

Forum:[Hack Forums](#)

Thread: [clipboard file location](#)

macpromo - May 30, 2020, 12:15 pm

I pressed ctrl+c on a file, not a text. I want set textbox.text or a string the location of the file. suppose c:\myfile.abc in clipboard. I want set text equal to the location/path present in the clipboard.

He B TeMy - May 30, 2020, 2:11 pm

You would have to [query](#) the clipboard looking for the [CF_HDROP](#) format to extract the file path(s) from.

You can look at Solution 3 [here](#) for example.

Accord - May 30, 2020, 6:08 pm

You can google it, heres the result,

<http://csharpHelper.com/blog/2018/02/paste-files-from-the-clipboard-in-c/>

MavenGradle - May 31, 2020, 7:28 am

(May 30, 2020 02:11 PM)ff20 Wrote: [\[CITE\]](#) You would have to [query](#) the clipboard looking for the [CF_HDROP](#) format to extract the file path(s) from.

You can look at Solution 3 [here](#) for example.

You do not need to query the clipboard. You can simply wait for Windows to send a message by placing a hook/viewer on the clipboard:

Code [DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)]
internal static extern IntPtr SetClipboardViewer(IntPtr hWndNewViewer);
<https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setclipboardviewer>.

You will need the help of these two functions as well:

Code [DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)]
internal static extern bool ChangeClipboardChain(IntPtr hWndRemove, IntPtr hWndNewNext);

[DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)]
internal static extern IntPtr SendMessage(IntPtr hWnd, int msg, IntPtr wParam, IntPtr lParam);
<https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-changeclipboardchain>
<https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendmessage>

Example usage:

Code WindowInteropHelper wih = new WindowInteropHelper(_window);
IntPtr hWndSource = HwndSource.FromHwnd(wih.Handle);

if (hWndSource == null) return;

hWndSource.AddHook(WinProc);
IntPtr hWndNextViewer = SetClipboardViewer(_hWndSource.Handle);

And here's where the message is sent to:

Code private IntPtr WinProc(IntPtr hwnd, int msg, IntPtr wParam, IntPtr lParam, ref bool handled)
{

```
// Capture the data here
}
```

You'll want to listen to the message [0x0308](#).

Don't forget to call GetLastError if something goes wrong:

<https://docs.microsoft.com/en-us/windows/win32/api/errhandlingapi/nf-errhandlingapi-getlasterror>

And finally, you will need to remove the 'hook' and dispose of your resources properly:

Code if (hWndSource != null)

```
{
    ChangeClipboardChain(hWndSource.Handle, hWndNextViewer);
    hWndNextViewer = IntPtr.Zero;
    hWndSource.RemoveHook(WinProc);
}
```

Quote: The windows that are part of the clipboard viewer chain, called clipboard viewer windows, must process the clipboard messages WM_CHANGECHAIN and WM_DRAWCLIPBOARD. Each clipboard viewer window calls the SendMessage function to pass these messages to the next window in the clipboard viewer chain.

A clipboard viewer window must eventually remove itself from the clipboard viewer chain by calling the ChangeClipboardChain function — for example, in response to the WM_DESTROY message.

That part about SendMessage is very important. For example in the WinProc function (found above) after listening to 0x0308:

Code SendMessage(hWndNextViewer, msg, wParam, lParam);

Likewise, when receiving the message [0x030D](#):

Code if (wParam == hWndNextViewer)

```
{
    hWndNextViewer = lParam;
}
else if (hWndNextViewer != IntPtr.Zero)
{
    SendMessage(hWndNextViewer, msg, wParam, lParam);
}
```

In terms of [0x0002](#):

See the code above about disposing of your viewer/hook. That should be sufficient information if you follow the documentation..

Perhaps this is a bit overkill though, and you might only want to listen to the events that are specific to your window. In that case, simply add DragDrop event handler to your Form:

```
Code FrmMain.DragDrop += new DragEventHandler(FrmMain_DragDrop);
Code public void
FrmMain_DragDrop(object sender, DragEventArgs e) {
    string[] files = (string[])e.Data.GetData(DataFormats.FileDrop);
    foreach (string file in files) Console.WriteLine(file);
}
```

Or the equivalent in WPF:

<https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/drag-and-drop-overview#drop-target-events>

He B TeMy - May 31, 2020, 8:47 am

(May 31, 2020 07:28 AM)MavenGradle Wrote: [\[CITE\]](#) You do not need to query the clipboard. You can simply wait for Windows to send a message by placing a hook/viewer on the clipboard:

I suggested a simple solution with less-overhead, but you went all-in... OP hasn't responded to any comment...

Anyway, why would you suggest him to use the Legacy SetClipboardViewer API instead of [AddClipboardFormatListener](#) ?

I'm not saying you didn't do it correctly in explaining him the right way of disposing things... i just have a thing for "legacy" stuff that has no guarantee it will work if some other app f'ks up the chain.

MavenGradle - May 31, 2020, 9:03 am

(May 31, 2020 08:47 AM)ff20 Wrote: [\[CITE\]](#) I suggested a simple solution with less-overhead, but you went all-in... OP hasn't responded to any comment...

Anyway, why would you suggest him to use the Legacy SetClipboardViewer API instead of [AddClipboardFormatListener](#) ?

I'm not saying you didn't do it correctly in explaining him the right way of disposing things... i just have a thing for "legacy" stuff that has no guarantee it will work if some other app f'ks up the chain.

You're right, OP hasn't responded to anything yet, which doesn't mean we can't discuss this particular topic without the OP being present. Depending on the needs of OP, I wouldn't call it simple because with your suggestion you'd be querying the clipboard until something relevant comes up, instead of listening to events instead (if, of course, OP's intent is to capture 'all' of them). Essentially making a sort of loop, wasting CPU cycles. If OP simply wants to capture files from drag and drop events, the code has also been provided above, which doesn't require calling native windows functions at all (well, not by the .NET programmer that is).

Your solution would be ideal if you're pressing a button or when a specific condition is reached (like for example on a mouse over event) that doesn't happen every x seconds. However, there's also the Clipboard object in C# which should be used instead of native windows functions:

<https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.clipboard.getfiledroplist?view=netcore-3.1>

We're still working with an abstraction called the .NET Framework / .NET Core, which should be preferred over native function calls to make sure that your code will continue to work, regardless of the version of Windows you're on. That was the whole point of the .NET Framework to begin with. You could be writing the entire application in C/C++ if you don't like .NET.

```
Code if(Clipboard.GetFileDropList().Count > 1) {  
    // Iterate over Clipboard.GetFileDropList()  
}
```

AddClipboardFormatListener would be ideal for modern development when it comes to placing a listener, you're very much correct about that, unless you want to support very old version of Windows (which would be the only reasonable use case of SetClipboardViewer).

```
Code [DllImport("user32.dll", SetLastError=true)]  
[return: MarshalAs(UnmanagedType.Bool)]  
static extern bool AddClipboardFormatListener(IntPtr hwnd);  
Using GetLastError if something goes wrong, of course:
```

<https://docs.microsoft.com/en-us/windows/win32/api/errhandlingapi/nf-errhandlingapi-getlasterror>

In terms of disposal:

```
Code [DllImport("user32.dll", SetLastError = true)]  
[return: MarshalAs(UnmanagedType.Bool)]  
static extern bool RemoveClipboardFormatListener(IntPtr hwnd);  
And the message to listen to would be 0x031D .
```

Example usage:

Code AddClipboardFormatListener(this.Handle);

You would have to override WndProc:

```
Code protected override void WndProc(ref Message m) {  
    base.WndProc(ref m);  
    // Listen to 0x031D  
}
```

And finally dispose of it like:

Code RemoveClipboardFormatListener(this.Handle);

It's, arguably, a lot easier than the code above that implements SetClipboardViewer .

But now, thanks to this discussion 'we' know how to listen to clipboard events on Windows 2000 and above and Windows Vista and above for newer functions, and also what to use depending on the scenario, using C#.

I would call this thread a success, with or without the OP.

He B TeMy - May 31, 2020, 4:45 pm

(May 31, 2020 09:03 AM)MavenGradle Wrote: [\[CITE\]](#) But now, thanks to this discussion 'we' know how to listen to clipboard events on Windows 2000 and above and Windows Vista and above for newer functions, and also what to use depending on the scenario, using C#.

I would call this thread a success, with or without the OP.

And that's all that matters, nice job in explaining ;)