

microsoft.public.dotnet.languages.csharp: Re: Simple and fast Singleton pattern for ya.

Re: Simple and fast Singleton pattern for ya.

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2004-02/1343.html>

From: William Stacey (*staceywREMOVE_at_mvps.org*)

Date: 02/10/04

Date: Mon, 9 Feb 2004 19:01:07 -0500

Not exactly what happens here. This is not the same as the double checked issue. DCL uses an unsynchronized testing of the object reference. So the ref could be assigned by thread1 and then context switch, then thread2 could get that ref and try to use it before construction. This does not happen here because construction is completed before the interlock releases the lock. The actual creation is protected by a memory barrier using the "lock()". The assignment also creates a "write" barrier on the ref from doco I have seen. Only then is the flag set atomically. If we want to be really sure, we could also add a Thread.MemoryBarrier() instruction before we open the lock like so:

```
...
Thread.MemoryBarrier();
Interlocked.Exchange(ref created, 1);
...
```

```
--
William Stacey, MVP
"Jon Skeet [C# MVP]" <skeet@pobox.com> wrote in message
news:MPG.1a9201aa83d9b45d98a0a2@msnews.microsoft.com...
> William Stacey <staceywREMOVE@mvps.org> wrote:
> > Thanks for seeing that Jon. There was a small window there. Here is
the
> > fix to correct for that possibility.
> > We use a lock once, in the case where multiple threads actually try to
get
> > the first instance at the "same" time.
> > The first thread to get the lock, will create the instance. If multiple
> > threads happen to wait on the lock at same time before the instance is
> > created, they will "wake" up to see instance was created and return it.
Any
> > future threads will not hit the lock, but just do the quick
> > interlocked.compareexchange to test for created. So except for the
noted
> > exception on first use, no expensive locks are hit. Cheers!
>
> Nope, you've then got the same problem as the normal double-check lock
> algorithm - you could see the reference to the singleton before it's
> properly initialised.
>
> You basically need a memory barrier there *somewhere*.
>
> --
```

Re: Simple and fast Singleton pattern for ya.

microsoft.public.dotnet.languages.csharp: Re: Simple and fast Singleton pattern for ya.

> Jon Skeet - <skeet@pobox.com>
> <http://www.pobox.com/~skeet>
> If replying to the group, please do not mail me too