Crashing in COM after I call CoUninitialize, how can COM be running after it is uninitalized?

devblogs.microsoft.com/oldnewthing/20200129-00

January 29, 2020

Raymond Chen

A customer reported that they found a bug in shell32. The customer liaison forwarded the question to the shell team, with the caution that since he is currently on vacation, he hadn't validated the legitimacy of the report, or its quality or correctness.

The customer was kind enough to reduce the program to its essence, and that made zeroing in on the problem much easier.

```
#import "shell32.dll"

// Code in italics is wrong
void demonstrate_problem()
{
 auto hr = CoInitialize(nullptr);

  Shell32::IDispatchPtr shell_application("Shell.Application");

  ... use the shell_application to do stuff ...

  CoUninitialize();
}
```

The customer's analysis of the crash was rather lengthy, but hidden inside is this fragment:

> If I remove the call to `CoUninitalize`, then the crash disappears, but from what I understand, I'm doing everything by the rules. Once I `Release` all the COM pointers I own, I have the right to call `CoUninitialize`.

Do you see the problem?

This is another case of paying attention to when your destructors run.

The `shell_ application` object is a smart pointer object, so it will release the raw COM pointer at destruction. When does it destruct?

It destructs *after* the `CoUninitialize` call.

The customer belived that they had `Release` all of their COM pointers at the point they called `CoUninitialize`, but in fact they hadn't. There was an unreleased COM pointer inside the `shell_application` object, as well as other objects in the code I elided.

The fix is to ensure that everything is properly released before uninitializing COM. One way is to force the destruction of all relevant objects by introducing a nested scope:

```
void demonstrate_problem()
{
 auto hr = CoInitialize(nullptr);

 { // nested scope
  Shell32::IDispatchPtr shell_application("Shell.Application");

  ... use the shell_application to do stuff ...

 } // force smart objects to destruct
 CoUninitialize();
}
```

Another is to put the initialize of COM into its own RAII object, so that it destructs last.

```
void demonstrate_problem()
{
  CCoInitialize init;

  Shell32::IDispatchPtr shell_application("Shell.Application");

  ... use the shell_application to do stuff ...
  // CoUninitialize();
}
```

Raymond Chen

Follow