Executing and then Deleting a DLL in c#

Asked 16 years, 2 months ago Modified 15 years, 7 months ago Viewed 3k times

2

I'm creating a self updating app where I have the majority of the code in a seperate DLL. It's command line and will eventually be run on Mono. I'm just trying to get this code to work in C# on windows at the command line.

How can I create a c# application that I can delete a supporting dll while its running?

```
AppDomain domain = AppDomain.CreateDomain("MyDomain");
ObjectHandle instance = domain.CreateInstance( "VersionUpdater.Core",
"VersionUpdater.Core.VersionInfo");
object unwrap = instance.Unwrap();
Console.WriteLine(((ICommand)unwrap).Run());
AppDomain.Unload(domain);
Console.ReadLine();
```

at the ReadLine the VersionUpdater.Core.dll is still locked from deletion

The ICommand interface is in VersionUpdater.Common.dll which is referenced by both the Commandline app and VersionUpdater.Core.dll

mono appdomain self-updating

Share
Improve this question
Follow

edited Oct 8, 2008 at 8:15

community wiki
2 revs
Scott Cowan

Why did you make this a wiki? You should have taken the rep. It is a good question. – GEOCHET Oct 7, 2008 at 17:16

oh shoot didn't know that I didn't get rep – Scott Cowan Oct 8, 2008 at 8:02

You've answered your own question here, surely. If you're updater and the thing you are updating share code, then both AppDomains will lock that code, so you can't delete it. The updater must not try to update its own code, that won't work. – Jeff Yates Oct 8, 2008 at 14:45

I thinking I'm going to write start and stop calls on the commandline app so I can wait for threads to finish instead of killing the process to update it. – Scott Cowan Oct 8, 2008 at 16:14

There may be a solution in using MEF or Mono Addins to do this – Scott Cowan Jun 1, 2009 at 15:19

4 Answers

Sorted by: Highest score (default)

6

The only way I've ever managed something similar is to have the DLL in a separate AppDomain to the assembly that is trying to delete it. I unload the other AppDomain and then delete the DLL from disk.

If you're looking for a way to perform the update, off the top of my head I would go for a stub exe that spawns the real AppDomain. Then, when that stub exe detects an update is to be applied, it quits the other AppDomain and then does the update magic.

EDIT: The updater cannot share DLLs with the thing it is updating, otherwise it will lock those DLLs and therefore prevent them from being deleted. I suspect this is why you still get an exception. The updater has to be standalone and not reliant on anything that the other AppDomain uses, and vice versa.

Share
Improve this answer
Follow

edited Oct 8, 2008 at 14:47

answered Oct 7, 2008 at 17:15
Jeff Yates
62.3k ● 20 ● 142 ● 192

Agreed, this is the only way I know of to do this. – GEOCHET Oct 7, 2008 at 17:16

I'll second this too. A lot of the newer applications I use (Paint.NET, FileZilla) have a seperate application fire up to perform the update which will call the uninstaller to remove the application and then reinstall the new one. In your case you can have it remove the DLL and then replace. – Dillie-O Oct 7, 2008 at 17:17

I agree - We are dealing with 2 aspects here. – Saif Khan Oct 7, 2008 at 17:37

sticking a proxy library in a seperate AppDomain still throws an Exception on deleting – Scott Cowan Oct 8, 2008 at 8:37

1

Unwrap will load the assembly of the object's type into the appdomain that calls it. One way around this is to create a type in your "base" assembly that calls command.run, then load that into your new appdomain. This way you never have to call unwrap on an object from a type in a different assembly, and you can delete the assembly on disk.

Share Improve this answer Follow

answered Oct 7, 2008 at 17:28
Philip Rieck
32.6k ● 11 ● 89 ● 99

When I built a self-updating app, I used the stub idea, but the stub was the app itself.

1

The app would start, look for updates. If it found an update, it would download a copy of the new app to temp storage, and then start it up (System.Diagnostics.Process.Start()) using a command-line option that said "you are being updated". Then the original exe exits.

The spawned exe starts up, sees that it is an update, and copies itself to the original app directory. It then starts the app from that new location. Then the spawned exe ends.

The newly started exe from the original app install location starts up - sees the temp file and deletes it. Then resumes normal execution.

Share

Improve this answer

Follow

answered Mar 8, 2009 at 3:42

community wiki
Cheeso

1

You can always use `MOVEFILE_DELAY_UNTIL_REBOOT` to delete on reboot. This is most likely the least hackey way todo this sort of thing, by hackey I ususally see things like; loading up new DLL's or injecting to explorer.exe even patching a system dll to get loaded into another process, etc...

MoveFileEx From MSDN;

> lpNewFileName [in, **optional**] The new name of the file or directory on the local computer.
>
> When *moving* a file, the destination can be on a different file system or volume. If the destination is on another drive, you must set the `MOVEFILE_COPY_ALLOWED` flag in dwFlags.
>
> When moving a directory, the destination must be on the same drive.
>
> If dwFlags specifies `MOVEFILE_DELAY_UNTIL_REBOOT` and lpNewFileName is **NULL**, *MoveFileEx* registers the lpExistingFileName file to be **deleted** when the system restarts. If lpExistingFileName refers to a directory, the system removes the directory at restart only if the directory is empty.

Share

Improve this answer

Follow

answered May 29, 2009 at 6:48

community wiki
RandomNickName42

wow thats super obscure, thanks. Of course it would be the same to check for the latest version on startup which is what I'm currently doing – Scott Cowan Jun 1, 2009 at 15:15