80 77c3a3b0 000de370 7c910732 00000005
COMSVCS!CSTAThread::WorkerLoop+0x1a1
0097ffb4 7c80b50b 0003e018 7c910732 00000005 msvcrt!_endthreadex+0xa9
0097ffec 00000000 77c3a341 0003e018 00000000 kernel32!BaseThreadStart+0x37
```

Feedback

Was this page helpful? Yes No

CoInitializeEx function fails after calling the HtmlHelp function on the same thread

Article • 12/19/2023

This article discusses an issue where the CoInitializeEx function fails after calling the `HtmlHelp` function on the same thread.

Applies to: All supported operating system

Symptoms

If an application calls `HtmlHelp` before calling `CoInitializeEx` with the specified `COINIT_MULTITHREADED` value, `CoInitializeEx` can return `RPC_E_CHANGED_MODE (0x80010106)`. As a result, the application may crash, hang, or display unexpected behavior.

Cause

If a thread that calls `HtmlHelp` hasn't been initialized with `CoInitialize` or `CoInitializeEx`, `HtmlHelp` initializes the thread as apartment-threaded with `COINIT_APARTMENTTHREADED`.

Workaround

To work around the issue and avoid the conflict of COM initialization on a single thread, create a new thread and call `HtmlHelp` on that thread.

Feedback

COM+ supports automatic collection of process dump file and process termination in Windows Server

Article • 12/19/2023

Applies to: Windows SDK for Windows 10
Original KB number: 910904

Introduction

The system logs an event when a COM+ component experiences an unusually high call time. The event log identifies the COM+ component that experiences the problem. Additionally, the event log mentions this article (910904). The system can be configured to perform one or both of these actions:

- Automatically collect a process dump file for root cause analysis of the problem.
- Terminate the process to help recover from the problem without manual intervention.

After the system has collected a dump file, use the Debug Diagnostics Tool (DebugDiag) to generate a report that describes the problem and provides known solutions.

Default behavior

Consider the following scenario:

- The call time for a COM+ component exceeds 10 minutes.
- You open the Component Services Microsoft Management Console (MMC) snap-in while the application that hosts this long-running COM+ component is running.

In this scenario, the following event is logged in the Application log:

Output

```
Event Type: information
Event Source Information: COM+
COM+ Event Category: (117)
Event ID: 782
Description: The average call duration exceeded 10 minutes.
If this is not the expected behavior, see Microsoft Knowledge Base Article
```

```
910904 in http://support.microsoft.com for detailed information about
how to use the COM+ AutoDump feature to automatically generate dump files
and terminate the process if the problem recurs.
Server application ID: <YourAppID>
Server application instance ID: <YourAppInstanceID>
Server application name: <YourAppName>
```

Configuration options

> **Important**
>
> This section involves modifications to the registry. Follow the steps carefully. Serious problems might occur if you modify the registry incorrectly. Back up the registry as a precaution. For more information about how to back up and restore the registry, see **How to back up and restore the registry in Windows**.

The system can be configured to perform one or both of the following actions when a long-running COM+ component is detected:

- Automatically collect a process dump file.
- Terminate the process.

  To do this, use the following registry values:

Expand table

Value name	Data type	Description	Default value
`AverageCallThreshold`	`REG_DWORD`	Threshold, in seconds, when the appropriate actions will be taken	**0**
`DumpType`	`REG_DWORD`	0 = Generate a full dump file; 1 = Generate a minidump file; 2 = No dump file	**0**
`Terminate`	`REG_DWORD`	0 = Process will continue; 1 = Process will be terminated	**0**

- To globally define actions for all COM+ components on the computer, add the configuration values under the following registry key:
  `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\COM3\AutoDump`

- To define actions to be taken for a specific COM+ component regardless of the global settings, add the configuration values under the following registry key: `HKEY_CLASSES_ROOT\AppId\{<YourAppID>}\AutoDump\{<YourCLSID>}`

Recommendations

The following content shows how to check and analyze the full dump files.

Collect full dump files

Collect a full dump file when a COM+ component experiences an unusually high call time. For example, create the following single registry value:
`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\COM3\AutoDump AverageCallThreshold = 300`

See the Considerations section for more information about how to select an appropriate `AverageCallThreshold` registry value for your particular environment.

Similarly, collect a full dump file when an unhandled exception occurs in a COM+ application. To do so, check the **Enable Image Dump on Application Fault** check box on the **Dump** tab in the properties of each COM+ application.

Analyze the dump files

To analyze the dump file, follow these steps:

1. Download and install the Debug Diagnostic Tool (DebugDiag).
2. Use the Debug Diagnostics Tool to generate an analysis report for the dump file by following these steps:
   a. Run the **DebugDiag Analysis** application from the **Start** menu.
   b. Select **Settings**, then select the **Preferences** tab.
   c. Make sure that the **Microsoft Public Symbol Servers** option in the **Symbol Search Path** field is checked, then select **Back**.
   d. Select the **Default Analysis \ CrashHangAnalysis** option.
   e. Select **Add Data Files**.
   f. Select the dump file(s) that you want to analyze.
   g. Select **Start Analysis**.

   The resulting HTML report is displayed in a new Microsoft Internet Explorer window on the desktop and saved to the DebugDiag Reports directory. The default location for this directory is *%USERPROFILE%\Documents\DebugDiag\Reports*.

3. To resolve the problem, follow the guidance that is provided in the Recommendations section of the report. This section of the report may recommend the following things:

    - It may direct you to a Microsoft Knowledge Base article that describes known issues.
    - It may provide the developers of the application with information that they can use to make corrections.
    - It may suggest that you follow up with the appropriate vendor or with Microsoft Support. When you contact Microsoft Support for more help, provide the report file to speed the analysis process. The full dump file might also be required.

Considerations

Here are some factors you should consider.

AverageCallThreshold registry value

A value of 300 seconds is an appropriate threshold for most environments. The ideal value depends on the particular environment. To ensure action is taken as quickly as possible, but only when a legitimate problem occurs, select the smallest possible value that is exceeded only in a problematic scenario.

TerminateProcess registry value

Terminating the process when high call times occur may help the COM+ component to automatically recover from certain problems. This is desirable in environments where high availability is important. When using this feature, select an appropriate `AverageCallThreshold` registry value to avoid unintentionally terminating the process.

DumpType registry value

Minidump files can be created faster and occupy less disk space than full dump files. However, they aren't as useful to analyze problems because they frequently lack the required data. The typical size of full dump files for a *Dllhost.exe* process ranges from 10 megabytes (MB) to 50 MB. Their actual size depends on the size of the working set of the dumped process. The files are normally generated within seconds.

Dump file options

By default, the dump files are stored in the *%systemroot%\system32\com\dmp* directory. Use the settings in the **Image Dump Directory** box and under the **Maximum Number of Dump Images** area for the appropriate COM+ application to control the location and the number of dump files.

Call time

The call time for a COM+ component is a running average for all instances of the COM+ component. The call time is calculated by the COM+ System Application and is displayed in the **Call Time (ms)** column of the Status View in the Component Services MMC snap-in.

Feedback

CreateProcessWithLogonW fails when called on a Microsoft Entra account

Article • 12/19/2023

The CreateProcessWithLogonW function may fail when you call it by setting `lpDomain` to a Microsoft Entra ID (formerly known as Azure Active Directory) account in UPN format.

Symptoms

The CreateProcessWithLogonW function documentation states that the `lpDomain` parameter must be set to `NULL` if `lpUsername` is in UPN format.

However, the `CreateProcessWithLogonW` function may fail, and the `GetLastError` function returns `ERROR_LOGON_FAILURE (1326)` or another error code on Windows 10 and Windows 11 when the following conditions are met:

- The `lpUsername` parameter is set to a Unicode string containing a Microsoft Entra account, specified in UPN format.
- The `lpDomain` parameter is set to `NULL`.

Cause

Microsoft has confirmed this is a problem in Windows 10 and Windows 11.

Workaround

To work around this issue, follow these steps:

1. Call `CreateProcessWithLogonW` by setting the `lpUsername` parameter to a user account in UPN format and the `lpDomain` parameter to `NULL`.
2. If step 1 fails, call `CreateProcessWithLogonW` again with the `lpDomain` parameter set to `AzureAD`.

Feedback

Was this page helpful? Yes No

A user-defined application using message filters may become unresponsive in Windows 10, version 2004/20H2/21H1/21H2

Article • 12/19/2023

This article helps you resolve the problem when an application using its own message filters stops responding in Windows 10.

Applies to: Windows 10, version 2004, Windows 10, version 20H2, Windows 10, version 21H1, Windows 10, version 21H2

Symptoms

Consider the scenario where you run an application on Windows 10, version 2004/20H2/21H1/21H2 and your application is using message filters. In this scenario, the application may become unresponsive.

> **Note**
>
> This issue is not observed in Windows 11.

Cause

Windows 10 adds Windows messages used by text input systems or Text Services Framework (TSF).

> **Note**
>
> Windows 10, version 2004 introduced the new version of TSF.

If the message filter of the application removes Window messages using `PeekMessage` API or `GetMessage` API and doesn't pass messages to `DispatchMessage` API, TSF can't complete processing the messages and the application may stop responding.

This problem may occur if the application has the message filter similar to the following example, which dispatches `WM_LBUTTONUP` messages only and removes other messages.

C++

```
while(::PeekMessage(&msg,NULL,0,0,PM_REMOVE))
{
    ::TranslateMessage(&msg);

    // Dispatch only specific messages.
    if (msg.message == WM_LBUTTONUP) {
        ::DispatchMessage(&msg);
    }
}
```

Workaround 1

Modify the message filter to only filter the required messages and to dispatch other messages through `DispatchMessage` API.

C++

```
while(::PeekMessage(&msg,NULL,0,0,PM_REMOVE))
{
    ::TranslateMessage(&msg);

    if (msg.message == WM_LBUTTONDOWN) {

    }
    else {
        // Dispatches all non-filtered messages
        ::DispatchMessage(&msg);
    }
}
```

Workaround 2

Enable the compatibility mode of the Microsoft IME (Input Method Editor), if you're using the new Microsoft IME in Windows 10.

For more information about how to use previous version of the Microsoft IME, see Revert to a previous version of an IME (Input Method Editor).

> **Note**
>
> We recommend that you use the IME compatibility setting as a temporary workaround.

Feedback

Was this page helpful? Yes No

Three-finger and four-finger touch interactions don’t work in Windows apps

Article • 12/19/2023

By default, Windows 11 uses three-finger and four-finger touch interactions for various operations, such as switching or minimizing windows, and changing virtual desktops. Because these interactions are handled at the system level, the functionality of various applications might be affected by this change.

To support three-finger or four-finger touch interactions within an application, a new user setting is introduced that specifies how the system handles these interactions.

Bluetooth & devices > Touch > Three- and four-finger touch gestures

- If the setting is set to **On** (default), the system can use three-finger and four-finger interactions, but applications can't use them.
- If the setting is set to **Off**, applications can use three-finger and four-finger interactions, but the system can't use them.

If a particular application must support these interactions, we recommend that you inform users about this setting, and provide a link that starts the Windows Settings app and opens the relevant page (ms-settings:devices-touch). For more information, see Launcher.LaunchUriAsync Method.

Feedback

Calling shell functions and interfaces from a multithreaded apartment

Article • 12/19/2023

When you call or access a shell function or shell interface from a thread that has been initialized as a multithreaded apartment, the function or interface may have its functionality impaired or completely fail.

Original version: Windows shell and Interface
Original KB number: 287087

Cause

A call to `CoInitializeEx (COINIT_MULTITHREADED)` allows calls to objects created on the calling thread to be run on any thread. When accessing objects that use the apartment threading model from a multithreaded apartment, COM will synchronize access to the object. In order for this synchronization to occur, COM must marshal calls to the object. Because the shell currently doesn't provide the necessary information, either through a type library or proxy/stub code, for its objects to be marshaled, attempts to access shell objects from a multithreaded apartment fail.

Calls that can affect shell functions

The following are examples of how calls to `CoInitializeEx (COINIT_MULTITHREADED)` can affect functions that rely on shell objects:

- GetOpenFileName/GetSaveFileName

  Users can navigate to namespace extension folders such as **My Documents** through the **Open and Save As** dialog box. However, these folders can't be browsed to because the browser can't create the required interfaces, such as `IShellFolder`.

- ShellExecute/ShellExecuteEx

  `ShellExecute` hooks can be written to extend the functionality of `ShellExecute` or `ShellExecuteEx` by implementing the `IShellExecuteHook` interface. When `ShellExecute` or `ShellExecuteEx` is called, registered `ShellExecute` hooks can't be loaded.

In both of these examples, the component that is attempting to obtain an interface pointer to a shell object with `CoCreateInstance`, `IUnknown::QueryInterface`, and so forth, will typically fail with error `E_NOINTERFACE` when called from multithreaded apartments. The reason, as noted above, is that there's no type information or proxy/stub code for the objects being requested.

References

Process, Threads, and Apartments

Feedback

Explorer-style common file dialog box doesn't display folder names longer than 67 characters

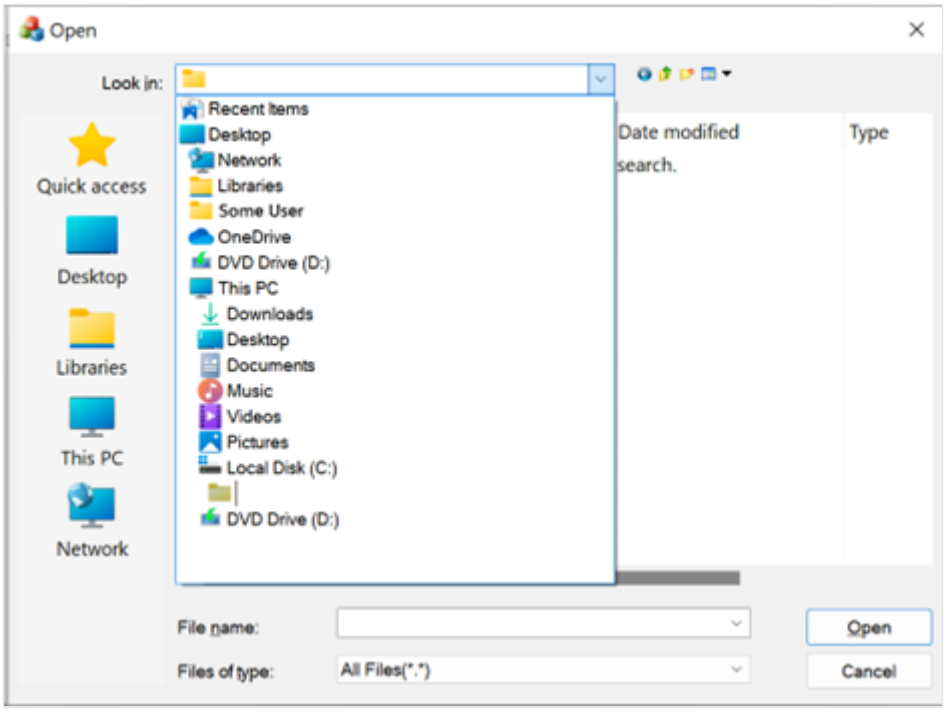
Article • 04/09/2024

Symptoms

Consider the following scenario:

- You use the Explorer-style common file dialog box in an application to open or save a document in a folder.
- You navigate to a folder whose name is longer than 67 characters.

In this scenario, you see that the **Look in** combo box is blank.



Cause

This problem occurs in applications that use the `GetOpenFileName` and `GetSaveFileName` functions to show Explorer-style common file dialog boxes. The `GetOpenFileName` and `GetSaveFileName` functions return the correct path when a file is selected, even though the folder name isn't displayed.

Status

Microsoft has confirmed this is a problem in Windows 7, Windows Server 2008, and later versions.

More information

Applications that use IFileDialog to show common file dialog boxes aren't affected.

Feedback

East Asian language first character not recognized in DataGrid cell

Article • 12/19/2023

This article helps you resolve the problem where the first input character for East Asian languages isn't recognized correctly in DataGridView cell on Windows 10.

Original product version: Windows 10
Original KB number: 4563779

Symptoms

The first input character for East Asian Languages isn't recognized correctly by IME in DataGrid cell.

Cause

The input composition for Edit control doesn't include the first character typed into the DataGrid cell. The text entered could therefore be incorrect. The Edit control content must be cleared to ensure the correct text is entered. This is an application compatibility issue. Changing compatibility registry is workaround.

Resolution

> **Important**
>
> This section explains how to modify the registry. Improper modifications can cause serious issues. Follow the steps carefully to avoid any mistake. For added protection, back up the registry so that it can be restored if a problem occurs.

For more information about how to back up and restore the registry, see: How to back up and restore the registry in Windows.

There are two registry keys to address this problem. Customers can apply one of the following registry key to the system.

Consider the following scenario.

- If you have multiple applications that encounter this problem, and each application has a different Window Class name for each DataGrid cell. In this case, you can add the new registry key, which contains an executable file name of application. Then, you can set the value to 0x00008000. You'll need to repeatedly set up the registry keys for every single application.

- If you have multiple applications that encounter this problem but your applications use single-Window Class name for DataGrid cell because all of those applications' Window Class names are the same. In this case, you can add `AppCompatClassName` registry key. Then, you can set the value to Window class name of your application.

1. **For specific process name:**
   Registry entry

   > HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\CTF\Compatibility\
   > <ExecutableFileName>
   > REG_DWORD: Compatibility
   > DWORD Value: 0x00008000 (Hex value of 32,768)

   If x86 applications are executed on a x64 Windows system, the following registry key can be applied instead of the one mentioned previously:

   > HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\CTF\Compatibility\<ExecutableFileName>
   > REG_DWORD: Compatibility
   > DWORD Value: 0x00008000 (Hex value of 32,768)

   For example: The workaround for a specific executable file name such as `sample.exe`

   > HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\CTF\Compatibility\sample.exe
   > REG_DWORD: Compatibility
   > DWORD Value: 0x00008000 (Hex value of 32,768)

2. **For specific Windows class name:**
   If you use this scenario, you have to apply the following Windows Updates on your system.

Expand table

Windows 10 Version	Article link
Windows 10 Version 1803	KB4550944
Windows 10 Version 1809	KB4550969
Windows 10 Version 1903	KB4541335
Windows 10 Version 1909	KB4541335
Windows 10 Version 2004	KB4571744

Registry entry:

> HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\CTF\Compatibility\AppCompat ClassName
> REG_SZ: Compatibility
> String Value: <WindowClassName>

If x86 applications are executed on an x64 Windows system, the following registry key can be applied instead of the one mentioned above:

> HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\CTF\Compatibility\AppCompatClassName
> REG_SZ: Compatibility
> String Value: <WindowClassName>

For example: The workaround for specific Window Class Name as **Edit**

> KEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\CTF\Compatibility\AppCompatClassName
> REG_SZ: Compatibility
> String Value: Edit

References

Learn about the terminology that Microsoft uses to describe software updates.

Feedback

Half-width and Full-width Katakana and Hiragana characters with consonant marks are treated as different

Article • 12/19/2023

This article helps you resolve a problem that occurs when .NET Framework 4.*x* applications compare Japanese strings.

Applies to: Windows 10 version 2004, Windows 10 version 20H2, Windows 10 version 21H1, Windows 10 version 21H2, Windows 10 version 22H2

Symptoms

Certain Japanese half-width and full-width Katakana and Hiragana characters that have a consonant mark aren't interpreted as the same character. When you use the `CompareInfo.IndexOf` method and the `IgnoreKanaType` or `IgnoreWidth` options as `CompareOptions` to make a comparison, these characters are evaluated as different because of an issue in the sorting rule.

Cause

Starting in version 2004, Windows 10 updated the version of National Language Support (NLS) to 6.3 and added support for Arabic and Hebrew. This addition affects the rules for sorting Japanese string comparisons that use NLS so that the comparisons will produce different results.

Workaround

> ⚠ **Warning**
>
> Serious problems might occur if you modify the registry incorrectly. These problems could cause you to have to reinstall the operating system or even prevent your machine from starting. Microsoft can't guarantee that these problems can be solved. Before you modify it, **back up the registry for restoration** in case problems occur. Modify the registry at your own risk.

Workaround 1

Revert the NLS sorting rule to version 6.2. This version is used in Windows 10, version 1909 and earlier versions. When you have to share data between systems, consider applying the workaround consistently. If you use this workaround, do sufficient testing and evaluations to mitigate problems that are caused by different sorting rule versions on multiple systems.

To use this workaround, follow these steps:

1. Open a Command Prompt window (*cmd.exe*) as an administrator.
2. Run the following command:

   ```
   reg add HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Nls\Sorting\Versions /ve /d 0006020F /f
   ```

3. Restart the computer or processes to see the full effect.

> **Important**
>
> If you haven't installed KB4586853 or a later update on the computer, setting an invalid value in this registry entry might prevent the computer from starting.

Workaround 2

Set the NLS sorting rule to version 6.4. This version is used in Windows 11. To do this, you must apply KB5014023 or a later update, and upgrade the version to version 1741 or a later version. In this case, version 22H2 is already applied and doesn't have to be updated.

1. Apply KB5014023 or a later update for Windows 10 versions 2004, 20H2, 21H1, and 21H2.
2. Open the Command Prompt window (*cmd.exe*) as an administrator.
3. Run the following command:

   ```
   reg add HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Nls\Sorting\Versions /ve /d 00060403 /f
   ```

4. Restart the computer or processes to see the full effect.

Feedback

Was this page helpful? Yes No

IME crashes when processing a window message sent from another thread

Article • 12/19/2023

Describes a scenario where an Input Method Editor (IME) crashes while processing a window message sent from another thread, and where the window procedure handling the message calls an `Imm*` function such as `ImmSetOpenStatus`.

Symptoms

An application using an IME crashes under one of these conditions:

- A user interface (UI) thread calls the `TranslateMessage` function as a part of the message loop.
- Another thread sends a window message to a window owned by the UI thread.
- The window procedure handling the window message sent by the other thread calls an `Imm*` function, such as `ImmSetOpenStatus`.

Cause

An IME included with Windows 10 may call the `PeekMessage` function when the IME is called by the `TranslateMessage` function to process a keyboard input. `PeekMessage` will process any pending window messages sent by other threads. This can result in a reentrancy issue when the window procedure processing the sent message calls an `Imm*` function and leaves the IME in an unexpected state.

Workaround

Avoid calling `Imm*` functions while the IME is processing another window message.

More information

The PeekMessage documentation contains the following:

During this call, the system delivers pending and non-queued messages sent to windows owned by the calling thread using the SendMessage, SendMessageCallback, SendMessageTimeout, or SendNotifyMessage function. The first queued message

matching the specified filter is retrieved. The system may also process internal events. If no filter is specified, messages are processed in this order:

- Sent messages
- Posted messages
- Input (hardware) messages and system internal events
- Sent messages (again)
- WM_PAINT messages
- WM_TIMER messages

> **Note**
>
> Sent window messages are non-queued messages. The message filter specified when calling PeekMessage call doesn't apply to sent messages.

Feedback

Japanese handwritten characters aren't recognized using Microsoft.Ink.Recognizer

Article • 04/09/2025

Symptoms

When using `Microsoft.Ink.Recognizer` as the recognition engine on Windows 11 version 24H2 and Windows Server 2025, you might encounter the following issues:

- Issue 1

  The Japanese handwritten characters aren't recognized correctly.

  For example, when you handwrite Japanese `カスイ`, the result is `カイ`, which isn't expected.

- Issue 2

  When you perform character recognition with specified stroke data, the specified character range isn't correctly applied.

  For example, when you handwrite `あいう`, specify the character range as `ラ リ ル レ ロ`, and then execute handwritten character recognition, the actual result is `あいう`. However, the expected result is `ロロラ`.

Status

Microsoft has confirmed this is a problem in Windows 11 version 24H2 and Windows Server 2025.

File Explorer's navigation bar doesn't update when navigating shell namespace extension folders

Article • 09/03/2024

This article describes a problem where File Explorer's navigation bar doesn't update when navigating folders in a shell namespace extension.

Symptoms

After you install KB 5031455 on Windows 11, version 22H2 or Windows 11, version 23H2, File Explorer's navigation bar doesn't update to show the current folder when navigating folders in some shell namespace extensions. Additionally, File Explorer's navigation bar can't be used to navigate folders in the same shell namespace extension.

Status

This issue is fixed in KB5041587.

More information

With a namespace extension, application developers can take any body of data and have Windows Explorer present it to the user as a virtual folder. When a user browses this folder, the application data is presented as a tree-structured hierarchy of folders and files. Users and applications can interact with the contents of this virtual folder in much the same way as with any other namespace object.

For more information, see the following articles:

- Introduction to the Shell Namespace
- Understanding Shell Namespace Extensions

Feedback

The owner window of a modal dialog can get activated when taskbar buttons are created for both windows

Article • 12/19/2023

Symptoms

Consider the following scenario:

- A .NET desktop application is created by using Windows Forms or Windows Presentation Foundation (WPF).
- The application uses the `ShowDialog()` method to display a modal dialog that is owned by another window.
- The application creates taskbar buttons for both windows by setting the `ShowInTaskbar` property value to `true`.

> **Note**
>
> The window has a taskbar button if the property value is `true`. The default value is `true`.

In this scenario, the owner window can be activated by selecting its associated taskbar button. However, activating the owner window of a modal dialog can cause unexpected behavior or crashes in some applications, as the owner window should remain inactive and disabled until the modal dialog is dismissed.

Resolution

Use the following method corresponding to your situation to create a single taskbar button for windows in a window ownership group, which includes the top-level window and any owned windows:

- If you use a `System.Windows.Forms.Form` object to display a modal dialog with an owner window, set the Form object's `ShowInTaskbar` property value to `false` prior to calling the `ShowDialog` method.

- If you use a `System.Windows.Window` object to display a modal dialog with an owner window, set the Window object's `ShowInTaskbar` property value to `false` prior to calling the `ShowDialog` method.

Then, when the taskbar button is selected, the system will activate the correct window in the ownership group.

More information

The following example demonstrates this resolution in a .NET 6 desktop application using Windows Forms:

C#

```
namespace WinFormsApp1
{
    internal static class Program
    {
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.SetHighDpiMode(HighDpiMode.SystemAware);
            Application.Run(new Form1());
        }
    }

    public class Form1 : Form
    {
        public Form1()
        {
            this.button1 = new System.Windows.Forms.Button();
            this.SuspendLayout();
            this.button1.Location = new System.Drawing.Point(20, 20);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(200, 34);
            this.button1.TabIndex = 0;
            this.button1.Text = "Form2.ShowDialog";
            this.button1.UseVisualStyleBackColor = true;
            this.button1.Click += new
System.EventHandler(this.Button1_Click);
            this.AutoScaleDimensions = new System.Drawing.SizeF(10F, 25F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(800, 450);
            this.Controls.Add(this.button1);
            this.Name = "Form1";
            this.Text = "Form1";
            this.ResumeLayout(false);
        }
```

```
        private void Button1_Click(object? sender, EventArgs e)
        {
            Form2 form = new Form2();
            form.Owner = this;
            form.ShowInTaskbar = false;
            form.ShowDialog();
        }
        private Button button1;
    }

    public class Form2 : Form
    {
        public Form2()
        {
            this.button1 = new System.Windows.Forms.Button();
            this.SuspendLayout();
            this.button1.Location = new System.Drawing.Point(20, 20);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(200, 34);
            this.button1.TabIndex = 0;
            this.button1.Text = "Form3.ShowDialog";
            this.button1.UseVisualStyleBackColor = true;
            this.button1.Click += new
System.EventHandler(this.Button1_Click);
            this.AutoScaleDimensions = new System.Drawing.SizeF(10F, 25F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(800, 450);
            this.Controls.Add(this.button1);
            this.Name = "Form2";
            this.Text = "Form2";
            this.BackColor = System.Drawing.Color.Aqua;
            this.ResumeLayout(false);
        }

        private void Button1_Click(object? sender, EventArgs e)
        {
            Form3 form = new Form3();
            form.Owner = this;
            form.ShowInTaskbar = false;
            form.ShowDialog();
        }
        private Button button1;
    }

    public class Form3 : Form
    {
        public Form3()
        {
            this.SuspendLayout();
            this.AutoScaleDimensions = new System.Drawing.SizeF(10F, 25F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(800, 450);
            this.Name = "Form3";
            this.Text = "Form3";
```

```
            this.BackColor = System.Drawing.Color.Olive;
            this.ResumeLayout(false);
        }
    }
}
```

Feedback

Was this page helpful? Yes No

New transaction can't enlist in the specified transaction coordinator when you try to start a transaction in MS DTC

Article • 12/19/2023

This article helps you resolve a problem when you start a transaction in Microsoft Distributed Transaction Coordinator (MS DTC).

Original product version: Windows
Original KB number: 922430

> **Important**
>
> This article contains information about how to modify the registry. Make sure to back up the registry before you modify it. Make sure that you know how to restore the registry if a problem occurs. For more information about how to back up, restore, and modify the registry, see **Windows registry information for advanced users**.

Symptoms

Consider the following scenario:

- You have a client computer that communicates with a server computer.
- MS DTC is installed on both computers.
- One or more of the following conditions are true:
  - You restart either of the computers.
  - You restart MS DTC on either of the computers.
  - The computers are in different domains.

In this scenario, you receive the following error message when you try to start a transaction in MS DTC:

> New transaction cannot enlist in the specified transaction coordinator (0x8004d00a)

Additionally, the first transaction fails. The subsequent transactions succeed for a while. However, the subsequent transactions may fail again. If the subsequent transactions fail, you receive the following error message:

> New transaction cannot enlist in the specified transaction coordinator (0x8004d00e)

Cause

This problem may occur when the MS DTC connection between the client computer and the server computer is closed. For example, an idle time-out, a remote procedure call (RPC) time-out, or the firewall may close the MS DTC connection between the client computer and the server computer. When a new transaction request occurs, the client computer must reestablish the MS DTC connection with the server computer.

When the client computer tries to reestablish the MS DTC connection with the server computer, the client computer sends a packet. Then, the client computer waits for a bind packet response from the server computer. By default, the client computer stops the transaction if the client computer doesn't receive a response from the server computer in 4 seconds. The response from the server computer may be delayed because of network latency issues or authentication delays. When the response from the server computer does finally reach the client computer, the subsequent transactions succeed.

The first transaction may take a long time, and then a later request to do a distributed transaction may finish immediately. This problem may occur when the client-side of MS DTC has a problem communicating with the Kerberos (KDC) server. Typically, this problem occurs if the client and the server are in different domains that have a firewall between them.

For example, this problem occurs in the following scenario:

- Web Service is in the Perimeter Network in a domain. Web Service has to use transactions with a database server in another domain in an intranet.
- A firewall is between the Perimeter Network and the intranet. The excessive delay on the first transaction occurs because User Datagram Protocol (UDP) port 88 (Kerberos) is blocked.
- The retry and the retry interval for the Kerberos request equal an excessive delay (over 100 seconds).

Resolution

> ⚠ **Warning**
>
> Serious problems might occur if you modify the registry incorrectly by using Registry Editor or by using another method. These problems might require that you

reinstall your operating system. Microsoft cannot guarantee that these problems can be solved. Modify the registry at your own risk.

To make sure that you're experiencing the problem that is described in this article, confirm that the MS DTC transaction trace log file contains the following data:

> ;eventid=TRANSACTION_PROPOGATION_FAILED_CONNECTION_DOWN_FROM_REMOTE_TM ;tx_guid=f11cd9c9-7b8a-41e3-a904-4840123bacf7 ;"failed to propogate transaction to child node ' ComputerName ' because the connection with the remote transaction manager went down"

> **Note**
>
> In this data, the word *propogation* is a misspelling for the word *propagation*. The word *propogate* is a misspelling for the word *propagate*.

If the MS DTC transaction trace log file contains this data, follow these steps:

1. Select **Start**, select **Run**, type *regedit*, and then select **OK**.
2. Locate the following registry subkey:
   `HKEY_LOCAL_MACHINE\Software\Microsoft\MSDTC`
3. Right-click **MSDTC**, point to **New**, and then select **DWORD Value**.
4. Type *CmMaxNumberBindRetries*, and then press ENTER.
5. Right-click **CmMaxNumberBindRetries**, and then select **Modify**.
6. Select **Decimal**.
7. In the **Value data** box, type *60*.

   This value increases the length of time that the client computer waits for the bind packet response from the server computer. This value is double the number of seconds before the client computer stops the transaction if the client computer doesn't receive the bind packet response. For example, a value of 60 equals 30 seconds. The value of 60 is only a recommended value. Additional testing on your configuration may be required.
8. Select **OK**.
9. Restart MS DTC.

> **Note**
>
> For the slow response scenario, make sure that the ports that are required by Kerberos authentication (UDP 88 and Transmission Control Protocol (TCP) 88) are open when a firewall is involved in the Perimeter Network. The ports UDP 389 and TCP 389 (both for Lightweight Directory Access Protocol (LDAP) to find Key Distribution Center (KDC)) must also be open.

Feedback

Enable diagnostic tracing for MS DTC on a Windows 10 computer

Article • 12/19/2023

This article discusses how to enable diagnostic tracing for the Microsoft Distributed Transaction Coordinator (MS DTC) on a Windows 10 computer.

Original product version: Windows 10
Original KB number: 926099

> **Important**
>
> This article contains information about how to modify the registry. Make sure that you back up the registry before you modify it. Make sure that you know how to restore the registry if a problem occurs. For more information about how to back up, restore, and modify the registry, see **Windows registry information for advanced users**.

Types of tracing facilities

- Transaction manager tracing

  Transaction manager (TM) tracing tracks transaction state changes. It's generated by the MS DTC transaction manager. The output is in binary format, and the output must be formatted. The transaction manager is part of the MS DTC service.

- Communication manager error tracing

  Communication manager (CM) error tracing tracks any process that loads the *Msdtcprx.dll* file and that uses the remote procedure call (RPC) interface of MS DTC to communicate with other MS DTC-related processes. The output is in text format. The **0x8004d00a** error is a typical error for which communication manager error tracing may be useful.

> **Note**
>
> Transaction manager tracing and communication manager error tracing are independent processes. You can independently enable transaction manager tracing

and communication manager error tracing. Or, you can independently disable them.

Enable transaction manager tracing

You can use the Component Services Microsoft Management Console (MMC) snap-in to enable transaction manager tracing. To do this, follow these steps:

1. Select **Start**, select **All Programs**, select **Accessories**, and then select **Run**.
2. Type *comexp.msc*, and then select **OK**.
3. Expand **Component Services**, expand **Computers**, expand **My Computer**, expand **Distributed Transaction Manager**, right-click **Local DTC**, and then select **Properties**.
4. Select the **Tracing** tab.
5. On the **Tracing** tab, you can modify the following TM tracing options:

    - **Trace Output**
        - **Trace Transactions**
            - **Trace All Transactions**
            - **Trace Aborted Transactions**
            - **Trace Long-Lived Transactions**

When you change the TM tracing configuration, the MS DTC service detects the changes. However, you don't have to recycle the process. For example, when you change the transactions that are traced or the location of the trace file, the MS DTC service detects the change in the registry. You don't have to restart the MS DTC service.

New MS DTC tracing system in Windows

In Windows, MS DTC has a new and extensive tracing system. The new system has the following design goals:

- Only a single line of code is required to add a formatted trace.
- The trace is readable by humans on production computers without a network.
- Output options are flexible.
- The trace is fast.
- You don't have to restart the computer to change the options.

Additionally, in Windows 10 and Windows Server Technical Preview, the tracing log file name includes the name of the process that invoked the tracing log. This is controlled by the Output registry key.

Configure tracing

> ⚠ **Warning**
>
> Serious problems might occur if you modify the registry incorrectly by using Registry Editor or by using another method. These problems might require that you reinstall the operating system. Microsoft can't guarantee that these problems can be solved. Modify the registry at your own risk.

Trace configuration is located on the local node, in a registry key that is named `Tracing` under the MS DTC registry key. The `Tracing` registry key includes connection manager tracing. Previously, you used the `TraceCMErr` registry key to configure connection manager tracing. The `Tracing` registry key contains the following two sub keys:

- `Sources`: This sub key configures the kind of tracing.
- `Output`: This sub key configures where the tracing output is sent.

> ⓘ **Note**
>
> Make sure that the process can access the `Tracing` registry key. By default, the Windows configuration doesn't create this key and doesn't grant permissions to this key. You can use Registry Editor to configure tracing functionality.

To create tracing functionality registry entries, follow these steps:

1. Select **Start** > **All Programs** > **Accessories** > **Run**, type *regedit*, and then select **OK**.
2. Locate and then select one of the following sub keys:
   - `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\Tracing\Sources`
   - `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\Tracing\Output`
3. On the **Edit** menu, select **New**, and then select the data type of the entry. For example, select **DWORD (32-bit) Value**.
4. Type the name of the desired source, and then press ENTER. For more information about the source names, see the table in the **Configure sources** section.

5. Right-click the new registry entry, select **Modify**, type the value that you want in the **Value data** box, and then select **OK**.

6. On the **File** menu, select **Exit**.

Configure sources

The `Sources` registry key contains a set of DWORD registry values that are listed in the following table.

Expand table

Name	Description
TRACE_MISC	Traces that can't be categorized into the other categories
TRACE_CM	Traces in the connection manager
TRACE_TRACE	The trace infrastructure itself
TRACE_SVC	Traces service and .exe file startup
TRACE_GATEWAY	Gateway source
TRACE_UI	Traces the user interface
TRACE_CONTACT	Traces the contact pool and contacts
TRACE_UTIL	Traces utility routines that are called from multiple locations
TRACE_CLUSTER	Traces the cluster-specific (utility) code
TRACE_RESOURCE	Traces the cluster resource-specific code
TRACE_TIP	Transaction Internet Protocol (TIP) tracing source
TRACE_XA	XA Transaction Manager (XATM) tracing source
TRACE_LOG	Log tracing
TRACE_MTXOCI	MTS/OCI layer (Mtxoci.dll) tracing source
TRACE_ETWTRACE	Event Tracing for Windows (ETW) tracing source
TRACE_PROXY	Traces that are generated in the MSDTC proxy DLL
TRACE_KTMRM	Tracing for integration with Kernel Transaction Manager
TRACE_VSSBACKUP	Tracing for integration with the Microsoft Visual SourceSafe backup and restore mechanism

Name	Description
TRACE_PERFMON	Tracing with support for performance counters

The DWORD value should be a number from 0 to 255. The DWORD value indicates the level of tracing that occurs. The following table lists possible DWORD values.

Expand table

Value	Description
0	const BYTE TRACE_OFF
1	const BYTE TRACE_ERROR
2	const BYTE TRACE_WARNING
3	const BYTE TRACE_INFO
4	const BYTE TRACE_VERBOSE
5	const BYTE TRACE_VERY_VERBOSE
6	const BYTE TRACE_INOUT
0xF0	const BYTE TRACE_OBSCURE
0xFF	const BYTE TRACE_EVERYTHING

> **Note**
>
> Higher values automatically include lower values. Therefore, when you enable the `TRACE_INFO` level, the `TRACE_ERROR` level is also enabled. Very few sources use any tracing that is higher than the `TRACE_VERBOSE` level.

Configure trace output

> **Warning**
>
> Serious problems might occur if you modify the registry incorrectly by using Registry Editor or by using another method. These problems might require that you reinstall the operating system. Microsoft can't guarantee that these problems can be solved. Modify the registry at your own risk.

The `Output` registry key contains a set of values that control where trace output is sent. These values are as follows:

- The `TraceFilePath` (REG_SZ) value is the root folder in which trace files should be stored. Tracing is written to a file in a folder that is named *msdtc-X.log*. In this folder name, *X* represents the decimal PID of the process that creates the file. Make sure that all the processes of interest can access the configured folder. Otherwise, trace information will be lost. If this value isn't set, traces aren't sent to a file.

- The `ImageNameInTraceFileNameEnabled` (REG_DWORD) value determines whether the name of the tracing log file that's generated includes the image file name of the process that invoked the tracing log. If this is set to a non-zero value, the image file name of the process will be included in the tracing log file that's generated. If this value is set to zero, the image file name of the process will not be included in the tracing log file that's generated. By default, the value is set to zero (0). The following is an example of a trace file name where the log file that's generated contains the process:
MSDTC-msdtc.exe-3552.log or *MSDTC-svchost.exe - 3556.log*

- The `MemoryBufferSize` (REG_DWORD) value is the size of the circular buffer in which trace messages are stored. If this value is set to 0, memory tracing is disabled. By default, this value is 10 MB. You may have to increase this value if you enable verbose tracing.

- The `DebugOutEnabled` (REG_DWORD) value enables or disables output to the debugger. If the value is nonzero, the output is enabled. By default, this `Output` registry key is disabled. When you change the connection manager error-tracing configuration, the changes take effect when a process that loads the *Msdtcprx.dll* file is recycled. For example, the changes to the connection manager error-tracing configuration take effect when the MS DTC service process is recycled.

Alternatively, you can create a .reg file, and then you can use Registry Editor to import the file. To do this, follow these steps:

1. Create a new .reg file that contains the following code example:

```
Console

Windows Registry Editor Version 5.00
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\Tracing]
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\Tracing\Output]
"DebugOutEnabled"=dword:00000000
"TraceFilePath"=""
```

```
"MemoryBufferSize"=dword:0000000a
"ImageNameInTraceFileNameEnabled"=dword:00000001
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\Tracing\Sources]
"TRACE_MISC"=dword:00000000
"TRACE_CM"=dword:00000000
"TRACE_TRACE"=dword:00000000
"TRACE_SVC"=dword:00000000
"TRACE_GATEWAY"=dword:00000000
"TRACE_UI"=dword:00000000
"TRACE_CONTACT"=dword:00000000
"TRACE_UTIL"=dword:00000000
"TRACE_CLUSTER"=dword:00000000
"TRACE_RESOURCE"=dword:00000000
"TRACE_TIP"=dword:00000000
"TRACE_XA"=dword:00000000
"TRACE_LOG"=dword:00000000
"TRACE_MTXOCI"=dword:00000000
"TRACE_ETWTRACE"=dword:00000000
"TRACE_PROXY"=dword:00000000
"TRACE_KTMRM"=dword:00000000
"TRACE_VSSBACKUP"=dword:00000000
```

2. Select **Start** > **All Programs** > **Accessories** > **Run**, type *regedit*, and then click **OK**.
3. On the **File** menu, select **Import**.
4. Locate the file that you created in step 1, and then select **Open**. The **Registry Editor** dialog box appears.
5. Select **OK**.
6. On the **File** menu, select **Exit**.

Performance effect

By default, tracing functionality is disabled in Windows. Therefore, no performance effect exists on a regular installation.

Don't enable tracing functionality on production computers unless a Microsoft Customer Support Professional indicates that the tracing information is required to diagnose a problem. Tracing may affect computer performance. First you must find the problem, and you must resolve it. Then, immediately disable the tracing functionality.

Cluster configuration

For cluster installations, make sure that all registry entries on all the nodes contain these registry entries. For any node that doesn't have these registry entries, the cluster code

ignores the registry entries in the shared registry because the registry entries don't exist in the local computer registry.

Feedback

Was this page helpful? Yes No

The Microsoft Distributed Transaction Coordinator service must run under the NT AUTHORITY\NetworkService Windows account

Article • 12/19/2023

This article introduces the Windows account that Microsoft Distributed Transaction Coordinator (MSDTC) must run in Windows.

> **Important**
>
> This article contains information about how to modify the registry. Make sure to back up the registry before you modify it. Make sure that you know how to restore the registry if a problem occurs. For more information about how to back up, restore, and modify the registry, see: **Windows registry information for advanced users**.

Original product version: Windows Server 2012, Windows 8, Windows 7
Original KB number: 903944

Summary

On all Windows Client and Server Operating Systems, you may have to restart the MSDTC service to perform these steps. To restart the MSDTC service, follow these steps:

1. For Windows 8.1 and Windows 8

   - From the Start screen, swipe in from the right side to display the charms, select **Search**, and then search for *cmd*. (Or, if you are using a keyboard and mouse, type *cmd* at the Start screen.) In the search results, press-and-hold or right-click Command Prompt, and then select **Run as Administrator**.

   For Windows 7 and earlier versions

   - Press the Windows logo **key+R**, type *cmd* in the Run box, and then press Enter. Right-click **cmd**, and then select **Run as Administrator**.

2. Type `net stop msdtc`, and then press the **ENTER** button.

3. Type `net start msdtc` , and then press the **ENTER** button.
4. Open the Component Services Microsoft Management Console (MMC) snap-in. To do this, click **Start**, and then click **Run** Type *dcomcnfg.exe*, and then click **OK**.
5. Expand **Component Services**, expand **Computers**, and then expand **My Computer**.
6. Right-click **My Computer**, and then click **Properties**.
7. Click the **MSDTC** tab, and then click **Security Configuration**.
8. Change the account in **DCT Logon Account** to **NT AUTHORITY\NetworkService**. If a password is needed, enter a blank password.
9. Click **OK** two times.

For Windows XP and Windows Server 2003

Starting in Windows XP and then continuing in Windows Server 2003, the MSDTC service must run under the `NT AUTHORITY\NetworkService` Windows account.

If you change the account to an account other than the NetworkService account, the distributed transaction fails. The transaction fails because the MSDTC service cannot do mutual authentication together with other parties that are involved in the transaction. Local transactions that use the MSDTC service may also fail.

> ⓘ **Note**
>
> Other parties can be transaction managers, resource manager, or clients.

In both Microsoft Windows NT 4.0 and Microsoft Windows 2000, you can change the default MSDTC service account to a domain account. You may change the account to perform Windows authentication when you are performing an XA recovery operation on an XA database such as an Oracle database.

However, in Windows Server 2003 and Windows XP, you cannot change the account. Instead, you must give the permissions and the roles that are required to perform an XA recovery operation to the NetworkService account on the computer where the MSDTC service is running.

The exact method of setting up an XA recovery operation is specific to each XA database. Typically, you have to add the computer account of the computer where the MSDTC service is running to the list of users who can perform an XA recovery operation

on the XA database. Additionally, because the NetworkService account is a restricted account, you must provide the NetworkService account access to the folder where the XA DLL is located.

To change the account that the MSDTC service runs under back to the NetworkService account, follow these steps.

> ⚠ **Warning**
>
> Serious problems might occur if you modify the registry incorrectly by using Registry Editor or by using another method. These problems might require that you reinstall your operating system. Microsoft cannot guarantee that these problems can be solved. Modify the registry at your own risk.

1. Click **Start**, click **Run**, type *regedit*, and then click **OK**.
2. Locate and then click the following subkey:
   `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC`.

   If the following entries exist, go to step 6:

   - `TurnOffRpcSecurity`
   - `AllowOnlySecureRpcCalls`
   - `FallbackToUnsecureRPCIfNecessary`

3. Create the `TurnOffRpcSecurity` entry:
   a. On the **Edit** menu, point to **New**, and then click **DWORD Value**.
   b. Type *TurnOffRpcSecurity*, and then press ENTER.
4. Create the `AllowOnlySecureRpcCalls` entry:
   a. On the **Edit** menu, point to **New**, and then click **DWORD Value**.
   b. Type *AllowOnlySecureRpcCalls*, and then press ENTER.
5. Create the `FallbackToUnsecureRPCIfNecessary` entry:
   a. On the **Edit** menu, point to **New**, and then click **DWORD Value**.
   b. Type *FallbackToUnsecureRPCIfNecessary*, and then press ENTER.
6. Set the DWORD value for the `TurnOffRpcSecurity` entry:
   a. Right-click **TurnOffRpcSecurity**, and then click **Modify**.
   b. In the **Edit DWORD Value** dialog box, type the value *1*, and then click **OK**.
7. Set the DWORD value for the `AllowOnlySecureRpcCalls` entry:
   a. Right-click **AllowOnlySecureRpcCalls**, and then click **Modify**.

b. In the **Edit DWORD Value** dialog box, type the value *0*, and then click **OK**.

8. Set the DWORD value for the `FallbackToUnsecureRPCIfNecessary` entry:
   a. Right-click **FallbackToUnsecureRPCIfNecessary**, and then click **Modify**.
   b. In the **Edit DWORD Value** dialog box, type the value *0*, and then click **OK**.

After you have made the registry changes, you must restart the MSDTC service. To restart the MSDTC service, follow these steps:

1. Click **Start**, click **Run**, type *cmd*, and then click **OK**.
2. Type `net stop msdtc` , and then press ENTER.
3. Type `net start msdtc` , and then press ENTER.
4. Open the Component Services Microsoft Management Console (MMC) snap-in. To do this, click **Start**, click **Run**, type *dcomcnfg.exe*, and then click **OK**.
5. Expand **Component Services**, expand **Computers**, and then expand **My Computer**.
6. Right-click **My Computer**, and then click **Properties**.
7. Click the **MSDTC** tab, and then click **Security Configuration**.
8. Change the account in **DCT Logon Account** to **NT AUTHORITY\NetworkService**. If a password is needed, enter a blank password.
9. Click **OK** two times.

References

- New functionality in the Distributed Transaction Coordinator service in Windows
- Managing Accounts and Privileges

Third-party information disclaimer

The third-party products that this article discusses are manufactured by companies that are independent of Microsoft. Microsoft makes no warranty, implied or otherwise, about the performance or reliability of these products.

Applies to

- Windows Server 2012 R2 Datacenter
- Windows Server 2012 R2 Standard
- Windows Server 2012 R2 Essentials
- Windows 8.1 Enterprise
- Windows 8.1 Pro
- Windows 8.1
- Windows Server 2012 Datacenter

- Windows Server 2012 Datacenter
- Windows Server 2012 Standard
- Windows Server 2012 Standard
- Windows Server 2012 Essentials
- Windows 8 Enterprise
- Windows 8 Pro
- Windows 8
- Windows Server 2008 R2 Datacenter
- Windows Server 2008 R2 Standard
- Windows Server 2008 R2 Enterprise
- Windows 7 Enterprise
- Windows 7 Professional
- Windows Server 2008 Datacenter
- Windows Server 2008 Standard
- Windows Server 2008 Enterprise
- Windows Vista Enterprise
- Windows Vista Business
- Microsoft Windows Server 2003 Enterprise Edition (32-bit x86)
- Microsoft Windows Server 2003 Standard Edition (32-bit x86)
- Microsoft Windows Server 2003 Datacenter Edition (32-bit x86)
- Microsoft Windows Server 2003 Web Edition
- Microsoft Windows Server 2003 Standard x64 Edition
- Microsoft Windows Server 2003 Enterprise x64 Edition
- Microsoft Windows Server 2003 Datacenter x64 Edition
- Microsoft Windows XP Professional
- Microsoft Windows XP Professional x64 Edition

Feedback

New functionality in the Distributed Transaction Coordinator service in Windows

Article • 12/19/2023

This article describes some Windows security-related updates and changes, and how they affect the Microsoft Distributed Transaction Coordinator (MS DTC) service.

Original product version: Windows
Original KB number: 899191

Summary

Since Windows Server 2003 Service Pack 1 (SP1) and Windows XP Service Pack 2 (SP2), some security-related updates and changes were introduced to Windows and affect the MS DTC service.

These changes can be accessed by using the updated **Security Configuration** dialog box that is available in the Component Services administrative tool.

These changes are made to the default security settings that cause DTC traffic to fail over the network. In this situation, you may receive one or more error messages or error codes.

By modifying the settings in the **Security Configuration** dialog box, you can help control how the DTC service communicates with remote computers over the network.

This article describes new functionality in the MS DTC service in the following operating systems:

- Windows Server 2003 Service Pack 1 (SP1)
- Windows XP Service Pack 2 (SP2)
- Windows Vista
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2

The DTC service coordinates transactions that update two or more transaction-protected resources. Transaction-protected resources include databases, message queues, and file systems. These transaction-protected resources may be located on a single computer or may be distributed between many networked computers.

Manage network communication by using Security Configuration

In Windows, the DTC service gives you more control over the network communication between computers. By default, all network communication is disabled. The DTC **Security Configuration** dialog box has been enhanced so that you can manage these communication settings. To view the **Security Configuration** dialog box, follow these steps:

1. Start the Component Services administrative tool. To do this, select **Start**, select **Run**, type **dcomcnfg.exe**, and then select **OK**.
2. In the console tree of the Component Services administrative tool, expand **Component Services**, expand **Computers**, right-click **My Computer**, and then select **Properties**.
3. Select the **MSDTC** tab, and then select **Security Configuration**.

New options in Security Configuration

The following information describes the new options that are available in the **Security Configuration** dialog box. This information also describes the registry entries that are affected by the new options in the **Security Configuration** dialog box.

Network DTC Access

The **Network DTC Access** check box lets you determine whether the DTC service can access the network. The **Network DTC Access** check box must be selected together with one of the other check boxes under the **Network DTC Access** check box to enable network DTC transactions.

The **Network DTC Access** check box affects the following registry entry:

- Registry path: `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\Security`
- Value name: `NetworkDtcAccess`
- Value type: REG_DWORD
- Value data: 0 (default)

> ⓘ **Note**
>
> On a server cluster, the **Network DTC Access** check box affects a value in the shared cluster registry key under the MS DTC resource registry key. The registry key of the shared cluster for MS DTC is located at the `HKEY_LOCAL_MACHINE\Cluster\Resources\<MSDTC resource GUID>` location.

By default, the value of the `NetworkDtcAccess` registry entry is set to 0. A value of 0 turns off the `NetworkDtcAccess` registry entry. To turn on the `NetworkDtcAccess` registry entry, set this registry value to 1.

Allow Inbound

The **Allow Inbound** check box lets you determine whether to allow a distributed transaction that originates from a remote computer to run on the local computer. By default, this setting is turned off. To enable this setting, click to select the **Network DTC Access** check box to set the following registry entry to 1:

- Registry path: `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\Security`
- Value name: `NetworkDtcAccess`
- Value type: REG_DWORD

To disable this setting, click to clear the **Network DTC Access** check box to set this registry entry to 0.

The **Allow Inbound** check box affects both of the following REG_DWORD registry entries under `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\Security`:

- `NetworkDtcAccessTransactions`

- `NetworkDtcAccessInbound`

Allow Outbound

The **Allow Outbound** check box lets you determine whether to allow the local computer to initiate a transaction and run that transaction on a remote computer. To enable this setting, select the **Network DTC Access** check box to set the following registry entry to 1:

- Registry path: `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\Security`
- Value name: `NetworkDtcAccess`
- Value type: REG_DWORD

To disable this setting, clear the **Network DTC Access** check box to set this registry entry to 0.

The **Allow Outbound** check box affects both of the following REG_DWORD registry entries under `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\Security`:

- `NetworkDtcAccessTransactions`
- `NetworkDtcAccessOutbound`

Mutual Authentication Required

The **Mutual Authentication Required** option adds support for mutual authentication in Windows. **Mutual Authentication Required** sets the greatest security mode that is currently available for network communication. We recommend this transaction mode for client computers that are running Windows XP SP2 together with server computers that are running Windows Server 2003 SP1.

Mutual Authentication Required affects the following registry entries under `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC`:

- Registry key 1
  - Value name: `AllowOnlySecureRpcCalls`
  - Value type: REG_DWORD
  - Value data: 1
- Registry key 2
  - Value name: `FallbackToUnsecureRPCIfNecessary`
  - Value type: REG_DWORD
  - Value data: 0
- Registry key 3
  - Value name: `TurnOffRpcSecurity`
  - Value type: REG_DWORD
  - Value data: 0

The functionality that is set by using **Mutual Authentication Required** differs from the functionality that is set by using **Incoming Caller Authentication Required**. The three options that are listed under **Transaction Manager Communication** behave as follows:

- The **Mutual Authentication Required** transaction mode requires the remotely accessing component to provide an authenticated connection with the local computer. This authentication is verified by impersonation on the local computer. Additionally, if the remote access communication is performed

between two DTC services, this authentication information must specify a computer account that matches the remote transaction mode computer's host name.

- The **Incoming Caller Authentication Required** transaction mode only requires the remote connection to be authenticated. Additionally, if the remotely accessing component is a DTC service, the authentication information must be for a computer account.

- The **No Authentication Required** transaction mode does not validate an authenticated connection or verify whether an authenticated connection is being established.

In a clustered environment, the computer account for the DTC service specifies the cluster node's host name. In a clustered environment, the DTC authentication does not use the transaction mode's host name. In a clustered environment, the transaction mode's host name is the name of the virtual service. Therefore, you cannot use the **Mutual Authentication Required** transaction mode in a clustered environment, or on any computers that are negotiating transactions with such computers. You can use the **Mutual Authentication Required** transaction mode between two nonclustered computers that are running Windows Server 2003 SP1 or between two computers that are running Windows XP SP2.

Use the **Incoming Caller Authentication Required** transaction mode between Windows Server 2003-based computers in a clustered environment.

Use the **No Authentication Required** transaction mode where one or more of the following conditions are true:

- The network access is between computers that are running Microsoft Windows 2000.
- The network access is between two domains that do not have a mutual trust configured.
- The network access is between computers that are members of a work group.

Incoming Caller Authentication Required

Incoming Caller Authentication Required requires the local DTC service to communicate with a remote DTC service by using only encrypted messages. Only the incoming connection will be authenticated. Only Windows Server 2003 SP1, Windows XP SP2, and later version of Windows support this feature. Therefore, only enable this option if the remote DTC service is running on a computer that is running Windows Server 2003 SP1, Windows XP SP2, or later version of Windows.

Incoming Caller Authentication Required affects the following registry entries under `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC`:

- Registry key 1
  - Value name: AllowOnlySecureRpcCalls
  - Value type: REG_DWORD
  - Value data: 0

- Registry key 2
  - Value name: FallbackToUnsecureRPCIfNecessary
  - Value type: REG_DWORD
  - Value data: 1

- Registry key 3
  - Value name: TurnOffRpcSecurity
  - Value type: REG_DWORD
  - Value data: 0

For more information about **Incoming Caller Authentication Required**, see the **Mutual Authentication Required** section.

No Authentication Required

No Authentication Required enables operating system compatibility between earlier versions of the Windows operating system. When this option is enabled, network communication between DTC services can fall back to non-authenticated communication or to non-encrypted communication if a secure communication channel cannot be established.

> ⓘ **Note**
>
> We recommend that you use this setting if the remote DTC service is running on a computer that is running Microsoft Windows 2000 or on a computer that is running a version of Windows XP that is earlier than Windows XP SP2.

You can also use **No Authentication Required** to resolve a situation where the DTC services are running on computers that are in domains that do not have a trust relationship established. Additionally, you can use **No Authentication Required** to resolve a situation where the DTC services are running on computers that are members of a work group.

No Authentication Required affects the following registry entries under `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC`:

- Registry key 1
  - Value name: AllowOnlySecureRpcCalls
  - Value type: REG_DWORD
  - Value data: 0
- Registry key 2
  - Value name: FallbackToUnsecureRPCIfNecessary
  - Value type: REG_DWORD
  - Value data: 0
- Registry key 3
  - Value name: TurnOffRpcSecurity
  - Value type: REG_DWORD
  - Value data: 1

> ⓘ **Note**
>
> On a server cluster, these registry entries are located in the shared cluster registry.

Significance of the new options

The new options that are available in the **Security Configuration** dialog box let you apply security settings to outgoing or incoming network communications. By default, after you install Windows, the computer does not accept network traffic. Therefore, the computer is less vulnerable to network access by a malicious user. Additionally, the protocols that are sent over the network are updated to support a more securely encrypted

and mutually authenticated communications mode. This helps reduce the chance that a malicious user could intercept and take over communications between DTC services.

Network communication changes in Windows

The network communication coming out of the DTC service or coming in to the DTC service is disabled. For example, if a COM+ object tries to update a Microsoft SQL Server database that is located on a remote computer by using a DTC transaction, this transaction does not succeed. Conversely, if the computer hosts a SQL Server database that components from a remote computer try to access by using a DTC transaction, this transaction does not succeed.

The following issues are related to the DTC service:

Transactions fail because of network connectivity issues

> **Important**
>
> This section, method, or task contains steps that tell you how to modify the registry. However, serious problems might occur if you modify the registry incorrectly. Therefore, make sure that you follow these steps carefully. For added protection, back up the registry before you modify it. Then, you can restore the registry if a problem occurs. For more information about how to back up and restore the registry, see **How to back up and restore the registry in Windows**.

If the DTC transactions fail because of network connectivity issues, click to select the following check boxes in the **Security Configuration** dialog box:

- Click to select the **Network DTC Access** check box.
- Click to select one or both of the following check boxes under **Transaction Manager Communication** depending on your requirements:
  - **Allow Inbound**
  - **Allow Outbound**

If you want to programmatically change these settings as part of a Windows, you can directly modify the registry settings that correspond to the settings that you want to set. After you modify the registry settings, you must restart the DTC service.

We recommend that you do not manually modify the registry to change these settings. If you manually modify these registry settings, you may experience issues with the Cluster service on Windows Server 2003 SP1-based server clusters.

Windows Firewall blocks DTC traffic

> **Important**
>
> These steps may increase your security risk. These steps may also make the computer or the network more vulnerable to attack by malicious users or by malicious software such as viruses. We recommend the process that this article describes to enable programs to operate as they are designed to or to implement specific program capabilities. Before you make these changes, we recommend that you

evaluate the risks that are associated with implementing this process in your particular environment. If you decide to implement this process, take any appropriate additional steps to help protect the system. We recommend that you use this process only if you really require this process.

If you use Windows Firewall, you must add the DTC service to the exception list in the Windows Firewall settings. To do this, follow these steps:

1. Select **Start**, select **Run**, type **firewall.cpl**, and then select **OK**.
2. In the **Windows Firewall** dialog box, select the **Exceptions** tab, and then select **Add Program**.
3. Select **Browse**, locate and then select `C:\Windows\System32\msdtc.exe`, and then select **Open**.
4. Select **OK**, select the **msdtc.exe** check box in the **Programs and Services** list if this check box is not already selected, and then select **OK**.

Settings that are changed or added

The following table describes the registry entries that are changed since Windows XP SP2 from earlier versions of Windows.

Expand table

Entry name	Location	Previous default value	Windows XP SP2 default value	Possible values
NetworkDtcAccess	`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\Security`	1	0	0 or 1
NetworkDtcAccessTransactions	`KEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\Security`	1	0	0 or 1
NetworkDtcAccessInbound	`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\Security`	Not applicable	0	0 or 1
NetworkDtcAccessOutbound	`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\Security`	Not applicable	0	0 or 1
AllowOnlySecureRpcCalls	`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC`	Not applicable	1	0 or 1
FallbackToUnsecureRPCIfNecessary	`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC`	Not applicable	0	0 or 1
TurnOffRpcSecurity	`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC`	Not applicable	0	0 or 1

Note

These changes appear in the shared cluster registry on a Windows Server 2003 SP1-based server cluster.

Error codes that are associated with the DTC service changes

You may receive one of the following error codes when you run DTC transactions between computers:

- Error code 1

  > MessageId: XACT_E_NETWORK_TX_DISABLED
  > MessageText:
  > The transaction manager has disabled its support for remote/network transactions.
  > #define XACT_E_NETWORK_TX_DISABLED *HRESULT_TYPEDEF*(0x8004D024L)

- Error code 2

  > MessageId: XACT_E_PARTNER_NETWORK_TX_DISABLED
  > MessageText:
  > The partner transaction manager has disabled its support for remote/network transactions.
  > #define XACT_E_PARTNER_NETWORK_TX_DISABLED *HRESULT_TYPEDEF*(0x8004D025L)

Applies to

- Windows Server 2012 R2
- Windows 8.1
- Windows 8
- Windows 7
- Windows Vista
- Windows Server 2008
- Windows Server 2003
- Windows XP

Feedback

Registry entries are required for XA transaction support

Article • 12/19/2023

Starting with Windows Server 2003, Microsoft Distributed Transaction Coordinator (MS DTC) requires that you create registry values for all XA DLLs that you plan to use. This article provides steps to modify the registry.

Original product version: Windows Server 2003
Original KB number: 817066

Summary

Starting with Windows Server 2003, MS DTC requires that you create registry values for all XA DLLs that you plan to use. This requirement was added to Windows Server 2003 to help you to minimize the risks that are associated with using third-party XA DLLs in the MS DTC process. To retain the same functionality when you use XA transactions, you must add a registry value in the XA DLL key for each XA DLL that you plan to use. This article describes these registry values.

For example, when you upgrade an existing system to Windows Server 2003, and the existing system uses MS DTC with third-party XA DLLs, support for XA transactions is disabled until you create these required registry values. Also, if you later install a third-party product that provides XA DLLs to support XA transactions, you must do one of the followings:

- Create these registry values manually
- Verify that the third-party installer creates these registry values

Turn on support for XA transactions

> **Important**
>
> This section, method, or task contains steps that tell you how to modify the registry. However, serious problems might occur if you modify the registry incorrectly. Therefore, make sure that you follow these steps carefully. For added protection, back up the registry before you modify it. Then, you can restore the registry if a

problem occurs. For more information about how to back up and restore the registry, see **How to back up and restore the registry in Windows**.

A security risk occurs when MS DTC uses user-specified DLLs. These DLLs are loaded directly in the MS DTC process. MS DTC uses these DLLs to communicate with the Transaction Manager (TM) of the XA partner. This scenario can expose the Resource Manager (RM) databases to serious data corruption. This scenario can also permit denial-of-service attacks if a malicious or defective XA DLL does not verify that the distributed transaction commits or aborts correctly. Also, if a malicious or defective XA DLL contains code that is not security-enhanced, an attacker might exploit this weakness to cause a denial-of-service attack.

To help to prevent this security risk, Windows Server 2003 turns off all XA transactions when you upgrade to Windows Server 2003. If the support for XA transactions is turned off, Windows Server 2003 helps to protect MS DTC from denial-of-service attacks.

You may have to turn on support for XA transactions. To do this, follow these steps:

1. Open **Component Services**.
2. Expand the tree view to locate the computer on which you want to turn on support for XA transactions (for example, **My Computer**).
3. Right-click the computer name, and then click **Properties**.
4. Click the **MSDTC** tab, and then click **Security Configuration**.
5. Under **Security Settings**, select the check box for **XA Transactions** to turn on this support.

Windows Server 2003 provides a registry entry for you to specify the XA DLLs that you will use. When you upgrade to Windows Server 2003, you can work with XA transactions in the same way that you worked with them in earlier versions of Microsoft Windows Server.

To do this, create a registry named-value under the following registry subkey:
`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\XADLL`

In your registry named-value, **Name** can be the file name of the XA DLL (for example, dllname.dll), although you are not required to use this naming convention. Also in this named-value, **Type** is String (REG_SZ), and the value is the full path name (including the file name) of the DLL file.

Create an entry for each XA DLL file that you plan to use. Also, if you are configuring MS DTC on a cluster, you must create these registry entries on each node in the cluster.

References

- Managing XA Transactions
- Disabling TIP, LU and XA Transactions
- DTC Security Considerations
- What's New in COM+ 1.5

Feedback

Converting PDF to bitmap causes partial data loss in the image

Article • 12/19/2023

This article helps you resolve a problem that causes a partial loss of image data when you convert a PDF file to a bitmap image by using Windows.Data.Pdf.

Symptoms

Parts of a PDF document are missing after you convert the PDF file to a bitmap image by using the classes of the Windows.Data.Pdf namespace. For example, if a PDF file contains a dashed line (also known as a line dash pattern), part of the table that contains the dashed line will be missing from the bitmap image.

Cause

The Windows.Data.Pdf namespace is implemented by using features of Microsoft Edge Legacy. This problem is known to affect some PDF files, depending on the implementation of Microsoft Edge Legacy.

Workaround

If this problem occurs, you might be able to fix it by regenerating the PDF file to eliminate any elements that seem to cause the data loss. For the line dash pattern example given in the "Symptoms" section above, remove the dashed lines in the source file before you create the PDF file, and then try again to convert the PDF file to a bitmap image. If your application can be modified, consider using the same features as the new Chromium-based Microsoft Edge. The new Microsoft Edge is implemented by using the open source PDFium library.

Feedback

The Save As dialog box is displayed behind the application that's printing to the XPS Document Writer

Article • 12/19/2023

This article helps you resolve a problem where the **Save As** dialog box is hidden behind the app that's printing to an XPS Document Writer printer and the application stops responding.

Original product version: XPS Document Writer
Original KB number: 2567869

Symptoms

Consider the following scenario:

- You run a 32-bit application on a 64-bit version of Windows 7.
- You print from the application to a Microsoft XPS Document Writer (MXDW) printer. In this scenario, the **Save As** dialog box is displayed behind the application.

Additionally, you may experience the following symptoms:

- The application seems to stop responding (hang) until you enter a file name or cancel the printing task.
- The application that's printing doesn't become the foreground (active) application when the **Save As** dialog box is closed.

> **Note**
>
> This problem may also occur when you print to a different printer whose driver displays the **Save As** dialog box or another modal dialog box. The printer driver for the Adobe PDF printer is this type of driver.

Cause

Printer drivers are implemented as dynamic-link libraries (DLLs) that are loaded into a process that's printing. Printer drivers are implemented as 64-bit DLLs on 64-bit versions of Windows and as 32-bit DLLs on 32-bit versions of Windows.

A 32-bit process can't load 64-bit DLLs. Therefore, 64-bit versions of Windows support printing from 32-bit processes through the Splwow64.exe process. Splwow64.exe is a 64-bit process that can load 64-bit printer drivers and that handles printing for 32-bit processes.

When an application calls the `StartDoc` function to print to the XPS Document Writer printer, the XPS Document Writer printer driver displays a **Save As** dialog box so that users can specify the name and location of the XPS file. The owner window of the dialog box is typically the active window of the thread that is calling the `StartDoc` function, and the dialog box will appear over the active window.

When a 32-bit application calls the `StartDoc` function on a 64-bit version of Windows, the Splwow64.exe process calls in to the XPS Document Writer printer driver for the 32-bit application. In this scenario, the **Save As** dialog box is unowned because the thread in the Splwow64.exe process doesn't have an active window. The dialog box may appear behind the application that is printing because the Splwow64.exe process doesn't have permission to set the foreground window. Also, since the dialog is unowned, the application that called the `StartDoc` function may not become the foreground application when the dialog is closed.

The `StartDoc` call doesn't return until the dialog box is dismissed, so the application may seem to stop responding.

The **Save As** dialog box has its own button in the Windows Explorer taskbar if it's created by the Splwow64.exe process. This is because the dialog box is unowned. The taskbar button also flashes when the Splwow64.exe process can't set the foreground window.

Workaround

To work around this issue, you can access the **Save As** dialog box through its taskbar button. Or, you can press Alt+Tab to switch the focus to the dialog box.

More information

Software developers can avoid this problem in their 32-bit applications by having these applications detect when the user is printing to the XPS Document Writer printer or to the Adobe PDF printer. The application then specifies the full path to a file in the `DOCINFO.lpszOutput` structure member when calling the `StartDoc` function. The printer driver will use the specified file instead of prompting the user for a file.

Third-party information disclaimer

The third-party products that this article discusses are manufactured by companies that are independent of Microsoft. Microsoft makes no warranty, implied or otherwise, about the performance or reliability of these products.

Feedback

Was this page helpful? Yes No

GDI APIs fail when large pages or VAD spanning are used

Article • 12/19/2023

This article introduces that GDI APIs fails when large pages or Virtual Address Descriptors (VAD) spanning are used.

Original product version: Windows
Original KB number: 4567569

When large page sizes are configured for virtual memory allocations and/or when buffers spanning multiple VAD are passed to the GDI APIs listed below, the function calls fail.

> **Note**
>
> VAD-spanning can occur if memory management facilities such as heap managers and/or garbage collectors request contiguous virtual address ranges(each of which has its own VAD), but attempts to use them as a single block of memory.

- SetDIBitsToDevice function
- StretchDIBits function
- GetBitmapBits function
- CreateDIBitmap function
- CreateDIBSection function
- PolyDraw function
- DrawEscape function
- CreateBitmap function
- SetBitmapBits function
- GetDIBits function

Feedback

Was this page helpful? Yes No

GetICMProfile function leaks one or more handles in Windows 10

Article • 12/19/2023

This article describes a problem in which the `GetICMProfile` function leaks one or more handles in Windows 10.

Applies to: Windows 10, version 2004, Windows 10, version 20H2, Windows 10, version 21H1, Windows 10, version 21H2

Symptoms

Applications that call the GetICMProfile function experience increased handle usage counts in tools such as Task Manager, or experience unexpected errors or behavior.

Cause

The `GetICMProfile` function uses the Windows registry functions to obtain color profile information from the registry. However, it doesn't close the opened registry handles. This problem can also occur when applications call other Image Color Management (ICM) or Windows Color System (WCS) functions. These include the following functions:

- EnumICMProfiles
- WcsGetDefaultColorProfile
- WcsGetDefaultColorProfileSize
- WcsGetUsePerUserProfiles

Status

Microsoft has confirmed that this is a problem in the products that are listed in the "Applies to" section.

This problem is fixed in Windows 11.

Feedback

Graphics::DrawImage with ImageAttributes is slow when drawing a PNG or JPEG image to a printer

Article • 12/19/2023

This article provides a resolution for an issue where the `Graphics::DrawImage` method appears slow when you call it with an `ImageAttributes` object to draw a JPEG or PNG image to a printer.

Cause

The `Graphics::DrawImage` method converts the source image to a device-independent bitmap (DIB) when you call it with an `ImageAttributes` object.

Resolution

To solve the issue, call `Graphics::DrawImage` with the `ImageAttributes` parameter set to `NULL` unless specifying an `ImageAttributes` object is necessary.

When supported by the printer, calling `Graphics::DrawImage` with the `ImageAttributes` parameter set to `NULL` will pass the source JPEG or PNG directly to the printer.

More information

- Graphics::DrawImage(Image*,constPointF*,INT,REAL,REAL,REAL,REAL,Unit,constImageAttributes*,DrawImageAbort,VOID*) method (gdiplusgraphics.h)
- Graphics::DrawImage(Image*,constPoint*,INT,INT,INT,INT,INT,Unit,constImageAttributes*,DrawImageAbort,VOID*) method (gdiplusgraphics.h)
- Graphics::DrawImage(Image*,constRect&,INT,INT,INT,INT,Unit,constImageAttributes*,DrawImageAbort,VOID*) method (gdiplusgraphics.h)
- Graphics::DrawImage(Image*,constRectF&,REAL,REAL,REAL,REAL,Unit,constImageAttributes*,DrawImageAbort,VOID*) method (gdiplusgraphics.h)
- ExtEscape function (wingdi.h)
- Testing a Printer for JPEG or PNG Support

Feedback

Was this page helpful? Yes No

Interoperability between GDI and GDI+

Article • 12/19/2023

It's sometimes desirable to mix Windows Graphics Device Interface (GDI) and GDI+ drawing operations in the same code path. This article provides tips to help you write code that allows GDI and GDI+ to work together.

Original product version: Windows Graphics Device Interface
Original KB number: 311221

Summary

Certain rules apply when mixing GDI and GDI+ code. For example, you shouldn't interleave GDI and GDI+ calls on one target object. It's okay to wrap a Graphics object around an HDC, but you shouldn't access the HDC directly from GDI until the Graphics object is destroyed.

Four primary scenarios for interoperability between GDI and GDI+ are covered in this article:

- Using GDI on a GDI+ Graphics object backed by the screen
- Using GDI on a GDI+ Graphics object backed by a bitmap
- Using GDI+ on a GDI HDC
- Using GDI+ on a GDI memory `HBITMAP` (a handle to a bitmap object)

Using GDI on a GDI+ Graphics object backed by the screen

One example of a need to use GDI on a GDI+ Graphics object backed by the screen would be to draw a rubber band or focus rectangle. GDI+ currently has no direct support for raster operations (ROPs), so GDI must be used directly if R2_XOR pen operations are required. In this case, you would use `Graphics::GetHDC()` to obtain an HDC to which the GDI output would be directed. GDI+ output shouldn't be attempted on the Graphics object for the life of the HDC (that is, until `Graphics::ReleaseHDC()` is called).

Using GDI on a GDI+ Graphics object backed by a bitmap

When `Graphics::GetHDC()` is called for a Graphics object that is backed by a bitmap rather than the screen, a memory HDC is created and a new `HBITMAP` is created and selected into the memory HDC. This new memory bitmap isn't initialized with the original bitmap's image but rather with a sentinel pattern, which allows GDI+ to track changes to the bitmap. Any changes that are made to the memory bitmap through the use of GDI code become apparent in changes to the sentinel pattern. When `Graphics::ReleaseHDC()` is called, those changes are copied back to the original bitmap. Because the memory bitmap isn't initialized with the bitmap's image, an HDC that is obtained in this way should be considered **write only** and is therefore not suitable for use with ROPs, the use of which requires the ability to read the target, like R2_XOR. Also, there's a performance cost to this approach because GDI+ must copy the changes back to the original bitmap.

Using GDI+ on a GDI HDC

You can facilitate the use of GDI+ on an HDC by using the Graphics constructor that takes an HDC as a parameter. The drawing members of the Graphics class can be used to draw on the HDC in this way. Once the Graphics object is attached to the HDC, no GDI operations should be performed on the HDC until the Graphics object is destroyed or goes out of scope. If GDI output is required on the HDC, either destroy the Graphics object before using the original HDC or use `Graphics::GetHDC()` to get a new HDC and then follow the rules described earlier in this article for interoperability while using GDI on a GDI+ object.

Using GDI+ on a GDI memory HBITMAP

The GDI+ Bitmap constructor that takes an `HBITMAP` as a parameter doesn't use the actual source `HBITMAP` as the backing image for the bitmap. Rather, a copy of the image is made in the constructor, and changes aren't written back to the original bitmap, even during execution of the destructor. The new bitmap can be thought of as copy on creation, so to get GDI+ to draw on a memory `HBITMAP` from GDI and have the changes apply to the `HBITMAP`, an approach like the following is needed instead:

1. Create a `DIBSection`.
2. Select the `DIBSection` into a memory HDC.
3. To use GDI+ to draw to the `DIBSection`, wrap a Graphics object around the HDC.
4. To use GDI to draw to/from the `DIBSection`, destroy the Graphics object, and use the HDC.

5. Destroy the Graphics objects, and then clear the `DIBSection` selection from the HDC. Later, a bitmap can be constructed from the `DIBSection` and used as a source image in `Graphics::DrawImage()` if needed.

Feedback

Memory leak occurs when printing documents in Windows 10 and Windows 11

Article • 03/29/2024

Applies to: Windows 10, Windows 11

Symptoms

When an application uses the OpenPrinter, StartDocPrinter, EndDocPrinter, and ClosePrinter functions to print documents, a small memory leak might occur. In addition, you might see increased memory usage in applications that print many documents.

Cause

`Print.PrintSupport.Source.dll` is a system printing component that might leak approximately 300 bytes for each printed document.

Workaround

To mitigate this issue, use one of the following methods:

- As a customer or IT audience, you can periodically restart long-running processes that print many documents.
- As an application developer, you can design the application to print each document in a separate process.

Feedback

Modify printer settings with the DocumentProperties() Function

Article • 12/19/2023

This article shows how to modify printer settings with the `DocumentProperties()` function.

Original product version: Printer
Original KB number: 167345

Summary

Using a DEVMODE structure to modify printer settings is more difficult than just changing the fields of the structure. Specifically, a valid DEVMODE structure for a device contains private data that can only be modified by the `DocumentProperties()` function.

This article explains how to modify the contents of a DEVMODE structure with the `DocumentProperties()` function.

More information

A DEVMODE structure, as documented by the Win32 SDK, contains public or 'device independent data' and private or 'device dependent data'. The private part of a DEVMODE exists immediately following the public part, which is defined by the DEVMODE structure, in a contiguous buffer of memory.

A program cannot predict the size of this buffer because it is different from printer to printer and from version to version of printer driver. Additionally, a DEVMODE structure that is declared by a program does not contain enough space for private device data. If a DEVMODE buffer that lacks private data is passed to functions such as `CreateDC()`, `ResetDC()`, and `DocumentProperties()`, the function may fail.

To reliably use a DEVMODE with a device driver, create and modify it by following these steps:

1. Determine the required size of the buffer from the device, and then allocate enough memory for it.

   `DocumentProperties()` returns the number of bytes that are required for a DEVMODE buffer when the last parameter is set to 0. The sample code in this

article uses this technique to determine the correct size of the buffer. The sample code then uses the C run-time memory allocation function of `malloc()` to allocate a buffer that is large enough. Because `DocumentProperties()` and functions like `ResetDC()` and `CreateDC()` take pointers to a DEVMODE as a parameter, most applications can allocate memory that is addressed by a pointer.

However, functions such as the common `PrintDlg()` take parameters that are needed to be handles to global memory. If a program uses the final DEVMODE buffer as a parameter to one of these functions, it should allocate memory using `GlobalAlloc()` and obtain a pointer to the buffer using `GlobalLock()`.

2. Ask the device driver to initialize the DEVMODE buffer with the default settings.

   The sample code calls `DocumentProperties()` a second time to initialize the allocated buffer with the current default settings. `DocumentProperties()` fills the buffer referred to as the *pDevModeOutput* parameter with the printer's current settings when the `DM_OUT_BUFFER` command is passed in the fMode parameter.

3. Make changes to the public portion of the DEVMODE and ask the device driver to merge the changes into the private portion of the DEVMODE by calling `DocumentProperties()`.

   After initializing the buffer with current settings in step 2, the sample code makes changes to the public part of the DEVMODE. See the Win32 SDK documentation for descriptions of the DEVMODE members. This sample code determines whether the printer can use orientation and duplex (double-sided) settings and changes them appropriately.

   > ⓘ **Note**
   >
   > A flag in a DEVMODE's dmFields member is only an indication that a printer uses the associated structure member. Printers have a variety of different physical characteristics and, therefore, may only support a subset of a DEVMODE's documented capabilities. To determine the supported settings of a DEVMODE's field, applications should use `DeviceCapabilities()`.

   The sample code then makes a third call to `DocumentProperties()` and passes the DEVMODE buffer in both the pDevModeInput and pDevModeOutput parameters. It also passes the combined commands of DM_IN_BUFFER and DM_OUT_BUFFER in the fMode parameter by using the OR("|") operator. These commands tell the function to take whatever settings are contained in the input buffer and to merge

them with the current settings for the device. Then it writes the result to the buffer specified in the out parameter.

> **Note**
>
> `DocumentProperties()` refers to a specific printer by a handle to a printer: hPrinter. This handle is obtained from `OpenPrinter()`, which the sample code also illustrates. `OpenPrinter()` requires the name of a printer, which is typically the friendly name of the printer as it appears in the Operating System's shell. This name can be obtained from `EnumPrinters()`, from the DEVNAMES structure returned by `PrintDlg()`, or from the Default Printer.

In this article, the first two steps of allocating the correct size of buffer and initializing that buffer is performed with `DocumentProperties()`. You can also follow these steps by using `GetPrinter()`.

For more information and an example of this, see Modify printer settings by using the SetPrinter function.

Sample Code

The sample code follows these three steps for obtaining and changing the DEVMODE buffer. The function takes a named printer and configures a DEVMODE to print double-sided and in the landscape orientation if it supports these features. The resulting DEVMODE that is returned to the caller is suitable for other API calls that use DEVMODE buffers, such as `CreateDC()`, `SetPrinter()`, `PrintDlg()`, or `ResetDC()`. When the caller has completed using the DEVMODE buffer, the caller is responsible for freeing the memory.

C++

```
LPDEVMODE GetLandscapeDevMode(HWND hWnd, char *pDevice)
{
   HANDLE hPrinter;
   LPDEVMODE pDevMode;
   DWORD dwNeeded, dwRet;

   /* Start by opening the printer */
   if (!OpenPrinter(pDevice, &hPrinter, NULL))
       return NULL;

   /*
   * Step 1:
   * Allocate a buffer of the correct size.
```

```
    */
    dwNeeded = DocumentProperties(hWnd,
    hPrinter, /* Handle to our printer. */
    pDevice, /* Name of the printer. */
    NULL, /* Asking for size, so */
    NULL, /* these are not used. */
    0); /* Zero returns buffer size. */
    pDevMode = (LPDEVMODE)malloc(dwNeeded);

    /*
    * Step 2:
    * Get the default DevMode for the printer and
    * modify it for your needs.
    */
    dwRet = DocumentProperties(hWnd,
    hPrinter,
    pDevice,
    pDevMode, /* The address of the buffer to fill. */
    NULL, /* Not using the input buffer. */
    DM_OUT_BUFFER); /* Have the output buffer filled. */
    if (dwRet != IDOK)
    {
        /* If failure, cleanup and return failure. */
        free(pDevMode);
        ClosePrinter(hPrinter);
        return NULL;
    }

    /*
    * Make changes to the DevMode which are supported.
    */
    if (pDevMode->dmFields & DM_ORIENTATION)
    {
        /* If the printer supports paper orientation, set it.*/
        pDevMode->dmOrientation = DMORIENT_LANDSCAPE;
    }

    if (pDevMode->dmFields & DM_DUPLEX)
    {
        /* If it supports duplex printing, use it. */
        pDevMode->dmDuplex = DMDUP_HORIZONTAL;
    }

    /*
    * Step 3:
    * Merge the new settings with the old.
    * This gives the driver an opportunity to update any private
    * portions of the DevMode structure.
    */
    dwRet = DocumentProperties(hWnd,
    hPrinter,
    pDevice,
    pDevMode, /* Reuse our buffer for output. */
    pDevMode, /* Pass the driver our changes. */
    DM_IN_BUFFER | /* Commands to Merge our changes and */
```

```
    DM_OUT_BUFFER); /* write the result. */

    /* Finished with the printer */
    ClosePrinter(hPrinter);

    if (dwRet != IDOK)
    {
        /* If failure, cleanup and return failure. */
        free(pDevMode);
        return NULL;
    }

    /* Return the modified DevMode structure. */
    return pDevMode;
}
```

Feedback

Was this page helpful? Yes No

Page setup dialog might unexpectedly close after the print support app is installed

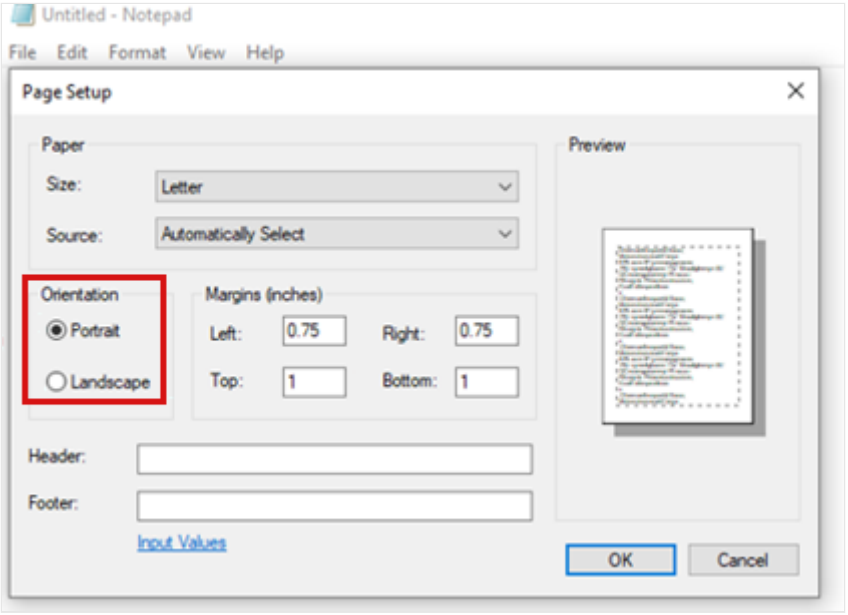
Article • 09/30/2024

Symptoms

Consider the following scenario:

- You're using Windows 10, version 22H2.
- You're developing a print support app (PSA) for IPP-based printers.
- You're using the Notepad or Paint application.

In this scenario, after you install the PSA on the system, toggling the page orientation setting might cause the host application to crash. For example, in *notepad.exe*, you can see the issue in the **Page Setup** dialog, as shown in the following screenshot.



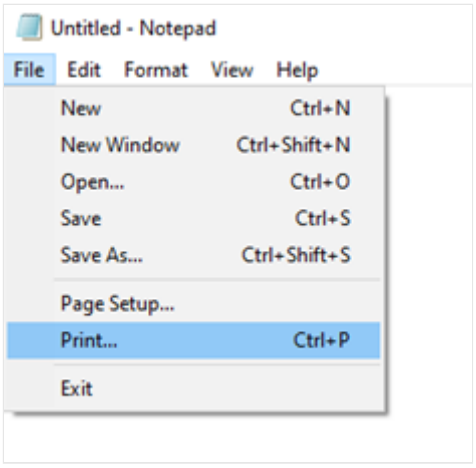
Cause

This issue occurs because of a problem with the common dialog.

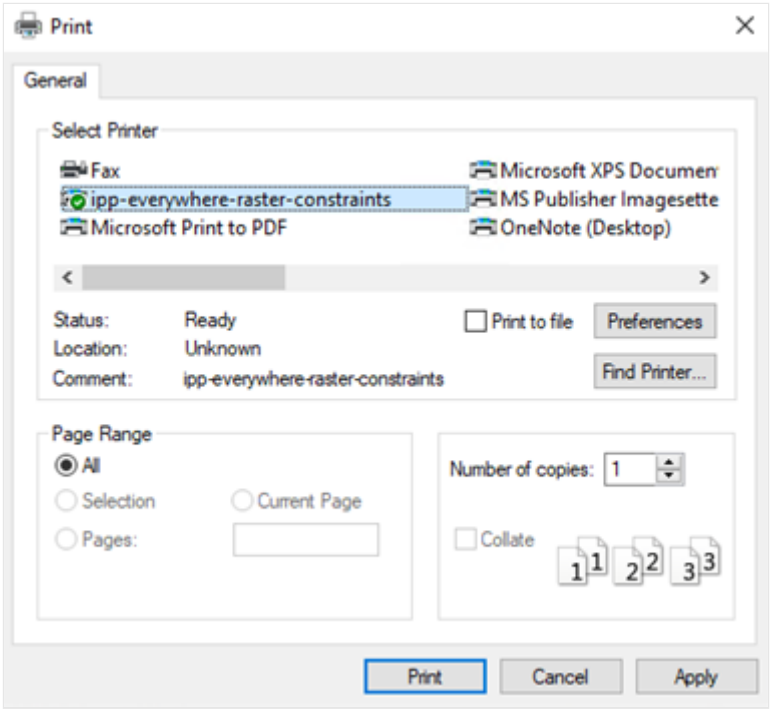
Workaround

To work around this issue, follow these steps to switch the page orientation setting:

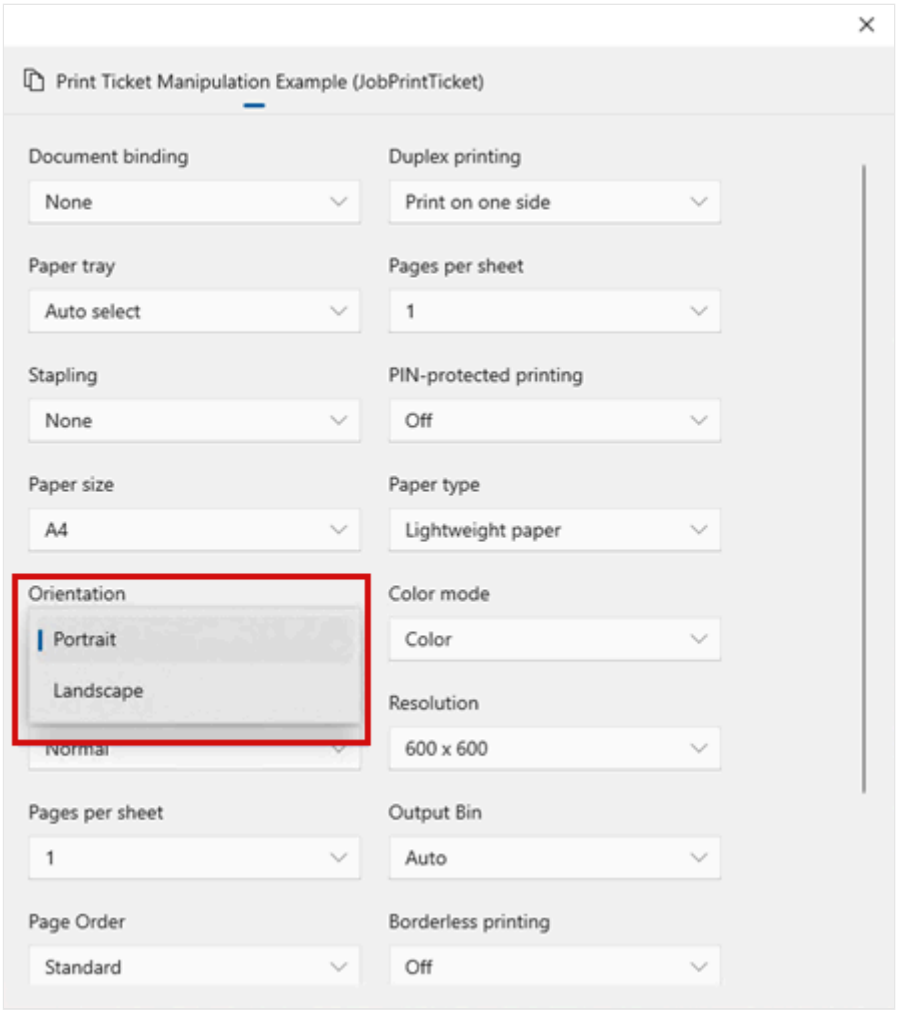
1. Select the **File** menu, and then select **Print**.



2. Select the printer you want to use, and then select the **Preferences** button.



3. Once you confirm that the PSA has been launched, you can modify the page orientation setting there.



Feedback

Was this page helpful? Yes No

The Win32_NetworkAdapterConfiguration class is unable to retrieve information about PPPoE (Point-to-point protocol over Ethernet) and VPN (Virtual Private Network)

Article • 12/19/2023

This article helps you resolve the problem where he `Win32_NetworkAdapterConfiguration` class is unable to retrieve information about PPPoE (Point-to-point protocol over Ethernet) and VPN (Virtual Private Network).

Applies to: Windows Vista
Original KB number: 2549091

Symptoms

On Windows Vista and later, the `Win32_NetworkAdapterConfiguration` class is unable to retrieve information about a PPPoE connection and VPN connection.

If a program is designed to get the information about the dial-up connection or a virtual private network by using the `Win32_NetworkAdapterConfiguration` class on Windows XP, it may not work on Windows Vista and later.

For more information about the `Win32_NetworkAdapterConfiguration` class, see Win32_NetworkAdapterConfiguration class.

Cause

On Windows Vista and later, the `Win32_NetworkAdapterConfiguration` class does not create an instance for a PPPoE connection or VPN connection. Microsoft has confirmed that this to be a problem in our product.

Resolution

On Windows Vista and later, you can retrieve almost the same information as the `Win32_NetworkAdapterConfigurationthe` class does regarding a PPPoE connection or VPN connection by using either of the following methods. It would be highly appreciated if you consider that either of the following method is acceptable.

1. Use the .NET Framework `NetworkInterface` class.

   Use the `NetworkInterface.GetAllNetworkInterfaces` method to get a `NetworkInterface` array. Then, go through the `NetworkInterface` array to find a `NetworkInterface` instance that has the `NetworkInterface.NetworkInterfaceType` property set as Ppp. Each value of a PPPoE or VPN connection can be retrieved by referencing each property that this instance has.

   For more information on the `NetworkInterface` class or the sample code for this, see NetworkInterface Class.

   For more information on each property of the `NetworkInterface` class, see NetworkInterface Class.

2. Use the `GetAdaptersAddresses` API.

   Use the `GetAdaptersAddresses` API to get the `IP_ADAPTER_ADDRESSES` structure. Then, go through the linked list of `IP_ADAPTER_ADDRESSES` structures to find an element that has the `IfType` member set as `IF_TYPE_PPP`. Each value of a PPPoE or VPN connection can be retrieved by referencing each member of the element.

   For more information on the `GetAdaptersAddresses` API or the sample code for this, see GetAdaptersAddresses function (iphlpapi.h).

   For more information on each member of the `IP_ADAPTER_ADDRESSES` structure, see IP_ADAPTER_ADDRESSES_LH structure (iptypes.h).

Steps to reproduce

VB

```
Set objWMIService = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")
Set colItems = objWMIService.ExecQuery _("Select * From
Win32_NetworkAdapterConfiguration Where IPEnabled = True")
```

Feedback

Was this page helpful? Yes No

PowerShell script files re-signed with SignTool may be corrupted

Article • 12/19/2023

Applies to: Windows SDK for Windows 10

Symptoms

Assume that you made changes to a Windows PowerShell script file (.ps1) that already has a digital signature. The script file may be corrupted after using SignTool (SignTool.exe) to add a new signature.

Cause

When using SignTool to sign a script file for the first time, a carriage return and a line feed are added at the end of the file before the signature is added. When using SignTool to add a new signature to the file, the old signature and the two characters before the signature are deleted. SignTool may corrupt the script if the two characters before the signature are not the carriage return and line feed characters added when the script was signed. This can occur when editing the script file after it has been signed using an editor that replaces carriage return and line feed pairs with a line feed character.

Workaround

Remove the existing signature before signing the file.

Feedback

Add-in component guidelines for core OS processes

Article • 12/19/2023

Applies to: Windows
Original KB number: 841927

> **Important**
>
> We recommend that you use only C or C++ languages and Win32 APIs for any add-in components that are loaded by core OS processes such as *lsass.exe*, *winlogon.exe*, and *logout.exe*.

The behavior of any high-level language, framework, or runtime in the components that are loaded by core operating system processes is undefined. For example, Microsoft .NET Framework and the common language runtime weren't designed to run in the context of core operating system processes.

Here is a partial list of high-level languages, frameworks, and runtimes for which behavior is undefined in the context of core OS processes:

- .NET and .NET Framework
- .NET and .NET Framework languages
  - C#
  - Visual Basic
  - Managed Extensions for C++
- Java
- Microsoft Component Object Model (COM)
- Microsoft COM+
- Microsoft Distributed Component Object Model (DCOM)
- Microsoft Foundation Classes (MFC)
- Microsoft Active Template Library (ATL) framework

More information

For more information about add-in components that are loaded by core OS processes, see the following articles:

- Winlogon and Credential providers

- Winlogon and GINA
- Winlogon Notification Packages
- Password Filters
- Security Support Providers (SSPs)
- Authentication Packages
- Subauthentication Packages
- Cryptographic Service Providers

Feedback

Was this page helpful? Yes No

GetTickCount resets to zero after approximately 776 days

Article • 02/01/2024

This article describes a problem where the value returned by `GetTickCount` resets to `zero` after approximately 776 days.

Symptoms

The time returned by the GetTickCount function resets from `0x9FFFFFF0` to `zero` if the system runs continuously for approximately 776 days.

This problem occurs in 32-bit applications running on Windows 8, Windows Server 2012, and later.

Status

Microsoft has confirmed this is a problem in Windows 8, Windows Server 2012, and later.

Workaround

Use the GetTickCount64 function instead.

More information

This problem isn't related to the behavior described in the GetTickCount function documentation, where the time wraps around from `0xFFFFFFFF` to `zero` if the system runs continuously for 49.7 days.

The number of days before this problem occurs may vary depending on the resolution of the system timer. The problem occurs after approximately 776 days on systems whose system timer resolution is 15.6 milliseconds.

Feedback

Was this page helpful? Yes No

The UMDH tool generates warnings

Article • 02/28/2025

This article describes a known issue with the User-Mode Dump Heap (UMDH) tool that is installed with the Debugging Tools for Windows. The issue affects the UMDH tool included in the Windows SDK for Windows 11, the Windows Driver Kit for Windows 11, and the standalone toolsets.

Symptoms

When the UMDH tool (also known as **umdh.exe**) executes, it displays the following messages and fails to capture memory allocations:

Output

```
Warning:
Warning: UMDH didn't find any allocations that have stacks collected.
Warning: Traces could not be collected because the Stack Trace Database is
full.
Warning: Increase the size of the Stack Trace Database using GFLAGS.
Warning:
```

Cause

This is a bug in the UMDH tool included in the Windows SDK for Windows 11 and the Windows Driver Kit for Windows 11.

Workaround

To work around the issue, install the latest version of the Windows SDK for Windows 10 or the Windows Driver Kit for Windows 10 and use the UMDH tool in it.

Feedback

User32.dll or Kernel32.dll does not initialize

Article • 12/19/2023

This article describes an issue where an application that is executed by `CreateProcess` or `CreateProcessAsUser` may fail.

Applies to: Microsoft Windows
Original KB number: 184802

Symptoms

An application that is executed by `CreateProcess` or `CreateProcessAsUser` may fail, and you receive one of the following error messages:

> Initialization of the dynamic library <system>\system32\user32.dll failed. The process is terminating abnormally. Initialization of the dynamic library <system>\system32\kernel32.dll failed. The process is terminating abnormally.

Additionally, the failed process returns exit code 128 or the following:

> error:ERROR_WAIT_NO_CHILDREN

Cause

This failure occurs for one of the following reasons:

- The executed process does not have correct security access to the window station and desktop that are associated with the process.
- The system ran out of desktop heap.

More information

- Cause 1

  The executed process does not have correct security access to the window station and desktop that are associated with the process.

The lpDesktop member of the STARTUPINFO structure that is passed to `CreateProcess` or `CreateProcessAsUser` specifies the window station and desktop that are associated with the executed process. The executed process must have correct security access to the specified window station and desktop.

- Cause 2

  The system ran out of desktop heap.

  Every desktop object on the system has a desktop heap that is associated with it. The desktop object uses the heap to store menus, hooks, strings, and windows. In Windows Server 2003 and Windows XP 32-bit, the system allocates desktop heap from a system-wide 48 megabytes (MB) buffer. In addition to desktop heaps, printer drivers and font drivers also use this buffer.

  Desktops are associated with window stations. A window station can contain zero or more desktops. You can change the size of the desktop heap that is allocated for a desktop that is associated with a window station by changing the following registry value.

  > ⓘ **Note**
  >
  > We do not recommend that you use the /3GB switch. The /3GB switch is specified in the Boot.ini file. The /3GB switch is supported only for 32-bit operating systems.
  > `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\SubSystems\Windows`

In Windows Server 2003 and Windows XP 32-bit, the default data for this registry value will resemble the following (all on one line):

Console

```
%SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows
SharedSection=1024,3072,512 Windows=On SubSystemType=Windows
ServerDll=basesrv,1 ServerDll=winsrv:UserServerDllInitialization,3
ServerDll=winsrv:ConServerDllInitialization,2 ProfileControl=Off
MaxRequestThreads=16
```

In different versions of Windows, the default data for this registry value will resemble the following:

- For Windows Vista RTM (32-bit)

```
Console

SharedSection=1024,3072,512
```

- For Windows Vista SP1, Windows 7, Windows 8, Windows 8.1 (32-bit), and Windows Server 2008 (32-bit)

```
Console

SharedSection=1024,12288,512
```

- For Windows Vista, Windows 7, Windows 8, Windows 8.1 (64-bit), Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2 (64-bit)

```
Console

SharedSection=1024,20480,768
```

The numeric values that following `SharedSection=` control how the desktop heap is allocated. These `SharedSection` values are specified in kilobytes. There are separate settings for desktops that are associated with interactive and noninteractive window stations.

> **Note**
>
> If you change the `SharedSection` values in the registry, you must restart the system for the changes to take effect.

> **Important**
>
> This section, method, or task contains steps that tell you how to modify the registry. However, serious problems might occur if you modify the registry incorrectly. Therefore, make sure that you follow these steps carefully. For added protection, back up the registry before you modify it. Then, you can restore the registry if a problem occurs. For more information about how to back up and restore the registry, see **How to back up and restore the registry in Windows**.

The first `SharedSection` value (1024) is the shared heap size common to all desktops. This includes the global handle table. This table holds handles to windows, menus, icons,

cursors, and so on, and shared system settings. It is unlikely that you would ever have to change this value.

The second `SharedSection` value is the size of the desktop heap for each desktop that is associated with the **interactive** window station WinSta0. User objects such as hooks, menus, strings, and windows consume memory in this desktop heap. It is unlikely that you would ever have to change this value.

Each desktop that is created in the interactive window station uses the default desktop heap of 3,072 KB. By default, the system creates the following three desktops in Winsta0:

- Winlogon
- Default

  The Default application desktop will be used by all the processes for which `Winsta0\default` is specified in the STARTUPINFO.lpDesktop structure member. When the lpDesktop structure member is NULL, the window station and desktop are inherited from the parent process. All services that are executed under the LocalSystem account with the Allow Service to Interact with Desktop startup option selected will use `Winsta0\Default`. All these processes will share the desktop heap that is associated with the Default application desktop.

- Screen saver

  The screen saver desktop is created in the interactive window station (WinSta0) when a screen saver is displayed.

The third SharedSection value is the size of the desktop heap for each desktop that is associated with a noninteractive window station. If this value is not present, the size of the desktop heap for noninteractive window stations will be same as the size that is specified for interactive window stations (that is, the second SharedSection value).

If only two SharedSection values are present, you can add a third value to specify the size of the desktop heap for desktops that are created in noninteractive window stations.

Every service process that is executed under a user account will receive a new desktop in a noninteractive window station that is created by the Service Control Manager (SCM). Therefore, each service that is executed under a user account will consume the number of kilobytes of desktop heap that is specified in the third SharedSection value. All services that are executed under the LocalSystem account when Allow Service to Interact with the Desktop is not selected share the desktop heap of the Default desktop in the noninteractive service windows station (Service-0x0-3e7$).

The total desktop heap that is being used in the interactive and noninteractive window stations must fit in the buffer.

Decreasing the second or third SharedSection value will increase the number of desktops that can be created in the corresponding window stations. Smaller values will limit the number of hooks, menus, strings, and windows that can be created in a desktop. On the other hand, increasing the second or third SharedSection value will decrease the number of desktops that can be created. However, this will also increase the number of hooks, menus, strings, and windows that can be created in a desktop.

Because the SCM creates a new desktop in the noninteractive window station for every service process that is running under a user account, a larger third SharedSection value will reduce the number of user account services that can run successfully on the system. The minimum that can be specified for the second or third SharedSection value is 128. Any attempt to use a smaller value will instead use 128.

Desktop heap is allocated by User32.dll when a process needs user objects. If an application is not dependent on User32.dll, it will not consume desktop heap.

> ⓘ **Note**
>
> In Windows Server 2003, the specific event is logged in the System log when one of the following conditions is true:

- If the desktop heap becomes full, the following event is logged:

  Output

  ```
  Event Type: Warning
  Event Source: Win32k
  Event Category: None
  Event ID: 243
  Date: Date
  Time: Time
  User: N/A
  Computer: ServerName
  Description: A desktop heap allocation failed.
  ```

  In this case, increase the desktop heap size.

- If the total desktop heap becomes the system-wide buffer size, the following event is logged:

  Output

```
Event Type: Warning
Event Source: Win32k
Event Category: None
Event ID: 244
Date: Date
Time: Time
User: N/A
Computer: ServerName
Description: Failed to create a desktop due to desktop heap exhaustion.
```

In this case, decrease the desktop heap size.

In Windows Server 2003, a system-wide buffer is 20 MB when one of the following conditions is true:

- You are in a Terminal Services environment.
- The /3GB switch is specified in the Boot.ini file.

Applies to

- Microsoft Windows XP Professional
- Microsoft Windows XP Home Edition
- Windows Vista Ultimate
- Windows Vista Enterprise
- Windows Vista Business
- Windows Vista Home Premium
- Windows 7 Ultimate
- Windows 7 Enterprise
- Windows 7 Professional
- Windows 7 Home Premium
- Windows 8 Enterprise
- Windows 8 Pro, Windows 8
- Windows 8.1 Enterprise
- Windows 8.1 Pro
- Windows 8.1
- Microsoft Windows Server 2003 Datacenter Edition (32-bit x86)
- Microsoft Windows Server 2003 Datacenter x64 Edition
- Microsoft Windows Server 2003 Enterprise Edition (32-bit x86)
- Microsoft Windows Server 2003 Enterprise x64 Edition
- Microsoft Windows Server 2003 Standard Edition (32-bit x86)
- Microsoft Windows Server 2003 Standard x64 Edition
- Windows Server 2008 Datacenter

- Windows Server 2008 Enterprise
- Windows Server 2008 R2 Datacenter
- Windows Server 2008 R2 Enterprise
- Windows Server 2008 Standard
- Windows Server 2012 Datacenter
- Windows Server 2012 Standard
- Windows Server 2012 R2 Datacenter
- Windows Server 2012 R2 Standard

Feedback

Was this page helpful? Yes No