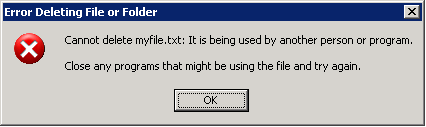
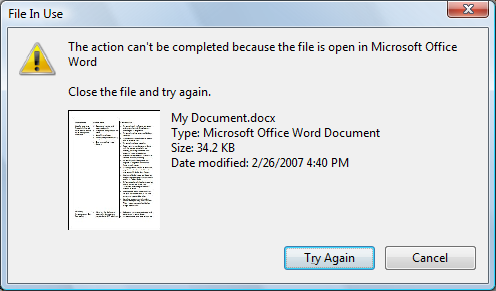
**Your File Is In Use… Demystified**

In past versions of Windows, when a user encountered a file that is in use by another process, he would be presented with an unfriendly dialog like this:



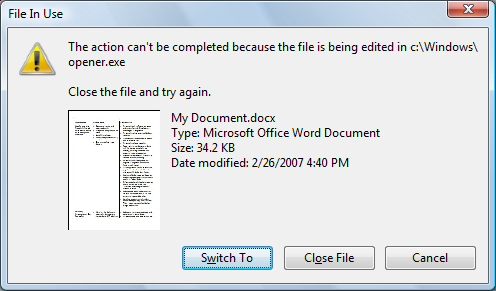
What process is using this file? What should the user do? There is nothing actionable for the user on this dialog. Some savvy users may use a [utility](http://www.microsoft.com/technet/sysinternals/Utilities/Handle.mspx) to determine which process has an open handle on the file. Yet, a typical user will restart their machine in the hope that he can get access to the file after logging back in. In Windows Vista, work was done to give the user more information about who has the file opened as well as exposing capabilities to gain control of the file. This is accomplished by [IFileIsInUse](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/shellcc/platform/shell/reference/ifaces/ifileisinuse/ifileisinuse.asp): an interface applications can implement and associate with the files they currently have open.

If you encounter a file that is currently opened in another process in Vista while performing operations on files in Windows Explorer, you will see a much different dialog:



This is because the operations engine in Windows Explorer is applying an implementation of the IFileIsInUse interface to the file it encountered. Now the user knows which program has the file open and can close the file from that program and click the *Try Again* button to repeat the operation that was being performed.

The implementer of the IFileIsInUse interface can specify further capabilities such as the ability to switch to the main window of the application that has the file open or to simply close the file in that application from the File In Use dialog. If these capabilities are exposed, the above dialog would now look like this:



The user can now click the *Switch To* button to set the window of the application that has the file open to the foreground. Also, the user could simply click *Close File* to notify the application to close its open handle to the file.

So how does an application add similar functionality? It must implement the IFileIsInUse interface.

The IFileIsInUse interface exposes the following methods:

|  |  |
| --- | --- |
| **Method** | **Description** |
| HRESULT CloseFile() | Closes the file currently in use. |
| HRESULT GetAppName(PWSTR \*ppszName) | Gets a string associated with an application name. |
| HRESULT GetCapabilities(DWORD \*pdwCapFlags) | Gets the capabilities flag for file in use. See below for a description of the accepted values. |
| HRESULT GetSwitchToHWND(HWND \*phwnd) | Gets the HWND associated with the application using the file. |
| HRESULT GetUsage(FILE\_USAGE\_TYPE \*pfut) | Gets the current file usage type. This is interpreted as editing, playing, or generic usage. See the [FILE\_USAGE\_TYPE](http://msdn.microsoft.com/library/en-us/shellcc/platform/shell/reference/enums/file_usage_type.asp) flags. See below for a description of the accepted values. |

**FILE\_USAGE\_TYPE Flags**

This can be one of the following values:

|  |  |
| --- | --- |
| **Name** | **Description** |
| FUT\_PLAYING | Indicates that the file is currently playing in the process that has it opened. |
| FUT\_EDITING | Indicates that the file is currently being edited in the process that has it opened. |
| FUT\_GENERIC | Indicates that the file is currently in use in the process that has it opened. |

**Capability Flags**

This can be any combination of the following values:

|  |  |
| --- | --- |
| **Name** | **Description** |
| OF\_CAP\_CANSWITCHTO | Indicates that the caller can retrieve the HWND associated with the process that has the file opened by calling GetSwitchToHWND |
| OF\_CAP\_CANCLOSE | Indicates that the caller can close the file by calling CloseFile |

If the application that has the file in use specifies the above flags, buttons on the file in use dialog will be visible – allowing the user to bring the application to the foreground or simply close the file directly and continue with the operation that was interrupted.

**The Running Object Table**

So how does an application notify other applications that the file is in use and what capabilities are available? This is done by inserting the instantiated IFileIsInUse object into the Running Object Table. The Running Object Table (ROT) is a globally accessible lookup table on each workstation. The ROT keeps track of those objects that can be identified by a moniker and that are currently running on the workstation. When a client tries to bind a moniker to an object, the moniker checks the ROT to see if the object is already running. This allows the moniker to bind to the current instance instead of loading a new one.

For example, to place an object in the ROT:

DWORD dwCookie;

IRunningObjectTable \*prot;

HRESULT hr = GetRunningObjectTable(NULL, &prot);

if (SUCCEEDED(hr))

{

IMoniker \*pmk;

hr = CreateFileMoniker(pszFileName, &pmk);

if (SUCCEEDED(hr))

{

// Add ROTFLAGS\_ALLOWANYCLIENT to make this work accross security boundaries

hr = prot->Register(ROTFLAGS\_REGISTRATIONKEEPSALIVE | ROTFLAGS\_ALLOWANYCLIENT,

static\_cast<IFileIsInUse \*>(this), pmk, &dwCookie);

if (hr == CO\_E\_WRONG\_SERVER\_IDENTITY)

{

// this failure is due to ROTFLAGS\_ALLOWANYCLIENT and the

// fact that we don't have the AppID registered for our CLSID.

// Try again without ROTFLAGS\_ALLOWANYCLIENT knowing that this

// means this can only work in the scope of apps running with

// the same MIC level.

hr = prot->Register(ROTFLAGS\_REGISTRATIONKEEPSALIVE,

static\_cast<IFileIsInUse \*>(this), pmk,

&dwCookie);

}

pmk->Release();

}

prot->Release();

}

In the above code, we call [GetRunningObjectTable](http://msdn2.microsoft.com/en-us/library/ms684004.aspx) to retrieve an instance of [IRunningObjectTable](http://msdn2.microsoft.com/en-us/library/ms695276.aspx). We then create an [IMoniker](http://msdn2.microsoft.com/en-us/library/ms679705.aspx) for the file that is currently in use. This moniker is then inserted into the ROT in the call to IRunningObjectTable::Register. We specify the ROTFLAGS\_ALLOWANYCLIENT in our call to Register. This is required to allow our entry in the ROT to work across security boundaries. This requires AppID registration so COM can inspect our security settings. Without this the Register()call will fail. In that case, we call Register() again, but without the ROTFLAGS\_ALLOWANYCLIENT – which will allow our application to work correctly but only in the same security level. The dwCookie parameter is used to identify the entry in later calls to retrieve or remove it from the ROT.

In the below code, the cookie is used to remove the moniker in the ROT.

IRunningObjectTable \*prot;

HRESULT hr = GetRunningObjectTable(NULL, &prot);

if (SUCCEEDED(hr))

{

hr = prot->Revoke(dwCookie);

prot->Release();

}

So from the above code we now know how to insert our object into the ROT so other applications can find it. But how does an application check if a particular file has been added as an instance of an IFileIsInUse object to the ROT? The application needs to enumerate the contents of the ROT to find a matching moniker.

IFileIsInUse \*pfiu;

PCWSTR pszFile = L"c:\\some\\full\\path\\to\\file";

IRunningObjectTable \*prot;

HRESULT hr = GetRunningObjectTable(0, &prot);

if (SUCCEEDED(hr))

{

IMoniker \*pmkFile;

hr = CreateFileMoniker(pszFile, &pmkFile);

if (SUCCEEDED(hr))

{

IEnumMoniker \*penumMk;

hr = prot->EnumRunning(&penumMk);

if (SUCCEEDED(hr))

{

hr = E\_FAIL;

ULONG celt;

IMoniker \*pmk;

while (FAILED(hr) && (penumMk->Next(1, &pmk, &celt) == S\_OK))

{

DWORD dwType;

if (SUCCEEDED(pmk->IsSystemMoniker(&dwType)) &&

(dwType == MKSYS\_FILEMONIKER))

{

// Is this a moniker prefix?

IMoniker \*pmkPrefix;

if (SUCCEEDED(pmkFile->CommonPrefixWith(pmk, &pmkPrefix)))

{

if (S\_OK == pmkFile->IsEqual(pmkPrefix))

{

// Get the IFileIsInUse instance

IUnknown \*punk;

if (prot->GetObject(pmk, &punk) == S\_OK)

{

hr = punk->QueryInterface(IID\_PPV\_ARGS(&pfiu));

punk->Release();

}

}

pmkPrefix->Release();

}

}

pmk->Release();

}

penumMk->Release();

}

pmkFile->Release();

}

prot->Release();

}

If the moniker was found in the ROT, the application now has an instance of an IFileIsInUse that was created in the application that is using the file. This object can then be used to query for the capabilities that are available or any other method exposed by IFileIsInUse. This is essentially what Windows Explorer does when it encounters a file that is in use during file operations.

The goal of creating this interface is to allow applications to play nice with one another when contending for access to common files. As more developers implement this interface, the better it is for the entire community.

**Windows XP Support**

Windows XP implemented a less ambitious version of this feature using the ROT and IOleObject and IOleWindow. Since many OLE applications implemented these interfaces, one could discover the application name (via IOleObject::GetUserClassID() that could then be used to look up the application name) and offer the “Switch To” function via the IOleWindow::GetWindow() method.

## Building the IFileIsInUse Sample

The FileIsInUse sample creates an IFileIsInUse instance of a file it has opened and adds it to the ROT with all capabilities enabled. You can then verify the functionality that the interface provides by attempting to move or delete the file through Windows Explorer. Windows Explorer will show a File In Use dialog as shown below.

1. Download and install the [Windows SDK](http://www.microsoft.com/downloads/details.aspx?FamilyId=7614FE22-8A64-4DFB-AA0C-DB53035F40A0&displaylang=en).
2. Download the IFileIsInUse\_Sample
3. Launch FileIsInUseSample.sln in Visual Studio
4. Open the properties for the project
5. Add a path to the SDK includes to the C/C++ - General page
6. Add a path to the SDK libs to the Linker – General page
7. Build

|  |  |
| --- | --- |
|  |  |

After launching the FileIsInUseSample.exe, you can drag and drop files to the list view or click the “Open File…” menu item from the File menu (Note that drag and drop does not work across security boundaries). After opening the file and adding an entry in the ROT with the IFileIsInUse implementation provided, the full path of the file will appear on the dialog. You can now attempt to delete the file from Windows Explorer. When you close the file from the File In Use dialog in Windows Explorer it will be removed from the dialog. Also, if you click “Close File” from the File menu, it will close the file, remove it from the ROT and clear the file path from the dialog.

While this sample does very little, it should helpful in importing this functionality with the provided IFileIsInUse implementation.