Why does Clipboard.SetData put extra junk in the clipboard data? And how can I get it to stop?

devblogs.microsoft.com/oldnewthing/20181130-00

November 30, 2018

Raymond Chen

One of the ways of putting data on the clipboard is with the `System.Windows.Forms.Clipboard` object. There are methods for putting text on the clipboard in one of a few the standard text formats. And if you use the `Clipboard.SetData` method, you can place data on the clipboard with a custom format name. But when you use `Clipboard.SetData` to put text on the clipboard, the actual raw data on the clipboard contains extra stuff.

```
Clipboard.SetData("customText", "Hello, world!");
```

The actual raw bytes on the clipboard are

```
96 A7 9E FD 13 3B 70 43 A6 79 56 10 6B B2 88 FB
00 01 00 00 00 FF FF FF FF 01 00 00 00 00 00 00
00 06 01 00 00 00 0D 48 65 6C 6C 6F 2C 20 77 6F
72 6C 64 21 0B
```

The underlined bytes are the ASCII string `Hello, world!`, but what's the other junk?

The `Clipboard.SetData` method must serve two masters. One master is the Windows clipboard. Custom formats on the Windows clipboard are just binary blobs of data with no externally-imposed format. Any format for the data is by mutual agreement of the two parties using that custom format.

The other master is the CLR. If a C# program puts a serializable object on the clipboard, then it should be able to read it back as an object.

The `Clipboard.SetData` method takes two parameters. The first, a string, is the custom clipboard format name. The second, an object, is the object to put on the clipboard.

When putting an object on the clipboard, the CLR uses a `BinaryFormatter` to serialize the object to a binary blob, and puts that binary blob on the clipboard. When reading an object from the clipboard, takes the binary blob from the clipboard and uses a `BinaryFormatter` to deserialize the object back into a CLR object.

Okay, so that keeps the second master happy. But what about the first master? Suppose the native clipboard has some arbitrary binary blob. How do we recognize that it is an arbitrary binary blob, rather than a serialized CLR object? Because if we try to deserialize it as a CLR object, we'll get garbage.

The answer is that the clipboard puts a secret signal at the start of the binary blob. If the secret signal is present, then it assumes that the data represents a binary-formatted serialized CLR object. Otherwise, it assumes the data represents an arbitrary binary blob.

When you read data from the clipboard, and it turns out to be an arbitrary binary blob, the `Clipboard.GetData` method returns a `Stream` containing the raw binary blob.

Conversely, if you want to write a raw binary blob, you can pass a `Stream` to the the `Clipboard.SetData` method.

Okay, so now with some help from [MS-NRBF]: .NET Remoting: Binary Format Data Structure. we can parse the raw bytes:

```
magic prefix:
    96 A7 9E FD 13 3B 70 43 A6 79 56 10 6B B2 88 FB

SerializationHeaderRecord
    RecordTypeEnum: 00
    RootId: 01 00 00 00
    HeaderId: FF FF FF FF
    MajorVersion: 01 00 00 00
    MinorVersion: 00 00 00 00

RecordTypeEnum: 06 (BinaryObjectString)
ObjectId: 01 00 00 00
Length: 0D
UTF-8 data: 48 65 6C 6C 6F 2C 20 77 6F 72 6C 64 21

End of serialization: 0B
```

And to wrap things up, a table, because people like tables.

Operation with custom format	Format	
	Raw binary data	**CLR binary serialized data**
`SetData`	Pass `Stream`	Pass anything except `Stream`
`GetData`	Returns `Stream`	Returns anything except `Stream`

That wraps up CLR week for this year. The good news is that you made it almost all the way to the end of the year before I inflicted it upon you. The bad news is that the new year is coming up soon, so the threat of another CLR week returns more quickly.

Raymond Chen

Follow