# Setting up private COM contexts to allow yourself to unload cleanly

devblogs.microsoft.com/oldnewthing/20191127-00

November 27, 2019

Raymond Chen

Last time, we saw how you could use the `CLSID_ ContextSwitcher` object to create a private COM context and register your object factory in that context. This allows you to disconnect all proxies to objects in that context, which is handy when circumstances such as service shutdown require you to unload a DLL prior to process termination, even though there may be outstanding references to its objects. Disconnecting the proxies forcibly releases those references, allowing you to unload.

This trick was originally intended for Windows NT services, but you can use it any other time you are required to unload your DLL while the host process continues to run. For example, your DLL may be a plug-in in another process, and that other process's plug-in model requires you to unload.[1] However, your plug-in displayed some UI, and a screen reader or other assistive technology tool may have an active `IAccessible` reference to one of your UI objects. Your plug-in created that UI, showed it to the user, and destroyed it, but while it was visible, an assistive technology tool managed to get an `IAccessible` for it, and the reference remains outstanding even though the UI is no longer visible. (Assistive technology tools do a lot of caching, so the `IAccessible` is just sitting in a cache somewhere, just in case your dialog reappears.)

To solve this problem, you can do what the Windows NT service does: Create a private `CLSID_ ContextSwitcher` in which all of your COM objects reside, and as part of your DLL's "prepare to be unloaded" steps, you can enter the context one last time in order to call `CoDisconnectContext` to disconnect all the proxies.

Disconnecting the proxies should cause all your objects to run down, and now you can destroy the empty context, and allow your DLL to be unloaded.

Creating those proxies to objects in the context is trickier using this pattern because that job is now on you. With Windows NT services, they register a COM object factory, and COM will use the factory to create objects. The people requesting the creation of the object are not in the context, so the natural COM infrastructure will create the proxy automatically.

In our scenario, however, we aren't registering a COM factory. Our UI object received a `WM_ GETOBJECT` message, and it needs to produce an `IAccessible` immediately.

So you enter the context, create the object, but now you have to figure out how to *take the object with you* when you exit the context (transforming it from a direct reference to a proxy), so you can return it to the caller.

I discussed some options some time ago. One is to use `RoGetAgileReference` to obtain an agile reference to your `IAccessible`. You can move this agile reference freely between contexts and apartments, so you can take that out of the context, then convert it back to an `IAccessible` to return to your caller.

Another option is to marshal the interface into a stream and take the stream with you out of the context, then reconstitute the interface from the stream once you're safely outside.

(I suspect the first way is implemented in terms of the second way.)

All this time, I've been talking about getting into and out of this context, but never actually showing how to do it. Next time, we'll look at the mechanics.

[1] The plug-in's model may not actually *require* you to unload. After all, you can force yourself to linger by doing a bonus `LoadLibrary` on yourself. But you want to go along with it, just to be a good citizen. Or perhaps you want to unload so that you can update the DLL. This is handy during development, since you can go to the host process, tell it to unload the extension, then rebuild and redeploy the extension, then ask the host process to reload the extension.

Raymond Chen

**Follow**